João Maria Mateus Palmeiro

Bachelor of Science

MevaL: A Visual Machine Learning Model Evaluation Tool for Financial Crime Detection

Dissertation submitted in partial fulfillment of the requirements for the degree of

Master of Science in
Analysis and Engineering of Big Data

Adviser:	David Polido, Data Visualization Engineer, Feedzai
Co-adviser:	Ludwig Krippahl, Assistant Professor, NOVA University Lisbon

Examination Committee

Chair:	Pedro Manuel Corrêa Calvente de Barahona, Full Professor, NOVA University Lisbon
Rapporteurs:	João Miguel da Costa Magalhães, Associate Professor, NOVA University Lisbon
	Ludwig Krippahl, Assistant Professor, NOVA University Lisbon

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

February, 2021

MevaL: A Visual Machine Learning Model Evaluation Tool for Financial Crime Detection

This document was created using the (pdf)LaTeX processor, based on the NOVAthesis template, developed at the Dep. Informática of FCT-NOVA by João M. Lourenço.

To Diana, João, Susana, and Tales de Mileto.

Acknowledgements

A data scientist, a data visualization engineer, and an *Alentejano* walk into a bar. After taking a seat, they start to see the bar menu. The data scientist complains, "Ah, unstructured drinks and missing prices, I can't choose!" The data visualization engineer protests, "Ah, there is no chart to help me decide!" Finally, the *Alentejano* says "Ah, *guardente* [322], please."

I believe this wannabe joke sums up one of my dreams. I dream of the day when all the things I deal with daily will be simple. Why? Simply to be able to help and contribute the best I can. This work is the first major threshold on this journey — it was the first time that I accomplished something so simple and so complex at the same time. So, I hope that the footprint of this venture will be big enough to prove useful to you as well.

However, the most beautiful part of this work cannot be summarized. The help and contribution of so many people magically enhanced the experience and the final result, in a way that I couldn't imagine initially. So, I would like to thank all these people symbolically — and also all open-source software developers for all the work that underpinned this one.

First of all, I would like to thank David Polido, Bruno Laraña, and Professor Ludwig Krippahl especially for all the support, guidance, and concrete contributions to this work. In a first-time effort like this, the possibility of working side by side with mentors who have so much to offer is simply incredible, both for the present and for the future.

To Artur Trindade, Beatriz Jorge, Daniela Burrinha, Liliana Fernandes, Pedro Bizarro, and Rita Costa, a heartfelt thanks for all the daily help, feedback, and friendship as a member of Feedzai's Data Visualization team.

To Alberto Ferreira, Alexandre Candeias, António Correia, Antonio Gonzalez, Francisco Fonseca, Hugo Ferreira, João Louro, Leonor Frazão, Ricardo Moreira, Ricardo Neves, and Stéphane Couvreur, a very special and profound thanks for all the availability and constant support in the development of this project, as well as for all the teachings about Data Science and Machine Learning.

To Roxana Attar and Poonam Angne, a huge thanks for giving MevaL an opportunity in their projects and for the collaboration that brought so many benefits to all parties.

To Bernardo Branco, Catarina Belém, João Bravo, João Caetano, Marco Sampaio, Maria

Inês Silva, and Pedro Saleiro, my gratitude, as a fellow member of the Research department, for helping to define an interesting path for this work and all the input and suggestions. Also, to Vladimir Balayan, a proud thanks for the countless conversations/discussions and for the tiny contributions that make all the difference as a friend and colleague.

To João Veiga and Samuel Pedro, an unconditional thanks for all the practical improvements and for the active role in transferring the final result to production.

To Feedzai, in general, a full-hearted thanks for this opportunity and the appreciation, in every sense, of my work.

Finally, I can only thank once again my parents, Diana Palmeiro e João Palmeiro, for welcoming me to life and helping me find my way, and Susana Calhau, my love and *sister-in-arms*.

This work stands on the shoulders of (my) giants and I hope I managed to live up to it.

"Why should we live with such hurry and waste of life?
We are determined to be starved before we are hungry."

— Henry David Thoreau, *Walden*

Abstract

Data Science and Machine Learning are two valuable allies to fight financial crime, the domain where Feedzai seeks to leverage its value proposition in support of its mission: to make banking and commerce safe. Data is at the core of both fields and this domain, so structuring instances for visual consumption provides an effective way of understanding the data and communicating insights.

The development of a solution for each project and use case requires a careful and effective Machine Learning Model Evaluation stage, as it is the major source of feedback before deployment. The tooling for this stage available at Feedzai can be improved, accelerated, visually supported, and diversified to enable data scientists to boost their daily work and the quality of the models.

In this work, I propose to collect and compile internal and external input, in terms of workflow and Model Evaluation, in a proposal hierarchically segmented by well-defined objectives and tasks, to instantiate the proposal in a Python package, and to iteratively validate the package with Feedzai's data scientists. Therefore, the first contribution is MevaL, a Python package for Model Evaluation with visual support, integrated into Feedzai's Data Science environment by design. In fact, MevaL is already being leveraged as a visualization package on two internal reporting projects that are serving some of Feedzai's major clients.

In addition to MevaL, the second contribution of this work is the Model Evaluation Topology developed to ensure clear communication and design of features.

Keywords: Financial Crime, Data Science, Data Visualization, Machine Learning, Model Evaluation, Python Package

Resumo

A Ciência de Dados e a Aprendizagem Automática [277] são duas valiosas aliadas no combate à criminalidade económico-financeira, o domínio em que a Feedzai procura potenciar a sua proposta de valor em prol da sua missão: tornar o sistema bancário e o comércio seguros. Além disso, os dados estão no centro das duas áreas e deste domínio. Assim, a estruturação visual dos mesmos fornece uma maneira eficaz de os entender e transmitir informação.

O desenvolvimento de uma solução para cada projeto e caso de uso requer um estágio cuidadoso e eficaz de Avaliação de Modelos de Aprendizagem Automática, pois este estágio coincide com a principal fonte de retorno (*feedback*) antes da implementação da solução. As ferramentas de Avaliação de Modelos disponíveis na Feedzai podem ser aprimoradas, aceleradas, suportadas visualmente e diversificadas para permitir que os cientistas de dados impulsionem o seu trabalho diário e a qualidade destes modelos.

Neste trabalho, proponho a recolha e compilação de informação interna e externa, em termos de fluxo de trabalho e Avaliação de Modelos, numa proposta hierarquicamente segmentada por objetivos e tarefas bem definidas, a instanciação desta proposta num pacote Python e a validação iterativa deste pacote em colaboração com os cientistas de dados da Feedzai. Posto isto, a primeira contribuição deste trabalho é o MevaL, um pacote Python para Avaliação de Modelos com suporte visual, integrado no ambiente de Ciência de Dados da Feedzai. Na verdade, o MevaL já está a ser utilizado como um pacote de visualização em dois projetos internos de preparação de relatórios automáticos para alguns dos principais clientes da Feedzai.

Além do MevaL, a segunda contribuição deste trabalho é a Topologia de Avaliação de Modelos desenvolvida para garantir uma comunicação clara e o *design* enquadrado das diferentes funcionalidades.

Palavras-chave: Criminalidade Económico-Financeira, Ciência de Dados, Visualização de Dados, Aprendizagem Automática, Avaliação de Modelos, Pacote Python

Contents

List of Figures

List of Tables

Listings

Glossary

acquirer	A bank or financial institution that processes card payments on behalf of merchants [321]
chargeback	A chargeback, in this context, is a charge that is returned to a card after a particular customer successfully disputes a transaction with his/her bank [156, 271]. In addition, it is assumed that this process is triggered by a fraudulent charge [156].
Customer Success	The department that works with Feedzai's clients to implement a DS-based project

Acronyms

a.k.a.	also known as
AI	Artificial Intelligence
AML	anti-money laundering
AP	average precision
API	application programming interface
AR	Alert Rate
AUC	Area Under the Curve
AUCPR	Area Under the Precision-Recall Curve
AUROC	Area Under the ROC Curve
AutoML	Automated Machine Learning
BC	binary classification
CLI	command-line interface
CSS	Cascading Style Sheets
CSV	comma-separated values
CV	cross-validation
DE	Domain Expert
DEV	Developer
DJL	Deep Java Library
DL	Deep Learning
DOM	Document Object Model
DS	Data Science
DSML	Data Science and Machine Learning
DSS	Data Science Studio
DT	Decision Tree
DV	Data Visualization
e2e	end-to-end
EDA	Exploratory Data Analysis

FI	Feature Importance
fintech	financial technology
FN	False Negative
FNR	False Negative Rate
FP	False Positive
FPR	False Positive Rate
GAM	Generalized Additive Model
GBDT	Gradient Boosting Decision Tree
GoG	Grammar of Graphics
GUI	graphical user interface
HTML	Hypertext Markup Language
IML	Interactive Machine Learning
JSON	JavaScript Object Notation
JVM	Java virtual machine
KM	Knowledge Management
k-NN	k-Nearest Neighbors
KS	Kolmogorov-Smirnov
LLOC	logical lines of code
LOC	lines of code
MCC	multi-class classification
MCC	Matthews Correlation Coefficient
MD	Model Decay
ML	Machine Learning
MVP	Minimum Viable Product
NN	neural network
OOP	object-oriented programming
OOTB	out-of-the-box
OOTL	out-of-the-loop
OSS	open-source software
PAIR	People + AI Research

pAUC	partial Area Under the Curve
PDP	Partial Dependence Plot
PNG	Portable Network Graphics
PoC	proof of concept
POS	point of sale
PPV	Positive Predictive Value
PR	Precision-Recall
PRD	product requirements document
PyPI	Python Package Index
RDF	Resource Description Framework
RF	Random Forest
ROC	Receiver Operating Characteristic
SDE	Score Distribution Estimation
SI	International System of Units
SLOC	source lines of code
SOM	Self-Organizing Map
SVG	Scalable Vector Graphics
t-SNE	t-distributed stochastic neighbor embedding
TN	True Negative
TNR	True Negative Rate
TP	True Positive
TPR	True Positive Rate
TT	Threshold Tuning
UAT	User Acceptance Testing
UDF	User-Defined Function
UE	Uncertainty Estimation
uFeatures	understandable features
UI	User Interface
USD	United States dollar
UX	User Experience
VPA	Visual Performance Analysis
WIT	What-If Tool
XAI	Explainable AI

Chapter 1

Introduction

1.1 Problem Statement and Motivation

Data Science (DS) and Machine Learning (ML) consist of a set of different tasks interspersed sequentially or in a loop, aimed at discovering actionable insights and/or creating models that allow the signal to be extracted from a dataset for use in future data without direct human intervention. However, within these tasks, there is a particularly interesting part for the data scientist since it is the first part where he/she gets some feedback on the state of the solution being developed — that part is called Model Evaluation.

Before deployment, it is only during this evaluation that the data scientist obtains quantitative and qualitative information on the suitability of the models developed to solve the intended problem. Although all parts are important, this part has special importance, given the insights that can be obtained and the incentive for new iterations. In other (more practical) words, usually, multiple ML models are methodically trained and evaluated in an exploratory process in order to identify the set of features, hyperparameters and other variables that allow the creation of the best possible model, within a time limit, for a given problem [108]. For Feedzai to continue to develop its DS-based value proposition in the field of financial technology (fintech) to fight financial crime, it is essential to ensure that data scientists have access to the appropriate tools to design and tune models for ever-changing challenges.

Currently, for Feedzai's data scientists to be able to fully analyze their ML models in a customized fashion, they have to create their own, sometimes one-shot, computational notebooks and/or scripts (single-purpose code). This recurring necessity is, in most cases, an impediment to deepening the Model Evaluation procedure due to lack of time or other resources (or to adapt it to something more standardized, shareable, and ready to communicate). Although Feedzai's data scientists may use some in-house standard capabilities for a first screening of Model Evaluation, there is no further tooling (except strict computational notebooks or scripts previously developed for a specific use case) to extend these benefits to consistent procedures and to test new ideas efficiently.

1.2 Main Goal and Objectives

1.2.1 Main Goal

The main goal of this master's thesis is to create a Python package to be integrated into Feedzai's DS environment and improve the capabilities currently available for Feedzai's data scientists to evaluate ML models with visual support. Developing ML models is a highly iterative process, composed of several pieces that must be tracked (data, both at the level of features, and at the level of samples used between training, validation, and testing, hyperparameters, algorithms, available computing resources, etc.), resulting in a demanding and difficult procedure in terms of comparing and evaluating models for each case [107].

This goal is backed by the following research question: how can we improve, accelerate, visually support, and diversify Feedzai's current Model Evaluation capabilities to enable data scientists to boost their daily work and the quality of their models? This question covers four fundamental pillars for the development of the proposed solution aiming at Model Evaluation and model-wise decision-making:

- *Improve* current features in terms of design or usability.
- *Accelerate* features to be customizable and easy-to-use.
- *Visually support* features with appropriate charts.
- *Diversify* features to allow other tasks to be performed out-of-the-box (OOTB).

1.2.2 Objectives and Constraints

The proposed solution should be aligned with the following targets:

- It should be integrable in Feedzai's DS environment.
- It should be a general-purpose solution, that is, it should work for all of Feedzai's use cases.
- It should be modular and extensible.
- It should be geared to the financial crime detection domain (the main implications of this domain are described in the Background Knowledge and Literature Review chapter, namely in the Feedzai section).
- It should be aimed at interactive Model Evaluation but also adaptable as a set of building blocks for the construction of input-agnostic computational notebooks (on-demand versus automation/reporting).

1.2.3 Success Criteria

In addition to the main goal and the derived objectives, there is a set of *metrics* established initially to facilitate the comparison of the work done with a possible definition of success. In this way, the success criteria are:

- The Python package enables the exploration of the created models from at least a new perspective that is not currently available OOTB in Pulse (Feedzai's end-to-end (e2e) ML platform).
- At least two action points of each of the defined and selected tasks/requirements are implemented and ready to use.
- All small, low-effort action points related to improvements of what Pulse currently offers in terms of Data Visualization (DV) for Model Evaluation are implemented (if the charts in question are implemented in the Python package).
- The Python package receives approval from the Product and Customer Success departments.
- The Python package is tested and validated by Feedzai's data scientists.
- All information about this project is documented in a central repository at Feedzai.
- The Python package features a modular architecture so that it is possible to add new functionalities efficiently.
- The Python package is complemented with a set of tutorials and documentation.
- The Python package is promoted internally on at least two occasions.
- The Python package is promoted externally on at least one occasion.

1.3 Contributions and Document Structure

The main contributions of this work are: (i) a Python package, MevaL (the explanation of this name and the project logo can be found in Appendix D), for visual Model Evaluation (standalone or for reporting); (ii) a Python package design adapted to Feedzai's DS environment and iteratively implemented with the help of data scientists and data visualization engineers; (iii) a topology to organize the different features for Model Evaluation. In order to fully describe them, this work is divided into the following chapters (there are also some appendices that can be checked at the end of the document):

1. Introduction (current chapter): this chapter is dedicated to inducing the problem and the proposed solution.
2. Background Knowledge and Literature Review: this chapter is dedicated to presenting the related and relevant work to characterize the general panorama, as well as Feedzai itself.

3. MevaL, a Python Package for Visual Model Evaluation: this chapter is dedicated to describing the details that allowed defining the scope of the Python package, as well as its functionalities.
4. User Validation and Case Study: this chapter is dedicated to reporting the feedback and conclusions of the user validation initiatives carried out, namely the user interviews where MevaL was demonstrated and evaluated qualitatively. It also contains the description of two internal projects at Feedzai that are taking advantage of MevaL.
5. Future Work and Conclusion: this chapter is dedicated to highlighting the main conclusions drawn throughout the development, as well as to project possible future iterations.

Chapter 2

Background Knowledge and Literature Review

2.1 Introduction

Since this work seeks to tackle the topic of ML Model Evaluation with visual support, via a Python package, it is essential not only to characterize Feedzai and its domain, but also to learn from related works. Thus, this chapter is divided into two main sections.

First, the Background Knowledge section is divided into four independent parts, each one designed to briefly describe Financial Crime, Data Science and Machine Learning, Data Visualization and, finally, Feedzai itself (especially from the point of view of the problem to be solved).

Second, in order to get to know the landscape of works in this area and the like, essential to enrich the in-house input collected and justify some of the design decisions for the developed Python package, the Literature Review section is divided as follows (this can be seen as its high-level scope):

1. Work related to Model Evaluation with Visual Support, that is, visual interfaces that help to evaluate models. In addition to papers, Python, R, and Java virtual machine (JVM) packages will also be analyzed (Python and R are two of the most used programming languages for DS and ML [281]), which, in a more practical way, also seek to contribute to the development of this area and the concrete tools available.

2. Work related to Model Evaluation in which the visual support is null (or clearly a secondary element) will be discussed. The main motivation for considering these works is to explore options such as statistical tests, in which the focus is on a specific

set of formulas and numbers, and to stimulate ideas to make this type of work more visual.

3. Work related to Data Visualization not applied to this specific context. Although these works do not have a direct application to Model Evaluation, they have characteristics that can be leveraged.

2.2 Background Knowledge

2.2.1 Financial Crime

Financial crime, also known as economic crime [74], can be viewed as improperly and unlawfully transferring property from one person or entity to another [333]. It is just an umbrella term that abstracts all the details of this illegal mapping that can take many forms. Credit card fraud, bank fraud, point of sale (POS) fraud, identity theft, and money laundering are just a few examples of financial crime.

According to PwC's 2020 Global Economic Crime and Fraud Survey [231], 47% of respondents stated that their companies had been victims of financial crime in the last 24 months, the second highest percentage in the past 20 years (only surpassed by 49% in 2018 [230]). Additionally, according to a study from Juniper Research, the value of online fraudulent transactions is expected to reach USD 25.6 billion by 2020 [276]. Since 2013, the percentage of organizations that experienced attempted and/or actual payment fraud has gone up [22], and in 2018 and 2019 it exceeded 80% [23]. Regarding occupational fraud, fraud committed against an organization by its people, of 2504 cases from 125 countries, the total losses exceeded USD 3.6 billion [24]. From an investment point of view, estimated spending on anti-money laundering (AML) technology solutions exceeded USD 8 billion in 2017 [292] (a proxy for an active problem). Regardless of the perspective, it is possible to see that the numbers associated with the different types of financial crime are overwhelming.

In the context of this work, there are three verticals to highlight: transaction fraud, money laundering, and account opening fraud.

Transaction fraud concerns the different flavors of detecting fraud as transactions come by in real-time [83]. For banks and acquirers [321], this generally means assessing the risk of prepaid, debit, credit, and ATM transactions. For merchants, this means typically assessing the risk of e-commerce transactions.

Regarding account opening fraud, there are two main types to consider: true name fraud and synthetic fraud [80]. In true name fraud, a criminal will try to take advantage of the personal information available to apply for an account in the victim's name. On the other hand, in synthetic fraud, the criminal will try to fabricate a new identity. So, for example, the criminal will attach a fake name to a real social security number.

Finally, money laundering is the process of integrating the proceeds obtained illegally into the legitimate mainstream of the financial system [287]. The main goal of money laundering is to eliminate the origin of the income obtained so that it looks similar to the *normal* money that circulates in the financial system. However, there is a fundamental difference in money laundering in relation to the types that have *fraud* in their name: the fraudsters already have the illicit money. This means that, unlike transaction and account opening fraud, in money laundering normally no one complains (about credit card transactions made without the cardholder's consent, for example).

A possible approach to fight and mitigate the harmful effects of financial crime is to leverage DS and ML [206] together with domain knowledge for the settlement of systems and communication flags for this global problem. Given the ever-changing scenario between crimes committed and their detection/prevention, as well as the number of transactions made daily by people as a proxy for the size of the global financial system, it is essential to work on how DS and ML derivatives are evaluated during its design against the references considered relevant. So, this master's thesis focuses precisely on evaluating the adequacy of the ML models, the main component responsible for answering whether a given item is associated with fraud or not, before its introduction into a system in action, and on the potential tooling improvements for Feedzai's data scientists to design these models according to the needs of different stakeholders.

2.2.2 Data Science and Machine Learning

DS is the intersection of different processes and methods for analyzing and understanding actual phenomena with data [102]. These processes and methods are mainly derived from Data Analysis, Statistics, and ML [329].

In order to structure such an effort, there are different methodologies that can be followed. In general, these are composed, in addition to understanding the problem in question, by one (or more) data acquisition phase(s), one (or more) modeling phase(s), and, finally, one (or more) deployment phase(s). An example is the Team DS Process, a methodology compiled by Microsoft [184]. It consists of five big steps (which are not necessarily sequential, as they can be revisited over time):

1. Business understanding
2. Data acquisition and understanding
3. Modeling
4. Deployment
5. Customer acceptance

The data acquisition and understanding stage is divided into at least three parts: data wrangling, exploration, and cleaning. Similarly, the modeling stage is divided into at least three blocks: feature engineering, model training, and Model Evaluation.

Another example of a (six-step) workflow is that of Uber that inspired Michelangelo, its e2e ML platform [108]:

1. Manage data
2. Train models
3. Evaluate models (also known as (a.k.a.) Model Evaluation)
4. Deploy models
5. Make predictions (online)
6. Monitor predictions (online)

Based on this general overview of the DS lifecycle, it is possible to get an idea of where ML fits (and that real-world problems are not singular e2e predictive problems [193]). First of all, ML is the field where different algorithms and statistical models are leveraged to detect patterns in data automatically, and then to take advantage of those patterns to predict future data, and/or to make other kinds of decisions under uncertainty [45, 202].

There are four main types of ML: supervised, semi-supervised, unsupervised, and reinforcement learning.

In supervised learning (the type of interest in this work), the main goal is to use a dataset, that is, a collection of N labeled instances $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^{N}$, to produce a model that takes a feature vector $\boldsymbol{x}$ as input and outputs some information (a *score* or a probability, for example) that allows deducing its label y [45]. Basically, it is intended to find the class, the real number or a more complex structure, that sums up (from a set of features to a label) an unknown instance according to the user's objective.

In unsupervised learning, on the other hand, the main goal is to use an unlabeled dataset, $\{\boldsymbol{x}_i\}_{i=1}^{N}$, to produce a model that takes a feature vector $\boldsymbol{x}$ as input and either transforms it into another vector or into a value that can be used to solve a practical problem such as clustering, dimensionality reduction, or anomaly detection [45]. A slightly more concrete example is the application of Isolation Forest to detect anomalies in card transactions since these anomalies can mean the fraudulent use of these same cards [171]. Isolation Forest is an algorithm that generates an ensemble of trees and allows the detection of abnormal instances (based on the dataset) by calculating the shortest average path lengths on the trees.

In semi-supervised learning, the main goal is similar to the goal of supervised learning in the sense that the dataset has both labeled and unlabeled instances, and that these instances can help the algorithm to produce a better model according to the metrics defined by the user [45].

Finally, reinforcement learning is a subfield with a lot of specificities. Generally speaking, reinforcement learning tries to solve problems where decision making is sequential, and the goal is long-term. At the same time, the types presented previously try to solve one-shot decision-making problems where instances are independent of one another and the predictions made in the past [45].

Within supervised learning, there are several problems, where classification and regression problems stand out. In both problems, the main goal is to learn a mapping from feature vectors $\boldsymbol{x}$ to labels y [202]. The difference is in the representation of the labels (and, of course, the algorithms to be used, the tasks to be solved, the metrics to be optimized, among other details):

- For binary classification problems, $y \in \{1, ..., C\}$ with $C = 2$. C is the number of classes.
- For multi-class classification (MCC) problems, $y \in \{1, ..., C\}$ with $C > 2$.
- For multi-label classification problems, $y \in \{1, ..., C\}$ but the class labels are not mutually exclusive.
- For regression problems, $y \in \mathbb{R}$ (y is continuous).

That said, imagining that the objective is to classify binary-labeled transactions as genuine or fraudulent, and assuming that all steps up to model training have been completed, it is necessary to divide the dataset into (at least) two subsets: the training set and the test set. The training set, as the name implies, will serve to train the model, that is, to provide the instances that an algorithm can use to learn to map, as best as possible, instances for the appropriate (according to the model) labels. Thus, in order to get an unbiased evaluation of the quality of the model, the test set, with instances "never seen before", is used — hence the utility of dividing the data into two subsets.

If the model is able to correctly map these instances to the expected labels, according to a metric and a pre-established minimum, then the model is said to generalize well and can be used for predictions. Generalization is fundamental because, at the end of the day, the interest is in predicting the labels of new instances, that is, it is important to ensure that the model works as a reliable function approximation of the true unknown mapping function (and not as a reliable function approximation of all the idiosyncrasies of a *small* sample) [202].

Other strategies for splitting the dataset can be considered, especially if there is an interest in evaluating different configurations (or hyperparameters) of ML models. There is a risk of *over-adapting* (or overfitting) the model to existing historical data by experimenting with different hyperparameters if the dataset is divided only into two parts. In this case, there is one more step, the choice of hyperparameters or validation, in addition to assessing the quality of the model, which makes it essential to have a separate set to *test* the various models with the different hyperparameters and, in the end, test the apparent best model in a separate set as described in the previous paragraph [67]. Thus, there are three strategies to highlight:

- The first strategy to consider is the division into three subsets: the train/validation/test split [67]. This strategy can be useful in a context where large amounts of data exist.
- The second one is called k-fold cross-validation (CV) and is one of many variants of the CV procedure [21] (a kind of resampling procedure). The main idea is to

consider k folds (or partitions) of training data, where $k-1$ folds are used for training and the rest for *testing* — something that can be particularly useful in contexts where the amount of data is not so large. A separate test set will later be used for the final evaluation of the model trained with all the data used during the CV procedure and the best hyperparameters [67, 87]. There is also a stratified variant of this algorithm, suitable for imbalanced (in relation to the class of each instance) datasets, which keeps the *original* ratio of classes in each fold constant.

- Finally, temporal CV is also an interesting option when working with time-series data. In the above strategy, it is assumed that the instances are independent and identically distributed, which contrasts with the correlation between instances close in time (autocorrelation) in time-series data [67]. Thus, in a simple way, the temporal CV can be seen as a k-fold CV strategy where the first k folds are used for training and the $(k+1)^{th}$ fold is used for testing (successive training sets are supersets of those that come before them).

A model is the product of a given (parametric or nonparametric) algorithm, its hyperparameters, and the data used. In order to assess the quality of such a model, performance metrics can be used. Once the model outputs a value between 0 and 1 (not necessarily a probability value [207]), it is necessary to define the classification/decision threshold (a kind of dataset cut-off point) that will define the output value or score from which a given prediction will be labeled positive. In some cases, the value 0.5 is considered as the classification threshold, but depending on the context, different values can help to produce better results.

In binary classification, when comparing the labels obtained using the model with the expected labels, it is possible to obtain a set of performance metrics from the confusion matrix [121]. The confusion matrix is a matrix that groups the labels obtained with the model according to each class and their correctness. Considering two classes, where $C = 1$ is the positive class and $C = 0$ is the negative one:

- If the actual class and the predicted class are $C = 1$, a True Positive (TP) is obtained.
- If the actual class is $C = 0$ and the predicted class is $C = 1$, a False Positive (FP) is obtained.
- If the actual class is $C = 1$ and the predicted class is $C = 0$, a False Negative (FN) is obtained.
- If the actual class and the predicted class are $C = 0$, a True Negative (TN) is obtained.

From the confusion categories, it is possible to calculate several metrics [226, 328], among which the following stand out:

- Accuracy is the fraction of labels that a model correctly predicted. This performance metric is not used in this work due to the *bias* caused by the large number of TNs in

imbalanced contexts. Formally,
$Accuracy = \frac{TP+TN}{TP+FP+FN+TN}$.

- Precision is the fraction of predicted positive labels that a model correctly predicted. Formally, $Precision = \frac{TP}{TP+FP}$. Precision is also known as Positive Predictive Value (PPV).

- Recall is the fraction of actual positive labels that a model correctly predicted. Formally, $Recall = \frac{TP}{TP+FN}$. Recall is also known as True Positive Rate (TPR) or Sensitivity.

- False Positive Rate (FPR) is the fraction of actual negative labels that a model incorrectly predicted. Formally, $FPR = \frac{FP}{FP+TN}$.

- F_1 Score is the harmonic mean (a kind of weighted average) of Precision and Recall [68]. Since the relative contribution of both metrics are equal, F_1 Score can be seen as a metric that tries to balance both Precision and Recall. Formally, $F_1\,Score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$.

- Complementing the metric above, it is possible to consider a more general performance metric, the F_β Score, with a positive real factor β that allows more weight to be attributed to one of the metrics that make up this one. Thus, the $F_{0.5}$ Score is a good alternative to give more weight to Precision, while the F_2 Score is a good choice for a similar effect on the Recall side [93]. Formally, $F_\beta\,Score = (1+\beta^2) \cdot \frac{Precision \cdot Recall}{(\beta^2 \cdot Precision) + Recall}$ [332].

- True Negative Rate (TNR) is the fraction of actual negative labels that a model correctly predicted. Formally, $TNR = \frac{TN}{TN+FP}$.

- Geometric Mean (for binary classification problems) is the square root of the product between Recall and TNR, that is, of the product between the class-wise sensitivity values [166]. As a performance metric for imbalanced settings, it tries, in a balanced way, to maximize accuracy in each of the classes (performance of the positive class versus performance of the negative class). Formally, $Geometric\,Mean = \sqrt{Recall \cdot TNR}$.

- Matthews Correlation Coefficient (MCC) is another useful performance metric for imbalanced contexts [93] since it is a metric that considers the four confusion categories, so that a value close to 1, its maximum, will only happen if the model, for a given classification threshold, produces good results for all the four categories (proportionally to the size of the data subset for each of the classes) [50]. Moreover, MCC is similar to the Pearson correlation coefficient in its interpretation. In this case, the value -1 indicates a model that always predicts the class opposite the true class and the value 1 reflects a perfect model (a value of 0 means that the model is no better than predicting classes at random). Formally, $MCC = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$.

In the simplest version, these metrics are count-based metrics, and all instances are treated equally [226]. If, for example, the amount of money associated with each instance is considered, these metrics become cost-based (or money-based) metrics, and all instances are treated differently, depending on the amount involved. In addition, for situations where performance in different subgroups of data is as important as overall performance, metrics are also calculated by *splitting* the dataset according to the unique values of interest in one or more breakdown features, in order to obtain an estimate for each of the subgroups of interest (filter-based metrics). Another relevant use is the paired or constrained use of two performance metrics (or business-related metrics). In addition to checking the individual values of various metrics for a given threshold, the value of, for example, Recall for a pre-defined FPR value is also directly determined (such as Recall at 5% FPR, for example).

There is also a *sliding* or *visual metric* to consider: the Receiver Operating Characteristic (ROC) curve. This curve is a Recall versus FPR line that shows the performance of a model considering different classification thresholds. To transform it into a single number, the area under this curve (integral) is computed. This (threshold-independent) metric is called Area Under the ROC Curve (AUROC) and ranges from 0 to 1. The trapezoidal rule can be used to calculate an approximation of the integral [98]. It is also possible to consider the AUROC as the probability that an instance from the positive class has a higher prediction score than an instance from the negative class [98]. In this way, the AUROC can be obtained by sorting the prediction scores in ascending order and using the following formula: $AUROC = \frac{S_0 - n_0(n_0+1)/2}{n_0 n_1}$, where n_0 and n_1 are the numbers of instances of the positive and negative classes, respectively, and S_0 is the sum of ranks of positive instances. Similarly, the Precision-Recall (PR) curve is a Precision versus Recall line that shows the performance of a model considering different classification thresholds. It can also be summarized in a single number, the Area Under the Precision-Recall Curve (AUCPR) [94], through the trapezoidal rule or the average precision (AP) formula:

$$AP = \sum_n (R_n - R_{n-1}) P_n \tag{2.1}$$

where P_n and R_n are the Precision and Recall values at the n-th classification threshold [217, 320]. AUCPR can be used as an alternative to AUROC for imbalanced data, since TNs (usually the majority) are not used to calculate the metrics that make up the PR curve. Thus, this number does not overshadows the effects caused by changes in the other confusion categories [57, 94].

In the fraud domain, looking at the entire ROC curve may not be particularly useful — it is more interesting to look at the partial ROC curve [347]. Since business decisions are usually based on the impact generated on good/legitimate customers, it is important to consider FPR, that is, the rate of genuine transactions blocked, so as not to harm legitimate customers. Bearing in mind that the ROC curve is a chart of the Recall against the FPR, checking the partial curve allows to answer the question: *at this given FPR, how* *much of the fraud am I capturing?* Basically, the higher the Recall for a given FPR previously defined depending on the use case's constraints (a small value up to 5% in most cases), the better the model. At the same time, with a (Artificial Intelligence (AI)-based, rules-based or hybrid) financial crime detection system in mind, it is also important to mitigate the number of FPs, to be as low as possible, in order to allocate fraud analysts (or investigators who check these kind of alerts by calling injured cardholders, for example) to ideally control/check only potentially fraudulent events, having a greater bandwidth for this type of activity [59].

Other relevant metrics to consider are the (number of) Alerts ($Alerts = TP + FP$) and the Alert Rate (AR), that is, the percentage of instances that produced an alert ($AR = \frac{Alerts}{TP+FP+FN+TN}$).

Furthermore, instead of considering a *hard* decision line through a classification threshold that separates the instances classified as negative and positive, some cases consider the division of this spectrum into three different bands. In the middle is the uncertainty band, that is, the band where the instances whose score (obtained from the ML model) is between the lowest scores, close to 0, which reflect genuine transactions and higher scores, close to 1, which reflect fraudulent transactions, are found. This uncertainty band is, in a way, associated with the classification threshold, since it is this value that distinguishes a negative instance from a positive one. Thus, in a variable way for each context, a range of values can be defined around the classification threshold that will correspond to the scores that the ML model is not so sure about the respective class, and that will correspond to instances to be reviewed, that is, with the final label to be decided. In this way, these instances imply that a fraud analyst must manually analyze and catalog them. Given the limited throughput of analysts, it may be necessary to tune the number of transactions that should be seen by them and consider comparing the Queue/Review Rate [30] and the (automatic) Decline Rate instead of the AR (where this separation does not exist). Based on the example in Figure 2.1, instances with scores between 0 and 0.4 would automatically be classified as *Non-Fraud*, instances with scores between 0.4 and 0.6 would need to be reviewed, and instances with scores between 0.6 and 1 would automatically be classified as *Fraud*.

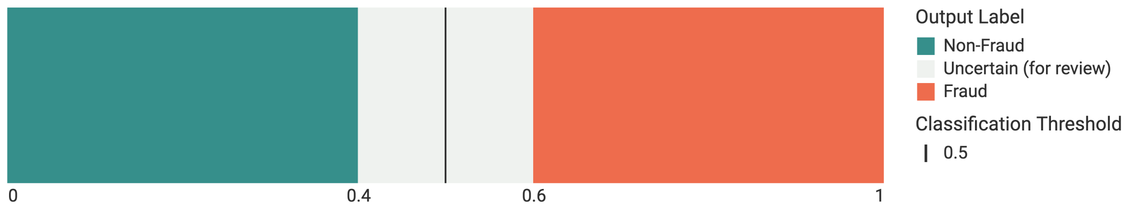


Figure 2.1: Output label according to predefined score ranges.

2.2.3 Data Visualization

Data is at the core of DS and ML — and, more relevant than that, it composes the pieces that can be used to understand financial crime from a practical point of view: the transactions. Structuring instances so that they are efficiently consumable by our visual system [144] provides an effective way of understanding the data and communicating insights. In other words, to make the data visual, it needs to be converted in a systematic and logical way into the visual elements (from the coordinate system to the axes, colors, and even the title) that constitute what is usually called a chart/graph/plot [338].

From a technical but also a design point of view, DV is responsible for *how* data is presented. DS is more comprehensive in the way it uses data, and ML is just a highly scalable mapping that helps with what is most important: exposing and leveraging patterns [122]. There is data throughout all the pipeline, from the original and raw instances to those outputs that have been digested. So, using DV techniques seems natural in this context.

However, DV is not just about creating a chart to represent a data table in a more consumable way by humans. DV is the discipline that explores the best way to transmit the desired information. In a simple way, DV starts the moment someone takes a raw table full of numbers — it ranges from how this table can be improved to the creation of highly customized charts.

In addition to the widespread idea that DV is a great vehicle for data, there are also some aspects that can prove decisive in helping a person searching for latent information in a dataset, and drive certain decisions and changes [182]. First, DV is quite flexible and can accommodate exploratory or explanatory scenarios [141], as well as be unfolded in charts for quick interpretation or tailored for reflective analysis [181]. Furthermore, a well-designed visualization can (and sometimes must) be combined with textual elements, with visual elements helping to develop an overview of the data, while the text provides precise and analytical information [144].

DV can also be the perfect tool for easy and clear detection of outliers, as these instances will stand out significantly from the others [144]. More generally, the reason for creating a visualization may be to see data in context, something more difficult to achieve through a simple table, for example [51].

In the same perspective of enhancing the image created by different visual elements, it is also possible to leverage principles and maxims such as Hick's law, the aesthetic–usability effect for creating charts, and Gestalt rules. Hick's law says, briefly, that the time to decide increases with the number and complexity of the options [157, 342], while the aesthetic–usability effect associates more aesthetically pleasing design and usability [341]. Gestalt rules are a set of rules that summarize strong inferences that humans make about the relationships between visual elements [104]. As an example, the proximity rule comes down to the apparent relationship between spatially close elements, and the common fate rule comes down to the apparent unitary relationship between elements that share the same direction of movement.

In general, the importance and usefulness of Data Visualization make it one of the fields that is definitely exploited in the context of this work, as it will be further explored starting from the Literature Review section.

2.2.4 Feedzai

Feedzai seeks, on a daily basis, to fulfill its mission by leveraging large amounts of data, designed and operationalized visualizations, DS and ML [78]: to make banking and commerce safe [76]. Through the developed fraud prevention and AML products, and their use by different financial institutions and merchants, such as Cuscal, Credorax, and First Data [79], Feedzai helps to manage risk and improve customer experience. To put it simply, Feedzai's job is to identify financial fraud.

In addition to different types of clients, Feedzai also acts in different use cases. The main ones are account opening, transaction fraud, and AML. Feedzai's software can be positioned at different points within a financial transaction: at the payment processor, at the issuing bank, or at the merchant. Also, this platform is divided into two major components [77]:

- For data scientists, there is Pulse (also known as ML Engine) where they can build a customized e2e ML pipeline (enabling big data preparation, and large-scale model training and serving [107], for example). It can also be seen as an AI-based fraud detection system [59].
- For fraud analysts, there is Risk Studio where they can gain deeper insights into financial crime and manage events.

For each project, there is a DS methodology, the DS Loop, which is followed more or less closely. First, this methodology is divided into two main phases: the offline phase (that takes place in the DS environment) and the online phase (that takes place in the streaming-oriented production environment). Within the first one, the following steps are followed (which can be repeated, hence the *Loop*):

1. Data cleaning
2. Data validation
3. Data exploration
4. Feature engineering
5. Model building
6. Model evaluation and comparison

After this phase, one or more models are deployed (a project/dataset does not necessarily correspond to a single model, as there are cases where there are different models for different regions, for example [223]), and the real action in the production environment begins. Here the essential tasks are to monitor the model(s) and monitor data changes (runtime monitoring). Since it is normal that, over time, the reality changes, it is also necessary to consider the transition from the online environment to the offline environment and retrain the model(s).

In addition, Pulse, the tool for building e2e ML pipelines, currently enables ML at scale (along with other products and expertise, for example). In terms of Model Evaluation, its main native features are:

- Two-way model comparison (performance and feature importance-wise).
- ROC curve, PR curve, and other two-dimensional plottable metrics.
- Performance metrics analysis (including custom algebraic metrics and an adjustable classification threshold with *real-time* feedback).
- Breakdowns (a.k.a. data slicing or subgroup discovery).
- Business and DS performance-based many-to-many model comparison.
- Prediction score distribution chart.
- Feature importance (generated agnostically to the model used).

Complementing its functionalities, it is also possible to use JupyterLab [140] (computational notebooks) in an integrated fashion, bringing added flexibility to the DS environment.

Feedzai mainly deals with binary classification problems (the positive class is usually the fraudulent one and the negative class is the legitimate one) and with considerably imbalanced (in terms of class frequency and interest in the minority class) big (and in many cases high-dimensional) data [221], as there are many more genuine transactions than fraudulent ones. Table 2.1 has some examples of summary statistics from real-world datasets used in previous articles from Feedzai's Research department. A related and particularly challenging phenomenon is that of overlapping classes [47]. On the one hand, fraudsters try to emulate the behavior considered legitimate by a cardholder, for example, while at times the behavior of a genuine cardholder, for example, can seem anomalous or illicit.

However, the fraud concept (or labeling logic), that is, the characteristics that define the fraudulent label, varies from client to client. Regarding the domain, it is an adversarial one since fraudsters are quick to adapt and evolve. Also, there are often few/limited "ground truth" labels, and concept drift (due to the non-stationary distribution of the data) is a non-trivial problem to face [2, 223]. This concept is usually applied in cases where the data shows different behaviors between the (offline) training and test sets, but it is generalizable to a continuous change across time (the behavior of customers and fraudsters changes over time [47]).

An example of a (real) fraud concept [174] is that of the IEEE-CIS Fraud Detection Kaggle competition datasets [119] (they are comprised of online transactions). In this

Table 2.1: Summary statistics for real-world (historical) datasets used in previous articles from Feedzai [39, 223]. Datasets A and B are divided into three and two regions, respectively [223]. These datasets comprise periods between approximately seven and ten months. Datasets C and D have fraudulent to legitimate instances ratios of 1:200 and 1:7000, respectively [39].

Dataset	# Features	# Transactions (Instances)
A1	213	1,046,482
A2	213	2,667,548
A3	213	4,945,509
B1	279	4,401,807
B2	279	9,229,013
C	19	$\approx 1,000,000,000$
D	59	$\approx 4,000,000,000$

case, the fraudulent labels correspond, in a nutshell, to the transactions that originated a chargeback [156, 271] on a card and to the transactions subsequent to this reported one if the user account, email address and/or billing address match. As for the remaining transactions, these are considered legitimate after 120 days. On the other hand, there may be unlawful activity that is not reported, simply because the cardholder did not notice the problem or missed the claim period, for example. Thus, some fraudulent transactions can be labeled as legitimate, but will go unnoticed (in this specific example, it is assumed that this scenario corresponds to a negligible portion).

Based on the above fraud concept, it is also possible to see that this detection problem differs from a more conventional classification one because of the way in which labels are made available. Although there is variability from case to case, and the associated workflow is not entirely linear and varies with the timing of a given project/model lifecycle (one thing is to start a project from a historical dataset, another is to maintain a project that requires model retraining with new transactions/labels, for example), a common scenario is one in which, in the first phase, only a limited set of supervised labels is accessible [59] (obtained with the help of fraud analysts and/or previous systems). After that, over time, most of the new labels will appear after a variable period of time, depending on whether or not there is a customer complaint about unauthorized transactions, for example (even considering a set of transactions annotated by fraud analysts that can be interpreted as *immediate feedback* labels compared to *delayed* labels [59, 223]). In a nutshell, the vast majority will not be obtained immediately after a fraudulent or legitimate event, as there is a maturation/latency period that needs to be respected given the functioning of the financial system (plus the limited human throughput and its associated cost). Moreover, given the concept-drifting environment [59, 223] in which this problem occurs, greater caution is needed, as in the retraining of ML models throughout a project, given the gap between the labels and the present moment.

An important note to keep in mind is that although there is a time variable associated with each transaction that makes up the typical datasets that Feedzai works with, the problem that Feedzai tries to solve is binary classification and not time-series forecasting (even that it is important to respect this idea of temporal sequence when training, validating and testing models, for example).

2.3 Literature Review

2.3.1 Model Evaluation with Visual Support

From improving the confusion matrix (pre-2016 papers) to visual tools tailored to model auditing and Deep Learning (DL).

In this section, there are some characteristics that make it possible to group the literature of visual tools for ML Model Evaluation into a couple of *fuzzy groups*. These characteristics are mainly in terms of (1) granularity, (2) ML problems, (3) features (especially if it is necessary to consider human-understandable features (uFeatures)/semantic features or not), (4) number of models, (5) tracking of changes, (6) readiness, (7) development context, (8) target group, (9) model availability, and (10) open-sourcing (a summary can be found in Appendix A). In addition, Figure 2.15 contains a coauthoring network based on the papers addressed in the next sections.

Throughout the text, some extra details are also highlighted, such as agnosticism/specificity (some of the tools are model-agnostic, and others are specific to Gradient Boosting or Random Forest (RF) models, for example), data types, scalability, and (explicit) support for imbalanced data (the typical scenario of the financial crime detection context).

2.3.1.1 Historical Perspective and First Look

The reference points presented give a first glimpse of the complexity of this field. However, the challenge of supporting data scientists (or users) in Model Evaluation is not a recent one (in *ML time*). In 1998, a scarcely concrete attempt appeared for the visualization of decision tables [28], while in 2002, the idea of adapting shaded similarity matrices, used in visual cluster analysis, in the visualization of the predicted labels of simple classification algorithms (Decision Tree (DT) and k-Nearest Neighbors (k-NN)) came up [307]. These ideas had no repercussions, and it was only years later that more concrete and more carefully grounded proposals began to emerge.

There are different papers that, based on a common motivation — to provide a way for data scientists and/or domain experts to understand their models and to envision actions that can bring positive iterations —, try to solve distinct ML problems. For this specific case, the focus will be on the visual tools proposed for binary classification, although there may be interesting ideas to adapt from the literature focused on MCC. Some works related to other problems will also briefly be considered to properly define the current landscape.

In the most basic way, the evaluation of binary classification models is done through one or more metrics derived from a confusion matrix. This form of evaluation compacts the performance of a model to one or more numbers, as if they were *averages*, which can be used in the decision-making process and in the comparison of different models. This perspective, at Feedzai, serves only to perform a first screening of the models and to have the first impression of them: the real evaluation begins after that. In a pragmatic way, all works in binary classification seek to provide frameworks and/or tools for this difficult part — this does not mean that all papers discard the more trivial part.

The Model Diagnostics workflow [147] is, in terms of positioning in the DS lifecycle, an example of a tool that aims to complement Model Evaluation after the creation of a model, similar to what is intended with the Python package. The Statistical Summary View, for example, provides a general overview of the performance of a given model before presenting the two views they developed for binary classification with sparse binary data. The second view, the Explanation Explorer, enables the data scientist/domain expert to explore different instance subgroups explained by different feature sets, as well as various statistics (these explanations consist of the features needed to be removed to change the predicted label of instances). The third one, the Item Level Inspector, shares a way to compare instances explained by a particular feature set in a matrix (each row represents a unique feature vector pattern while columns represent features) and to check how labels can be separated. The input data type is substantially different from that used at Feedzai (which invalidates the use of this solution as it is). Moreover, this solution has a very high computational cost. However, the use of odds ratio (and its plotting) is an interesting idea to detect whether an explanation (in more general terms, if a certain way of dividing the data through a predetermined logic) describes a consistent instance subgroup or if it describes a random subgroup. The odds ratio compares the number of positive and negative instances in a given subgroup with the rest of the dataset.

In an attempt to iterate the matrix representation and the information contained in the *traditional* confusion matrix, some papers highlight the weaknesses of it and propose alternative visualizations.

The confusion wheel, a chord diagram to show sample-class probabilities as histograms colored by classification outcomes, is the first example [8]. This radial proposal is integrated in a set of interconnected views: in addition to the confusion wheel, the feature analysis view allows to check feature distributions among selected instance subgroups, separated by their results and ranked by a separation measure, and different histograms and scatterplots allows to visually check the separability of selected true and false classified instances by one or two features. There is also a *traditional* confusion matrix augmented with histograms of instance probabilities in the respective rows and columns; moreover, the size of the squares that make up the cells is proportional to the number of instances. An important point is a concern with imbalanced data, even if the focus of the solution is MCC.

On the other hand, Squares is a visual interface that enables the user to check the performance of several classes in MCC problems [248]. Squares represents each class in a color-coded column. Each column contains a vertical axis, each one of them aligned with the leftmost axis indicating the prediction score range, annotated below by the corresponding class name and optional summary statistics. Instances predicted as an axis' corresponding class are positioned as boxes on the right side, while boxes on the left side represent instances labeled as an axis' corresponding class but predicted incorrectly as a different class. Using parallel coordinates, Squares reveals prediction scores for an instance across all classes when a user hovers or clicks on a box in the visualization. Also, Squares summarizes confusion information via a sparkline above each axis, displaying the parallel coordinates of all instances labeled as the corresponding class. Although it is optimized for MCC, the various parts of this visualization, as well as the encodings used, are interesting for possible adaptations for binary classification.

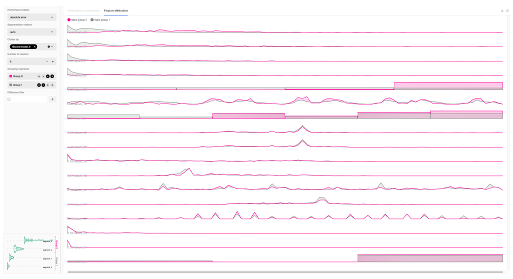
Finally, ModelTracker [12] is a kind of *father* of visual debugging tools due to the high number of citations considering this specific field. Its paper introduces an interface that combines overall model performance with instance-level inspection. The main idea is to arrange all instances in a visualization and, through various encodings, convey information about a model's binary classification results. ModelTracker also emphasizes prediction score changes on individual instances from one iteration to another. It displays the magnitude of score changes via directed arcs from an instance's previous score location to its current score location. Arcs are only displayed on instances whose predicted label changed from the previous iteration. Although the User Interface (UI) is quite outdated and the way the correlation is shown seems to be useful only in cases where the correlation is very high, the instance-level exploration and the encodings highlighting the differences between iterations are important steps to keep in mind.

2.3.1.2 Model-Agnostic Evaluation

As for the duality of model-specific and model-agnostic, there is a set of tools for each of the verticals. In the case of model-specific tools, papers in which the algorithms are also used at Feedzai will be considered. In addition, all papers presented so far are model-agnostic.

First of all, Prospector relies on one of the agnostic interpretability methods, the Partial Dependence Plot (PDP) [85], to help understand how features affect predictions overall [148]. In addition, it supports localized inspection that helps to understand how and why specific instances are predicted as they are, as well as tweaking feature values and seeing how the prediction responds. Prospector relies on partial dependence for one input feature at a time, but this approach relies on the orthogonality of input features. However, in real-world data, this is not often the case, as features may be correlated. Prospector can only model changes along one axis at a time as it cannot take correlations or influences between features into account. However, it is necessary to consider that this tool was developed in the context of healthcare and introduces small improvements like the histograms under each PDP (for a user to have a sense of the distribution of the data and in which areas the results may not be as reliable due to the lack of representativeness).

In 2018 and 2019, some of the most interesting papers emerged combining model-agnostic evaluation and visual support. Manifold [349], from Uber, presents a proposal in the original paper and, over time, the idea has matured, and today it is possible to find the current open-source version with different functionalities [168] (Figure 2.2). In the paper, Manifold consists of a model comparison overview that provides a visual comparison between model pairs using small multiples, and a local feature interpreter view that reveals a feature-wise comparison between user-defined subsets and provides a similarity measure of feature distributions. In the open-source version, the feature interpreter view, now called as feature attribution view, maintains its purpose. However, the model comparison overview, now called as performance comparison view, changed quite a bit. Manifold now uses the k-Means (clustering) algorithm to break prediction data into k segments/clusters based on performance similarity in order to enable the user to compare two models in different parts of the data (it also allows an interactive data slicing strategy using feature values selected manually, for example). On the other hand, in this version, there is a new view as well — the geo(spatial) feature view. Here it is possible to see the geo features displayed on a map and realize if there is a spatial pattern according to the selected segments or not, for example. The first impression received at Feedzai was that the constitution of the segments is not entirely clear to interpret. In addition, although the geo feature view is a unique proposal with a lot of potential, the current version currently implemented is still not reliable. Using the Python package made available in early 2020 [168], it is only possible to see the points of each segment on a map with a very low resolution, which prevents the true usefulness of this idea (and the documentation about this functionality is very short). However, the idea of using geo features to transmit other types of information to the user is something to consider, in addition to the fact that Manifold continues to be further developed by Uber.


(a) Manifold's feature attribution view.

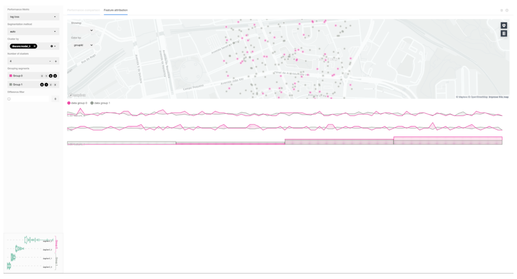
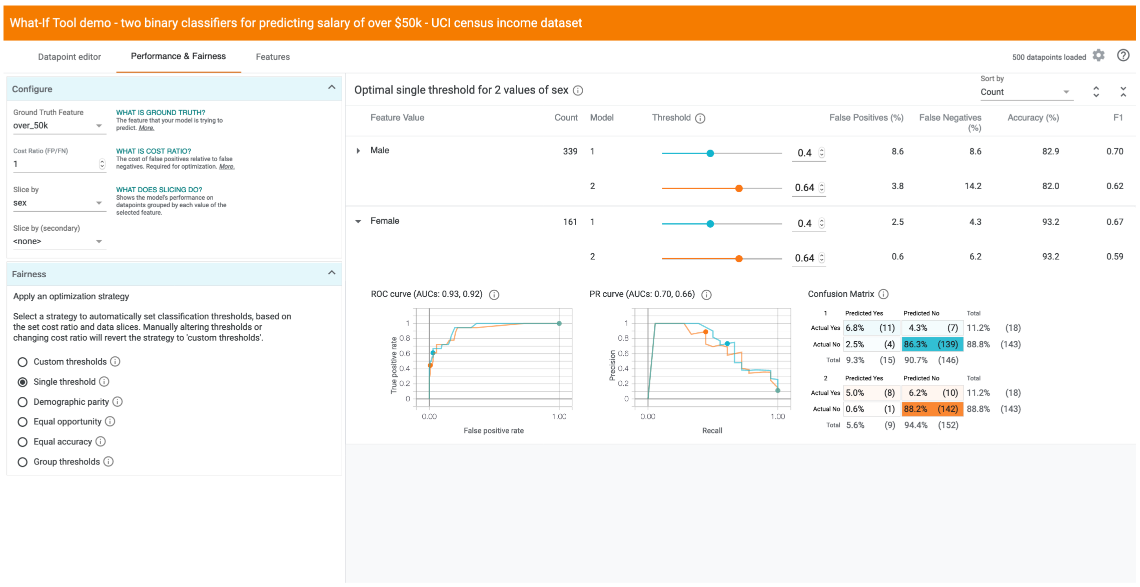

(b) Manifold's geo feature view.

Figure 2.2: Manifold web interface [168, 349].

Moreover, Manifold is part of the Michelangelo ecosystem or, in other words, of Uber's e2e ML scalable platform (unfortunately, Michelangelo is not available as open-source software (OSS)) [107–109, 262]. Excluding the existing possibilities when using the flexible Jupyter Notebook environment with Python [108, 285], Michelangelo (at least the version presented in 2017 [108] and updated in 2018 [107]) has three different tabs with information to evaluate/compare each (regression or binary classification) model trained (each model is stored as a versioned object in a Apache Cassandra database and the respective metadata can be accessed through the UI or programmatically) [108]. The first tab (for binary classification) is dedicated to performance and contains the ROC curve chart, the PR curve chart, and the probability calibration/reliability chart. An interesting space-saving detail of these charts is the bet on the positioning of the axis titles inside the charts (on the right side of the vertical axis and above the horizontal axis). At first glance, the overlap of the curve and the title does not appear to be harmful as the curve gives an overview and the user can traverse the curve to obtain more information about each threshold-driven point via tooltips. In addition, it also shows the confusion matrix and some scaled-up number [82] widgets for certain performance metrics (or a table for the training and test sets [107]), such as Area Under the Curve (AUC) — and the user can choose the classification threshold through an interactive slider [108]. In the second tab, there are some model-specific visualizations for tree-based models (apparently developed with MLlib [183], Spark's ML package) [108]. At the top, there is a Feature Importance (FI) (the importance of each feature is relative to each tree) heatmap across the various trees (the trees correspond to the columns and the features to the rows) — this matrix also helps to analyze the relative importance of each tree for the overall model. At the bottom, Michelangelo shows an individual tree inspector (the tree can be selected from the heat map), a kind of flowchart or tree diagram, where it is possible to depict the triggered path based on an instance whose features are defined by the user through a data generator [107, 108]. Finally, in the third tab, there are some charts and summary statistics about the features (or a FI vertical bar chart and a table with summary statistics and distribution sparkbars [107]). Namely, it is possible to find distribution histograms, FI values, PDPs and a three-dimensional chart for feature interactions (basically, it is a two-way PDP for two features picked by the user) [108].

In terms of model interpretability, RuleMatrix [187] uses a surrogate model that generates a list of rules (using the Scalable Bayesian Rule List algorithm) as the first step in the analysis of a given model. It then leverages a matrix-based visualization of these extracted rules to help users understand, explore and validate the knowledge learned by the original black-box model. It also allows the user to configure various options, filter data, check the original data, and compare rules in terms of fidelity and evidence. Fidelity is related to the error between the rule and the model, and evidence is related to the error between the model and real data. The scalability of this tool is a problem, but the openness to address this issue is noteworthy.

From Google's People + AI Research (PAIR) team, there's What-If Tool (WIT) [314]. With WIT (Figure 2.3), it is possible to test performance in hypothetical situations using instance-level counterfactuals [305], analyze the importance and summaries of different features (it is possible to explore the original data), visualize model behavior across two models at the same time and multiple instance subgroups, adjust the classification threshold, and test algorithmic fairness constraints (only available for binary classification [312]). Originally, the tool was only really OOTB for the TensorFlow production ecosystem. However, due to the active development of the tool since its launch, it is possible, considering version 1.7.0 [313], to use WIT on Google's AI Platform [251] (in addition to computational notebooks and together with TensorBoard, its *parent* tool), create custom prediction functions to encapsulate models created with other Python frameworks/libraries, leverage various Explainable AI (XAI) methods, such as LIME and SHAP, as well as learning about the tool with the various materials now available. As it runs in the browser [312], there is still a plateau for tabular data due to memory constraints and the number of possible charts, although the approximate value of 100,000 [314] (instances), in theory, is promising for many applications. Although a larger number would be positive, mainly for DL applications, the user can, however, promote different strategies [131] to check several samples, one at a time, for example (leveraging computational notebooks).



What-If Tool demo - two binary classifiers for predicting salary of over $50k - UCI census income dataset
Datapoint editor
Performance & Fairness
Features
500 datapoints loaded
Configure
Fairness
ROC curve (AUCs: 0.93, 0.92)
True positive rate
False positive rate
PR curve (AUCs: 0.70, 0.66)
Precision
Recall

Ground Truth Feature: over_50k
WHAT IS GROUND TRUTH? The feature that your model is trying to predict. More.
Cost Ratio (FP/FN): 1
WHAT IS COST RATIO? The cost of false positives relative to false negatives. Required for optimization. More.
Slice by: sex
WHAT DOES SLICING DO? Shows the model's performance on datapoints grouped by each value of the selected feature.
Slice by (secondary): <none>

Apply an optimization strategy

Select a strategy to automatically set classification thresholds, based on the set cost ratio and data slices. Manually altering thresholds or changing cost ratio will revert the strategy to 'custom thresholds'.

○ Custom thresholds
● Single threshold
○ Demographic parity
○ Equal opportunity
○ Equal accuracy
○ Group thresholds

Optimal single threshold for 2 values of sex

Sort by: Count

Feature Value	Count	Model	Threshold	False Positives (%)	False Negatives (%)	Accuracy (%)	F1
Male	339	1	0.4	8.6	8.6	82.9	0.70
		2	0.64	3.8	14.2	82.0	0.62
Female	161	1	0.4	2.5	4.3	93.2	0.67
		2	0.64	0.6	6.2	93.2	0.59

Confusion Matrix

1	Predicted Yes		Predicted No		Total	
Actual Yes	6.8%	(11)	4.3%	(7)	11.2%	(18)
Actual No	2.5%	(4)	86.3%	(139)	88.8%	(143)
Total	9.3%	(15)	90.7%	(146)		

2	Predicted Yes		Predicted No		Total	
Actual Yes	5.0%	(8)	6.2%	(10)	11.2%	(18)
Actual No	0.6%	(1)	88.2%	(142)	88.8%	(143)
Total	5.6%	(9)	94.4%	(152)		

Figure 2.3: WIT's second panel dedicated to (fairness-aware) threshold tuning [314].

By combining individual counterfactual explanations with an interface, ViCE [89] is a tool that seeks to help non-experts better understand the predictions made by ML models. Instead of showing merely what changes would be needed in a reduced set of continuous features to change a single binary prediction of a ML model, ViCE combines this information with some context in a visual way. In other words, the explanations are placed next to the data distribution (for each feature) and there is the possibility to filter this distribution by choosing only positive or negative instances according to the ground truth, for example.

For regression problems, 2019 brought FeatureExplorer [350], a tool developed in a very specific context (a collaboration with three remote sensing experts and plant scientists whose goal was to predict plants' wet biomass using data recorded in hyperspectral imagery). It supports the dynamic evaluation of regression models and importance of feature subsets through the interactive selection of features in high-dimensional feature spaces typical of hyperspectral images. One interesting point is the use of a double visual encoding of the Pearson's coefficient values in the correlation matrix.

2.3.1.3 Model Bias and Fairness Evaluation

Concerned with model bias and fairness, there are also three proposals that are intended to support this specific field: FairSight [3], FairVis [5], and Aequitas [259] (and partially WIT [314]). It is important to emphasize that bias and fairness mainly concern the comparison of the results obtained over different instance subgroups, which have specific characteristics in the features that compose them.

FairSight [3] is a visual interface divided into six components to help in achieving fair decision-making through the ML pipeline of a learning to rank problem (it is not flexible enough for classification). It instantiates a model-agnostic theoretically framework, FairDM, that incorporates different notions of fairness (group and individual) and supports measuring, identifying, and mitigating bias. Each of these parts has associated metrics and operate in different spaces, namely the input, the mapping, and the output spaces. The six components support the following workflow: (1) build a model, (2) overview ranking, (3) inspect global and local fairness, (4) inspect features, and (5) compare different rankings.

For binary classification problems, FairVis [5] presents a different proposal (Figure 2.4). It is tailored for subgroup discovery to audit the fairness of ML models. Also, it lets the user apply domain knowledge to generate and investigate known subgroups, and explore suggested and similar subgroups (via intersectional groups). The feature distribution view gives users an overview of the dataset distribution and allows them to generate groups to visualize in the subgroup overview. Users can then add additional subgroups provided by the suggested and similar subgroup view, and compare and further analyze them in the detailed comparison view. One of the main visualizations is a small multiple with strip plots to compare different performance metrics in different instance subgroups. The ideas in this paper have the potential to be tested at Feedzai, although some of the techniques, such as the clustering-based subgroup generation technique (pre-running k-Means), need to be further measured and validated.

Aequitas [259] is a bias and fairness audit toolkit (there is a Python version of it). Through different bias and fairness metrics, which will depend on the context, it is possible to compare these values for different subgroups and evaluate ML models. Attached to this possibility, there are plotting capabilities that allow the user to check the results.

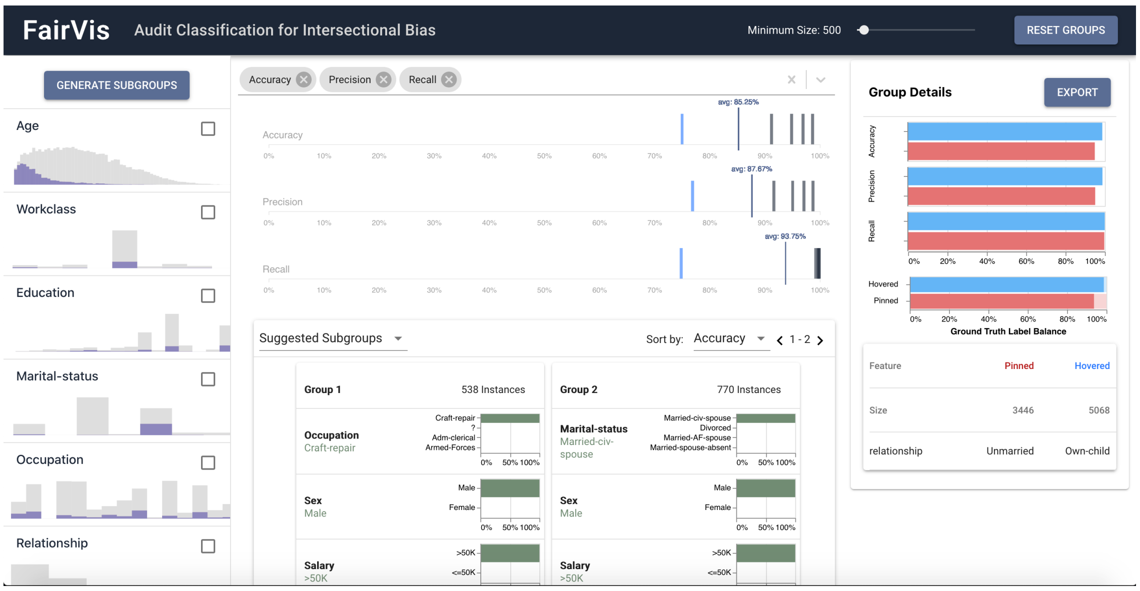


Figure 2.4: FairVis web interface [5].

2.3.1.4 Model-Specific Evaluation

As for papers with model-specific tools, there are three orientations that align with the main algorithms currently used (directly or indirectly) at Feedzai: DT (indirectly), RF, and Gradient Boosting.

For DT models, BaobabView [296] is a visual interface for the manual and algorithmically supported construction and analysis of DT models. A top-down node-link diagram is used for tree visualization. Each node contains information regarding the split predicate, the class distribution, and the feature-class values and distributions for a future split in a streamgraph format. There is a focused layout for tree and data analysis where the emphasis is placed on links. Nodes are not shown; only predicates (some are difficult to read), and links are drawn as continuous streams (the idea of adapting the interface according to the main objective is interesting). In addition, there are other mechanisms that make it possible to ascertain the confusion matrix and see the different features ordered by an impurity measure (a measure of the homogeneity of the classes at the node level [183]), for example. Although DT models are not used directly at Feedzai, it is also worth mentioning the encodings given to the links and their drawing as Bézier curves.

iForest, as the name implies, aims at interpreting RF models and their predictions [352]. The interface is divided into (1) a data overview section displaying an overview of how Random Forests classify data, (2) a feature view depicting the relationships between features and predictions, and (3) a decision path view revealing the underlying working mechanisms by enabling users to audit and compare different decision paths. The most exciting part is the decision path view, given the way it organizes how to analyze the different paths of the different trees. Basically, it contains a list in which all decision paths of a prediction (an instance, in other words) are represented on a row. These rows have three components: (1) the decision path projection that provides an overview of all decision paths in a single prediction based on their similarities (using t-distributed stochastic neighbor embedding (t-SNE)), (2) the feature summary that summarizes the critical feature ranges of decision paths, and (3) the decision path flow that provides the detailed information of decision paths layer by layer (each column represents a layer). If a layer contains leaf nodes, a pie chart is appended in the corresponding column, where the red sector represents negative labels and the blue sector represents positive labels (this pie chart only compares a given category within the total of a single one, so its usage can be considered effective [48]). Curves connect features from different layers. The curve width encodes the number of decision paths that have the corresponding feature pair in adjacent layers. This interface has a long learning curve, and the use of PDP technique has the problems presented in [148, 192].

Considering gradient boosting models, namely LightGBM [132] and XGBoost [49], BOOSTVis [172] is a visual interface (Figure 2.5) that helps users analyze and diagnose the training process. It combines a temporal (or sequential) confusion matrix (to show the evolution of model performance at the class-level), a t-SNE projection (to show relationships between instances and outliers), a classifier view (to provide an overview of all the decision trees and highlight the selected one), and a table displaying the feature distributions on the selected subsets of instances. When the user selects a specific predicted class, the temporal prediction scores of its instances is represented by a line chart. Also, this paper presents a set of algorithms that feed the proposed visualizations:

- A recursive time-series segmentation algorithm is used to present only the k segments in the temporal confusion matrix that minimize the intra-segment variances. It is a kind of undersampling of the most interesting points.
- A constrained variation of the tree edit distance is used to estimate the distances between the tree structures.
- k-Medoids is used to cluster the trees according to the output of the previous algorithm.
- A tree cut algorithm is used to highlight the layers of interest when the user is looking to the cluster glyphs (simplified representations), and gray out the others.
- k-Means is used to group similar lines (and with the same actual class) on the line chart of the performance of a selected instance subgroup.

However, learning how to use this tool is not as simple as the authors argue. It would be interesting to have a tour function that explains the interface step by step. Also, BOOSTVis is computationally expensive (a lot of different algorithms to build the visualizations). Ignoring the fact that this tool is optimized for MCC problems, the temporal confusion matrix, the t-SNE projection, and the tree clusters are interesting ideas, but they have to be adapted and tested at Feedzai. Something that strengthens this perspective

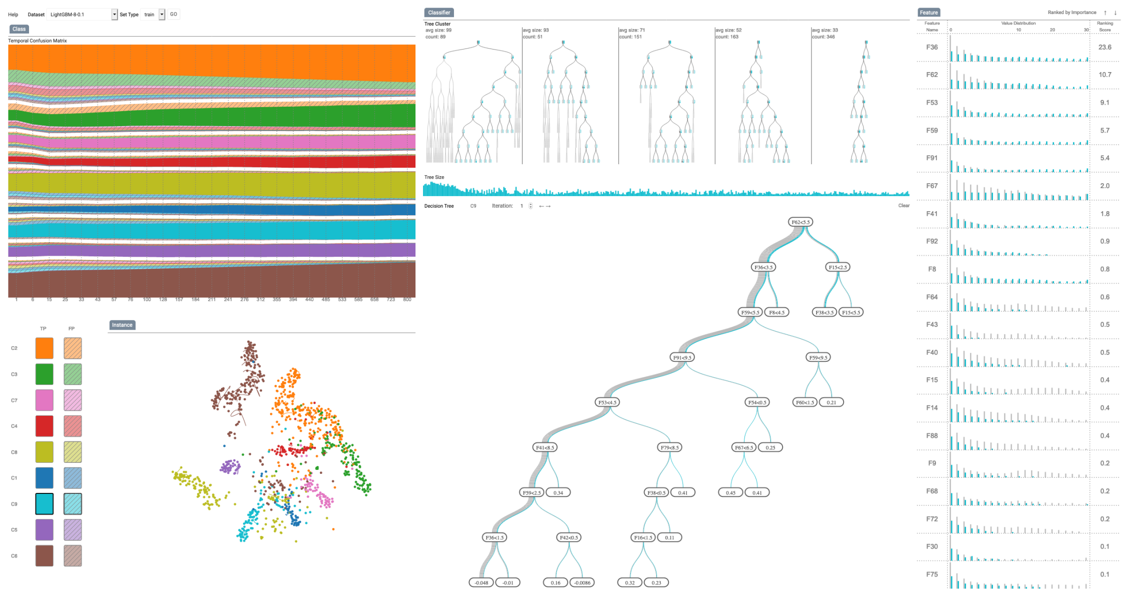


Figure 2.5: BOOSTVis web interface [172].

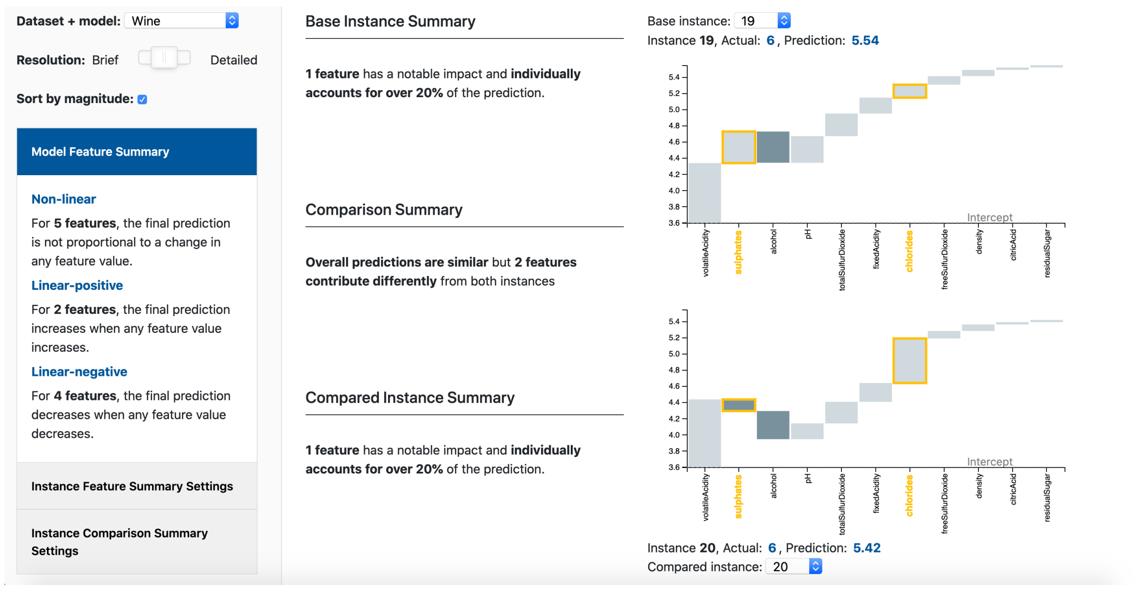
is the feedback collected during a presentation of this paper in an internal initiative at Feedzai where the utility of the temporal confusion matrix and access to t-SNE (although for the original data, since this paper uses an adapted version where the input is the prediction score vectors and where it is not clear whether the use of Euclidean distance is the best option) was highlighted.

For Generalized Additive Models (GAMs) (a generalization, as the name implies, of linear models), TeleGam [112] offers an interface (Figure 2.6) in which the main aspect that stands out is the combination of interactive charts with textual descriptions. Through descriptions (or explanations) generated from threshold-based heuristics, line charts and waterfall charts, which aim to cover features and specific instances, TeleGam seeks to reconcile and link these two elements in a complementary way, helping to clarify the (global and local) behavior of a given GAM. In addition, this interface also includes a three-level slider (*Resolution*, in the upper left corner in Figure 2.6) that enables the adjustment of the level of detail of all explanations, something that can contribute to a greater range of users and use cases.

2.3.1.5 Model Building and Evaluation

There are other papers, focused on other parallel issues, which also deserve to be highlighted briefly.

In order to iteratively build ensemble models and linear models from a local perspective and not just from a global perspective, EnsembleMatrix [289] and LoVis [351] present specific interfaces for this, respectively, where the Model Evaluation component is also present. EnsembleMatrix is composed of interactive confusion matrices to help users understand the relative performance of various classification models and manually



Dataset + model: Wine

Resolution: Brief / Detailed

Sort by magnitude: ✓

Model Feature Summary

Non-linear

For **5 features**, the final prediction is not proportional to a change in any feature value.

Linear-positive

For **2 features**, the final prediction increases when any feature value increases.

Linear-negative

For **4 features**, the final prediction decreases when any feature value decreases.

Instance Feature Summary Settings

Instance Comparison Summary Settings

Base Instance Summary

1 feature has a notable impact and **individually accounts for over 20%** of the prediction.

Comparison Summary

Overall predictions are similar but **2 features contribute differently** from both instances

Compared Instance Summary

1 feature has a notable impact and **individually accounts for over 20%** of the prediction.

Base instance: 19

Instance **19**, Actual: **6**, Prediction: **5.54**

Instance **20**, Actual: **6**, Prediction: **5.42**

Compared instance: 20

Figure 2.6: The TeleGam web interface [112] with an example of the interactive link between the explanatory text and the respective bars in the waterfall charts (for two selected instances).

create model combinations (a.k.a. ensemble models). One interesting point is the linear combination widget that allows the user to combine models in a simple way, to quickly test some hypotheses, for example, with some sliders to adjust in more detail. The user can scrub inside this polygon to specify classifier weights (there is a classifier at each vertex). To make the appropriate mapping of position to weight, the polygon was parameterized using Wachspress coordinates. LoVis is a very cryptic paper that presents a tool to partition instances and analyze which models or features produce better predictions for each local segment [349].

2.3.1.6 Theoretical Frameworks

From a more theoretical perspective, there are two papers that present frameworks more linked to the interpretability of models. Explanatory Debugging [152] is, in a few words, a two-way exchange of explanations between a user and a ML system. In terms of explainability (accurately explain the learning system's reasons for each prediction to the user), its principles are: (1) soundness, (2) completeness, and (3) don't overwhelm. In terms of correctability (allow users to explain corrections back to the learning system), the principles are: (1) be actionable, (2) be reversible, (3) always honor user feedback, and (4) incremental changes matter. EluciDebug is an instantiation of these principles for the classification of emails in different folders (classes). One interesting point is the consideration of principles at the interface level, something to keep in mind for the assembling of a Python package. On the other hand, explAIner [280] is a TensorBoard [1] plugin that tries to instantiate a modular conceptual framework designed to understand ML models, diagnose model limitations using different XAI methods, and refine and optimize the models, complemented by eight global monitoring and steering mechanisms. However, the solution is oriented towards DL models, and explAIner has only a small portion of the theoretically proposed things implemented, which leads to the lack of a more concrete way, for example, to have a data shift scoring mechanism or provenance tracking (a kind of data lineage). An alternative to shareable provenance tracking, in this case for Jupyter Notebook, is the extension called ProvBook [260]. Simultaneously, the ReproduceMeGit tool [261] can be used to visualize different metrics about the reproducibility of Jupyter notebooks housed in GitHub repositories (it shows, for example, several pie charts with the exceptions raised during notebook execution or the number of different or similar results).

2.3.1.7 Model Evaluation for Data Exploration

From a perspective more inclined towards data exploration than model debugging [215], Prospect proposes the use and analysis of various models for debugging purposes (to link classification results back to noteworthy instances) in a tool that trains a collection of models, based on different configurations, automatically, aggregates results from those models, and provides interactive visualizations to help users understand and debug data. The authors hypothesize that multiple models can marginalize the bias of individual models and act as a lens for scrutinizing some of the key properties of the original data. One interesting point is the *incorrectness versus label entropy* plot. This plot is, basically, a scatterplot where the X-axis is the percentage of configurations that misclassify an instance (incorrectness) and the Y-axis is the entropy of the distribution of labels predicted by each configuration for an instance (label entropy). Also, it has three different regions: (1) the canonical region consists of instances that most configurations classify correctly, (2) the unsure region consists of instances for which different configurations generate widely varying predicted labels, and (3) the confused region consists of instances with high incorrectness and low entropy (it is interesting for detecting label noise).

2.3.1.8 Python, R, and JVM packages

In terms of packages, in a more practical and hands-on perspective, there are some references to point out. First, the ceterisParibus package [33] offers a set of charts that iterate and instantiate the idea of PDP, and allows to agnostically check and compare classification (or regression) model responses around a single instance in the feature space. Yellowbrick [30] is a ML visualization Python package that provides a set of visualizations for feature and target analysis, classification, regression, and clustering models, and text analysis. Its OOTB integration with the scikit-learn application programming interface (API) [44] enables it to be used in scenarios where the most common Python stack for ML is available. One of the plots that stands out the most for its usefulness in evaluating different classification thresholds, especially in a context where this threshold needs to be carefully adjusted (or just to have a general overview), is the discrimination threshold plot for binary classification (Figure 2.7). Similarly, EvalML [9], an Automated Machine Learning (AutoML) Python package, also contains a version of this chart (called "Binary Objective Score vs. Threshold"), as well as other (common) classification-oriented, Plotly-based (an interactive visualization package) charts under its Model Understanding API, such as the ROC curve chart, the PR curve chart, the confusion matrix, the PDP, and ranking bar charts for FI (including for the model-agnostic Permutation FI algorithm). scikit-learn [217] also has some common plotting capabilities to help with Model Evaluation. There are plotting functions for the confusion matrix, PR curve, ROC curve, and PDP.

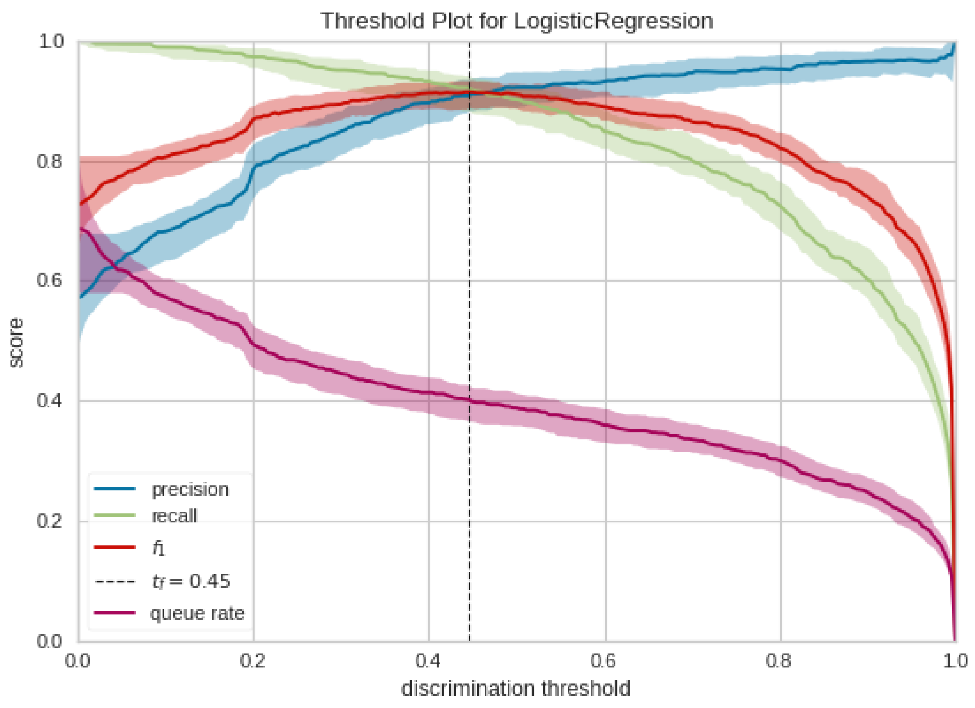


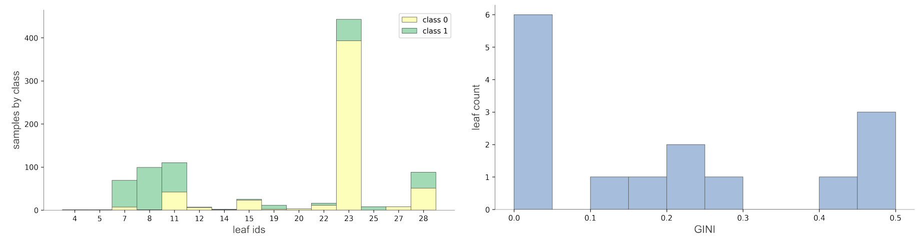
Figure 2.7: Discrimination threshold plot [30]. This plot places different performance metrics (Precision, Recall, F_1 Score, and Queue Rate, for example) against different classification thresholds in order to help adjust the threshold to, consequently, tune a certain performance metric.

Specifically designed for PySpark (in order to provide a user experience similar to pandas and scikit-learn), HandySpark [88] is a Python package that, in terms of Model Evaluation for binary classifiers, offers the possibility to plot Matplotlib-based ROC and PR curves, as well as check the confusion categories and performance metrics for various classification thresholds.

For ML and DV, with a proposal that aims to reach several target groups (in terms of coding skills), Orange is a component-based Python package and visual programming tool (also tailored for interactive DV) conceived in the 90's and still in active development [65]. In a simplified way, Orange is a kind of scikit-learn outside the NumPy-based scientific Python ecosystem. In addition to a significant number of algorithms and data processing capabilities, Orange has a set of charts (for projections as well), both for data analysis and for Model Evaluation. Within the Model Evaluation charts, in addition to the typical ROC curve chart and confusion matrix, Orange also provides access to the cumulative lift/gains curve chart [56] (a TPR versus percentile chart pre-ordered by prediction score), as well as the probability calibration/reliability chart. The last chart mentioned is a chart of the fraction of positive instances against the average of the predicted probability-based scores for a given number of bins (in order to verify whether a classifier can be interpreted as a probabilistic one). This type of curve-based charts (and more) is also available in Scikit-plot [203], a ML visualization Python package that can be easily integrated with scikit-learn-like output (however, its development has been stalled since August 2018). This package contains two extra curve-based charts worth mentioning:

- Lift curve chart: This chart is a variation of the cumulative lift chart in which the values of the Y-axis are divided by the respective percentile value (in this case, the baseline is a horizontal line at value 1).
- Kolmogorov-Smirnov (KS) statistic chart: This chart is also similar to the cumulative lift chart, but in this case, it focuses on the separation between the positive and negative classes (so, it contains two curves). Thus, the percentages of (true) positives and (true) negatives below a specific threshold (a cut-off point that reflects a certain percentage of the instances that make up the dataset [123]) are plotted on the Y-axis, while the threshold itself is plotted on the X-axis. There is also a version of this chart in which the classification threshold is plotted on the X-axis and the lines encode the TPR and FPR values [136].

In the panorama of model-specific packages, namely for classification or regression DT models, dtreeviz [214] appears with a wide range of visualizations, both for Model Evaluation and for learning purposes, which allow not only to have an overview of the trees (via tree diagrams), but also the prediction paths for an instance, FI values, various leaf-based distributions, and feature-target relationships. These feature-target relationships, in the case of classification problems, are framed in charts that result from the junction of scatterplots for a pair of features and a heatmap in the background for the decision surface, with the color of the points encoding the true classes (the tree diagrams themselves can also be used for this purpose, as they can be customized to include class-based stacked histograms, for example). There is also a univariate jittered version in which the heatmap is replaced by a kind of stacked bar intercepted by vertical lines that simulate the decision surface. For this work, the charts for leaf-based distributions are implicitly interesting, like those in Figure 2.8, as they allow to inspect the structure of the model and the molding of the instances to this same structure (a kind of model-driven data analysis). So, they can serve as inspiration for charts that allow exploring the volume-like structure of RF models (they are composed of multiple trees), through leaf/terminal node depth histograms or node depth histograms considering a certain feature, helping not only to understand the behavior of these models (interpretability) but also to obtain some insights that can help in choosing hyperparameters. Another relevant model-specific chart is the (feature) split value (count) histogram (an example can be seen in Figure 2.9) available in LightGBM's Python API [132] (in addition to the FI value ranking chart that will be adapted in the package developed for this work).



(a) Stacked bar chart for the distribution of the instances (and the respective classes) in the various leaf/terminal nodes of a DT model. In this example, leaf node 23 contains the vast majority of instances and is dominated by class 0.

(b) Histogram with the number of leaf nodes for each bin of Gini impurity values (the value 0 implies that all values are of the same class, while the value 0.5 implies that there is a 50% probability of incorrectly classifying an instance) for a DT model. In this example, most leaf nodes have a low Gini impurity value but there are some with high values (looking at the number of instances of each leaf is something interesting to complement this chart).

Figure 2.8: Two examples of charts generated with the dtreeviz [214] package using a baseline DT model [217] (with a maximum depth of 4) and the Titanic dataset [127]. The leaf node identifier increases from left to right.

In the R and Bioinformatics ecosystems, it is possible to find a set of packages that iterate over the basic ROC curve plot in order to provide an empowered suite for ROC curve analysis:

- cvAUC [162] provides a confidence interval estimation method for cross-validated AUROC values.
- PRROC [92] allows to estimate the ROC curve and AUROC for soft-labeled data.
- plotROC [257] presents a very easy way to compare several ROC curves on the same plot and to add labels.
- pROC [250] includes functions for computing confidence intervals, statistical tests for comparing total or partial Area Under the Curve (pAUC), and methods for smoothing ROC curves.

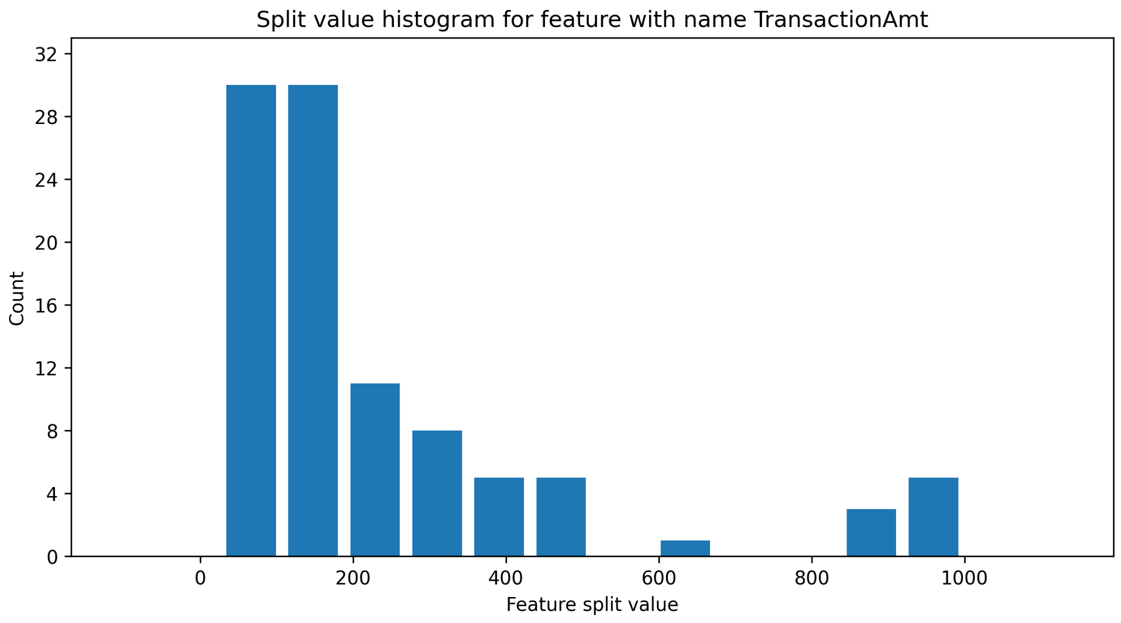


Figure 2.9: An example of a split value histogram obtained from the LightGBM model described in Appendix K. The X-axis corresponds to the binned values (automatically calculated) of the transaction amount (continuous) feature (*TransactionAmt*) for a part of the IEEE-CIS Fraud Detection Kaggle competition dataset [119]. The Y-axis encodes the number of times that values for each of the bins were used as the feature split value in the different trees of the model (LightGBM is a Gradient Boosting Decision Tree (GBDT) algorithm).

- ROCit [135], by default, plot the location of the Youden index [130] (if Recall and TNR are equally important or desirable, this value will indicate the optimal classification threshold [282]) to the plot and allows to estimate the ROC curve empirically, parametrically, or nonparametrically.

For a broader curve-based analysis (to tackle, in particular, imbalanced datasets), which includes PR curves in addition to ROC curves, there is Precrec [258]. This R package provides mechanisms for computing and visualizing partial and full curves, with or without confidence (interval) bands (via vertical averaging [75], that is, mean TPR with fixed FPR, for multiple *test* sets obtained, for example, through a resampling method such as *k*-fold CV), as well as for calculating a set of performance metrics, including the full and/or partial AUC. As an interesting detail in the header of the generated charts, Precrec adds the number of positive and negative instances. In addition, this package is used by mlr3viz [159] (for binary classification), one of the mlr3 ecosystem packages for ML.

Finally, in terms of Java/JVM-based packages/tools (for ML and DL), there are some options to highlight, although the visualization component for Model Evaluation is not constant for all of them [134]. Eclipse DeepLearning4J [72], in addition to performance metrics and probability calibration capabilities, has a UI to monitor the training of DL models with different neural network (NN)-oriented (and time series-like) charts and have an overview of the final results (more about DL in the next section). Smile [167], on the other hand, in addition to the evaluation metrics (and mechanisms to validate and iterate models, such as hyperparameter tuning and CV), has a DV API (actually, it has an imperative API and a declarative one wrapping Vega-Lite [264], similar to Altair [297]). The rest of the verified packages/tools, namely Spark's MLlib [183], Tribuo (it also has a built-in provenance tracking system, as well as a LIME implementation for local explanations of instances) [211], Apache Ignite [18], Deep Java Library (DJL) [10] and Dagli [170], support multiple performance metrics and other types of metrics (they are metric-centric packages in terms of Model Evaluation, basically).

2.3.1.9 Deep Learning

As expected from a very active field today, there are a considerable number of proposals for DL models. Although not the type of models in focus here, they present a very current and future perspective on, mainly, the importance to understand the underlying behavior of ML/DL models. In 2018, a survey [111] was published covering 38 different papers and analyzing them based on the *Five W's and How* technique [334]. Although analyzing DL models *from the inside* is not of interest for this work, there are some papers that present visualizations whose main idea can be adapted:

- ActiVis [129] is a visual assessment tool for deep neural network models from Facebook. One of the panels, the instance selection one, is an interesting way of providing an overview of instances with their prediction results and highlight meaningful instances to further explore (in a faceted waffle chart).
- Seq2Seq-Vis [284] is a visual debugging tool for sequence-to-sequence models. The idea of having a kind of neighborhood view, in order to allow the user to look at model decisions in the context of finding similar instances, has some intrinsic potential.
- TensorWatch [272], from Microsoft, is a visual debugging tool that can be used in notebooks. Its real-time visual logging philosophy is an interesting window for the idea of providing information during model training and not just after it finishes.

In addition, Ludwig [190, 191], a TensorFlow-based Python package for training and testing DL models using a no-code command-line interface (CLI) (supported by dataset and YAML configuration files) or a low-code programmatic API (also from Uber), provides a visualization suite (Visualization API) for Model Evaluation (particularly for binary and MCC problems) — most of its charts are also used or can be adapted for ML models (the learning curve chart is, however, tailored to DL as it allows the verification of one or more metrics over the course of the training epochs). Within the available charts, it is possible to highlight the following:

- The characteristic ROC curve chart and (heatmap-based) confusion matrix (paired with a bar chart, particularly useful for MCC, where classes are ranked by entropy).
- Several variations of bar charts to compare various performance metrics (particularly useful to compare two or more models) considering the whole dataset or just a part of it.
- The probability calibration chart (for binary classification and MCC). The main chart is complemented by a (count) histogram (similar to that of scikit-learn [217]) that aims to approximate the distributions of the predicted scores/probabilities for each model or by a bar chart with the Brier Score for each model. The Brier Score is a binary/multi-class performance metric and, in general terms, it consists of the quadratic difference between the predicted scores and the value 0 or 1 (indicator variable) according to the true class of each instance [34]. Formally, the Binary Brier Score is given by the following formula: $\frac{1}{n}\sum_{i=1}^{n}(I_i - p_i)^2$ (I_i is 1 if the instance i belongs to the positive class and it is 0 otherwise) [158].
- A dual-axis line chart that puts the frequency of each class and the respective F_1 Score values in perspective (the idea of reporting the values of a performance metric by class, together with the number of associated instances is interesting to look more critically at the results obtained).
- Different versions of a chart, with more or less dimensions, under the name *confidence thresholding* that aim, in general, to compare Accuracy and data coverage for various classification thresholds (with a threshold step equal to 0.05 [190]). Different data coverage percentages are obtained based on the highest class-oriented probability/score for each instance (or based on the score of the positive class in a binary classification problem) compared to different classification thresholds, that is, the percentages are obtained by dividing the number of instances with a probability greater (or equal) than a certain classification threshold by the total number of instances (the Accuracy values are also calculated based on this filtered dataset) [190].
- Two threshold-dependent charts to compare the predictions of two or more classifiers: (1) a nested (two-level) donut chart (Figure 2.10) for two models whose segments are colored according to the correctness of the predictions (in terms of output labels) of both models (with the different possible combinations); (2) a probability/ratio-based radar chart, designed for MCC problems, to show the distribution of predictions, in terms of output labels, for two or more models, including the class-based "ground truth" distribution.
- The performance metric by classification threshold chart for one or more models, with each performance metric plotted on a separate line chart (similar to Yellowbrick's discrimination threshold chart [30] for different metrics related to a model).

Two other interesting aspects of Ludwig are the idea of a CLI argument or function, corresponding to a visualization, not necessarily meaning the production of a chart, but

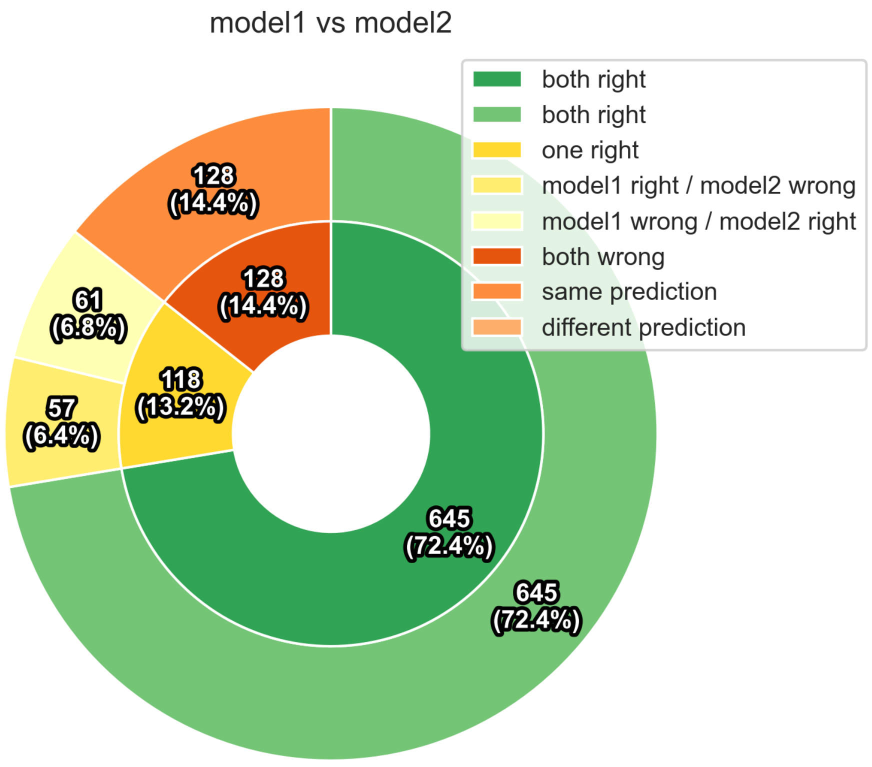


Figure 2.10: A nested (two-level) donut chart [190, 191] to compare the correctness of the predictions of two (binary) classification models, namely the segmentation and the number of labels correctly and incorrectly estimated considering the instances of the Titanic's training dataset [127]. This type of graph can be problematic since it implies the comparison of circles/rings with different radii and areas [275], something mitigated by the low number of segments (identified by the legend and the color), by the presence of absolute and relative numbers, and by the fact that there are two rings with distinct well-identified granularities, which invite a two-stage reading (however, it is important to clarify how this graph works with users). This chart is most useful for MCC contexts due to the extra (light) red zone segmentation possible since there are more than two labels. The function to create and save this image was monkey patched to ensure better image quality/resolution (300 DPI).

rather a small set of charts complementary to each other, and the organization of dependencies in various groups. with different purposes, facilitating the installation of this package with the functionalities only relevant to the user (through the *extras_require* keyword argument from the *setup()* function used with the standard Setuptools build system [233]).

2.3.1.10 Vendors of Data Science and Machine Learning (DSML) platforms

While vendors of DSML (closed-source) platforms [150] are not an option due to the associated costs and because they entail the introduction of a completely new tool in the workflow of Feedzai's data scientists, they can serve as inspiration through community/open-source editions and information available online for their built-in (industrial-strength) Model Evaluation features distinguishable from those discussed so far. So, these different features, as far as possible, will be the only ones covered in this section.

In KNIME Analytics Platform [31], Cohen's kappa is calculated by default with the confusion matrix [143, 319] (computed through a node available in one of the basic extensions). Cohen's kappa is a commonly used metric for inter/intra-rater reliability, and here it is used as an alternative to Accuracy (also considering the predicted and actual classes), especially for imbalanced datasets [319]. In terms of charts for Model Evaluation, the linked charts (these charts are common for binary classification, but when they work together, in a kind of small dashboard, the user experience is enriched) shown in Figure 2.11 stand out.

Regarding RapidMiner Studio [186], there are three aspects to highlight: (1) *Visualize Model by Self-Organizing Map (SOM)*, that is, it is possible to generate a two-dimensional SOM from the input space colored by the predictions of a given model [242]; (2) Model Comparison via statistical significance tests (ANOVA and T-Test) [239, 241]; (3) in addition to the performance metrics, it is also possible to see the models' runtimes [240].

For DataRobot, there is a set of interesting charts for Model Evaluation and Comparison. The Accuracy Over Time chart [228] is a line chart to check the stability and performance of a model over time (X-axis). It can be useful for this project since Feedzai's datasets also contain a time component. In addition, it is also possible to compare the predicted line with the line composed of current (aggregated) values, and the existence of peaks in only one of the lines may be an indicator of something that deserves to be investigated (in the shared example, the Y-axis corresponds to the field target of a regression problem, but this idea can be adapted to different performance metrics [228]). For multiple models, the Speed vs Accuracy chart [100] is a scatterplot to analyze the trade-off between prediction runtime/speed (the time to make 1000 predictions in milliseconds, for example) and a specific performance metric. The Learning Curves chart [100], another line chart whose Y-axis corresponds to a performance metric and the X-axis to a percentage of the dataset size, has two areas highlighted in the chart itself for the validation set and the holdout set (that is, for the percentages that add up to the training data), as these

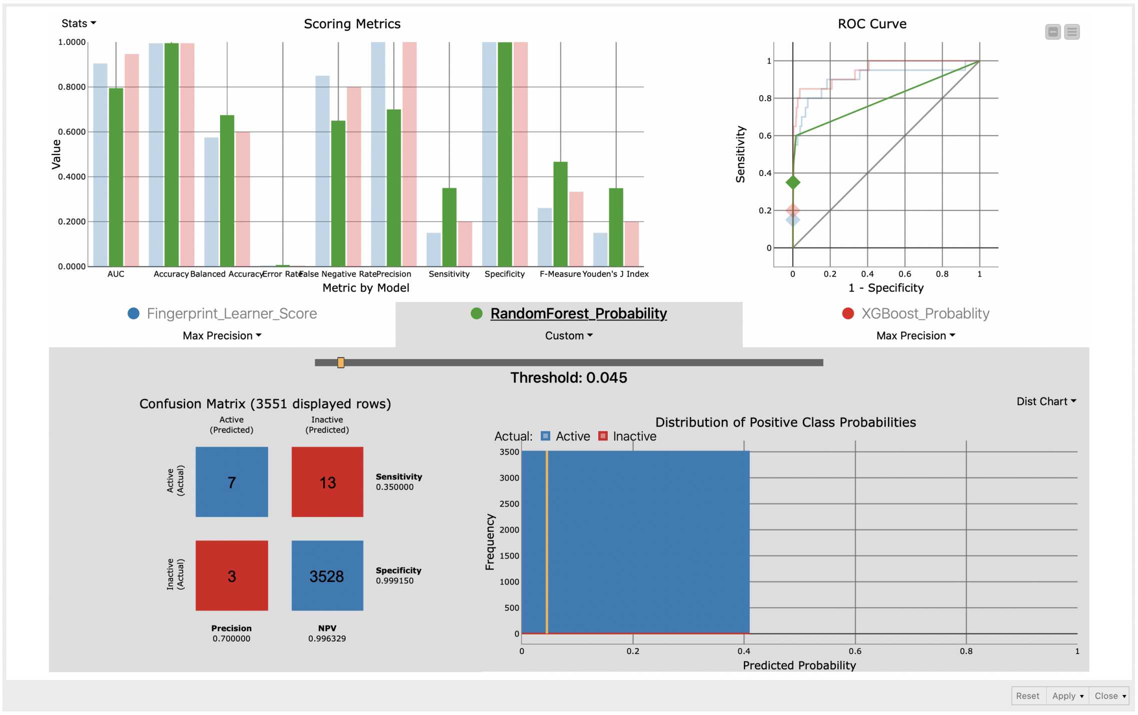


Figure 2.11: An example of the four linked charts produced by the Binary Classification Inspector node (an element of a pipeline) available on the KNIME Machine Learning Interpretability Extension [142] for KNIME Analytics Platform [31]. This widget can be used for Model Comparison, Threshold Tuning, and Visual Performance Analysis (more information about these terms in the Model Evaluation Topology section). In the ROC curve chart, the highlighted points (with a diamond-like mark) correspond to the classification thresholds that maximize a given performance metric (Precision in this case) or a user-defined threshold through the slider for the RF model.

separate sets, together with the training data, make up the total size (100%) of the dataset. The Lift chart [100, 101, 228], in DataRobot, can be leveraged for one (containing two lines, one of which is for the "ground truth") or more models (this Lift chart is different from the one described in the Python, R, and JVM packages section). In this way, the Y-axis corresponds to the average target (prediction score) value, while the X-axis corresponds to the score distribution ranked and binned from lowest to highest (the number of bins can be changed in a dropdown menu). The "ground truth" line corresponds to the average proportion of positive instances, assuming a binary classification problem, for each specified bin [228]. Moreover, the Dual Lift chart [100] is similar to the Lift chart, but specific for comparing two *competing* models (it can be useful for ensemble modeling). For this chart, the predictions are sorted by the magnitude of the difference between the scores of each model and then binned. Finally, the Profit Curve chart [101] is a line chart of the profit against the classification threshold. Profit is calculated based on the confusion matrix and, by default, all outputs are worth the same amount of money (the true confusion categories have a payoff value of 1 and the negative ones a payoff value of -1). These values can be customized through the Payoff Matrix, a user-defined, cost-based confusion matrix for simulation purposes.

As for Dataiku Data Science Studio (DSS) [61], among the various general-purpose charts and those for Model Evaluation, the *Subpopulation analysis* table (Figure 2.12) is one of the highlights (this kind of analysis is important at Feedzai). In the *Confusion matrix* tab, there is a slider for the classification threshold that controls the confusion categories and the performance metrics displayed (the interaction between the slider and the horizontal bar chart for performance metrics causes an effect that resembles a bar chart race), as well as a cost matrix to define different weights for each confusion category. In terms of performance metrics, this platform also includes the Hamming loss (the fraction of labels incorrectly predicted) and calibration loss (the average distance between the calibration curve and the diagonal line) metrics.

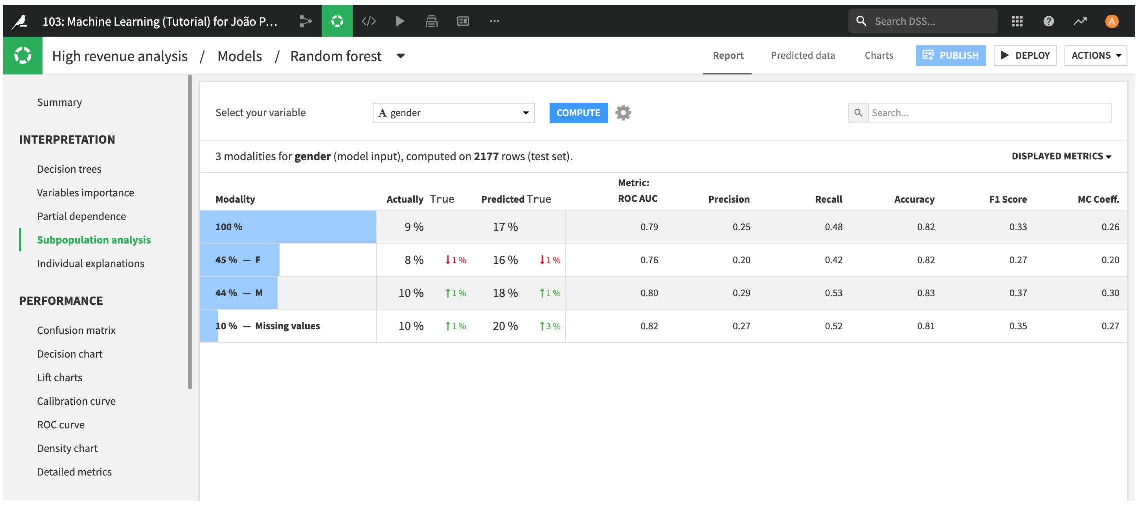
Considering H2O (H2O-3) and H2O Flow (H2O's web UI) [94], the first feature that stands out is the combination of interactive charts and tables by design (the context button in Highcharts [110], a button with several options in the upper right corner of each chart, if specified, allows the addition of a simple table with the data next to the chart as well). In the case of the ROC Curve chart, in addition to two dropdown menus to select a classification threshold to be highlighted manually or based on a criterion, a table is also shown with a series of metrics that complement the threshold-independent and general perspective offered by the curve (an example can be seen in Figure 2.13). The Variable (or Feature) Importance (ranking) chart is another example, where the table shows, for a specific feature, the original value and the value scaled to 1 (or 100%), for example. The Standardized Coefficient Magnitudes chart, for generalized linear model, for example, is a horizontal bar chart (not a diverging bar chart) where the color of each bar encodes the sign of the respective coefficient. However, when using the Python API [93] (in Jupyter Notebook, for example), these charts are redesigned and adapted for Matplotlib. Last but not least, in particular for comparing a considerable number of models, there are two relevant heatmap-based charts: (1) the Variable Importance Heatmap, with rows for features and columns for models; (2) the Model Correlation Heatmap, where the correlation, for classification, is computed based on the frequency of identical predictions.

As a side note, OutSystems' (recent) ML Builder [73] presents, in addition to a donut chart for a performance metric (Accuracy), a short message with the indication of how much better a model is compared to the baseline.

2.3.2 Model Evaluation

Different statistical hypothesis testing approaches.

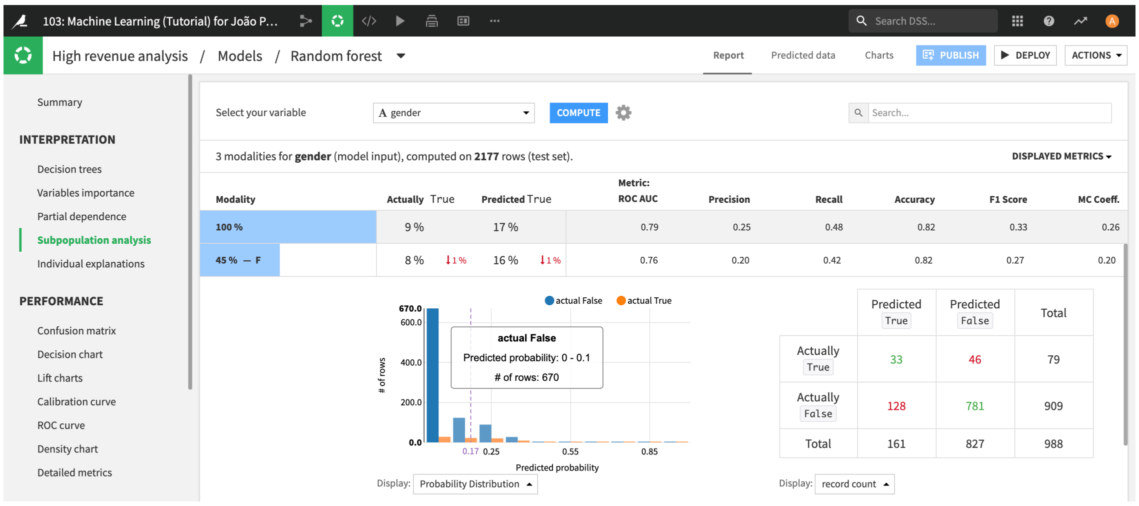
With regard to papers that address Model Evaluation from a purely quantitative/statistical perspective, the first paper to highlight is a kind of survey of *classic* methods [245]. This paper covers bootstrapping and uncertainty, CV and hyperparameter tuning, and



3 modalities for **gender** (model input), computed on **2177** rows (test set).

Modality	Actually True		Predicted True		Metric: ROC AUC	Precision	Recall	Accuracy	F1 Score	MC Coeff.
100 %	9 %		17 %		0.79	0.25	0.48	0.82	0.33	0.26
45 % — F	8 %	↓1 %	16 %	↓1 %	0.76	0.20	0.42	0.82	0.27	0.20
44 % — M	10 %	↑1 %	18 %	↑1 %	0.80	0.29	0.53	0.83	0.37	0.30
10 % — Missing values	10 %	↑1 %	20 %	↑3 %	0.82	0.27	0.52	0.81	0.35	0.27

(a) In addition to the overall performance, this table also shows the performance (and other metrics) for a given breakdown field (*gender*), that is, for each of the values that divide the dataset into different subpopulations.



3 modalities for **gender** (model input), computed on **2177** rows (test set).

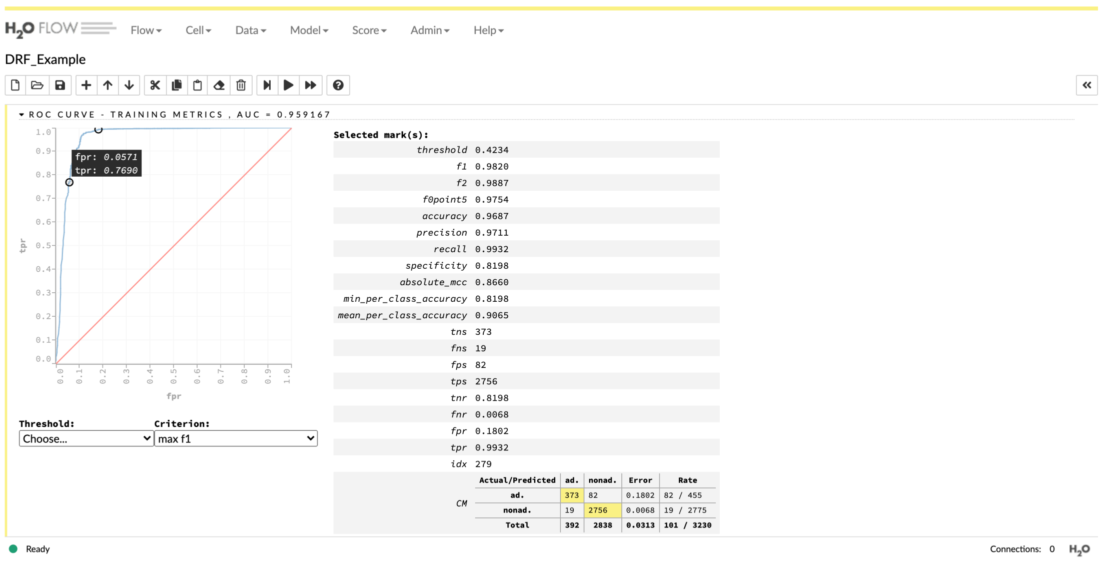
Modality	Actually True		Predicted True		Metric: ROC AUC	Precision	Recall	Accuracy	F1 Score	MC Coeff.
100 %	9 %		17 %		0.79	0.25	0.48	0.82	0.33	0.26
45 % — F	8 %	↓1 %	16 %	↓1 %	0.76	0.20	0.42	0.82	0.27	0.20

actual False
Predicted probability: 0 - 0.1
# of rows: 670

	Predicted True	Predicted False	Total
Actually True	33	46	79
Actually False	128	781	909
Total	161	827	988

(b) By clicking on one of the rows (which act as an accordion), it is possible to see complementary interactive charts, such as a score distribution chart and a confusion matrix.

Figure 2.12: An example of the *Subpopulation analysis* interactive table available on the Dataiku DSS platform [61].



Selected mark(s):

threshold	0.4234
f1	0.9820
f2	0.9887
f0point5	0.9754
accuracy	0.9687
precision	0.9711
recall	0.9932
specificity	0.8198
absolute_mcc	0.8660
min_per_class_accuracy	0.8198
mean_per_class_accuracy	0.9065
tns	373
fns	19
fps	82
tps	2756
tnr	0.8198
fnr	0.0068
fpr	0.1802
tpr	0.9932
idx	279

CM

Actual/Predicted	ad.	nonad.	Error	Rate
ad.	373	82	0.1802	82 / 455
nonad.	19	2756	0.0068	19 / 2775
Total	392	2838	0.0313	101 / 3230

Figure 2.13: An example of the ROC Curve chart with the respective table referring to the classification threshold that maximizes the F_1 Score available on the H2O Flow platform.

algorithm/model comparison using statistical tests and nested CV. The author makes a distinction between algorithm comparison and model comparison. Algorithm comparison occurs when comparing between different sets of models where each set was fitted to different training sets. Looking at the author's suggestions for classification models and large datasets, it is worth noting the computation of confidence intervals to estimate performance, and the use of McNemar's (a nonparametric test that uses the misclassified part of the confusion matrix to compare two classification models [64]) and Cochran's Q tests (a generalized version of McNemar's test that can be used to compare three or more classification models) to compare models (considering the same test set). The use of this type of tests can be an asset to allow the comparison of different models, especially when there is a set whose performance is similar (both are fast to compute, for example). On the other hand, these methods and techniques are implemented in a ready-to-use Python package called mlxtend [244].

2.3.3 Data Visualization

From how to present ML-based information to time-based charts and fraud detection tools.

Time-based charts are a type of chart that can be leveraged given the sequential nature of Feedzai's problems. Periphery plots [201] (Figure 2.14) can be seen as a technique for augmenting time-aligned temporal charts with a set of horizontally aligned peripheral views that provide multi-resolution contexts before and after a focal temporal period. This idea has the potential to be tested at Feedzai, notably as a way to visualize model performance across various metrics over time (over the period of the test set, for example). It may be an option to evaluate the offline degradation/stability of different models.

Although a model may, in the period closest to the one that was trained, have a better performance than the other models, it can get worse more quickly than other more stable models.

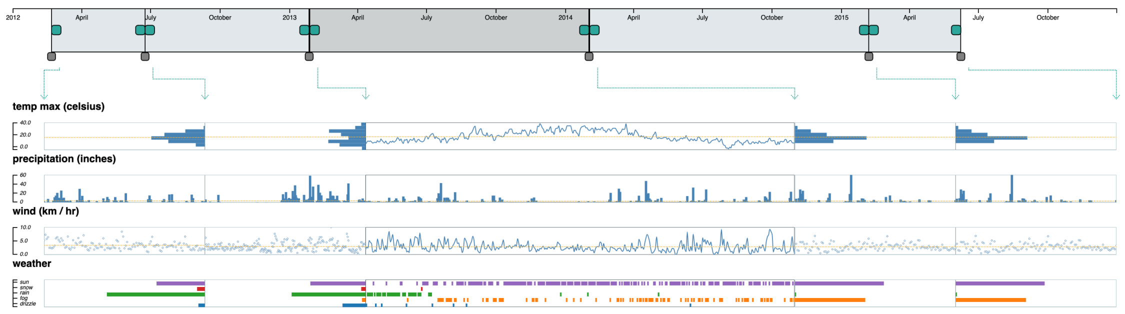


Figure 2.14: Periphery plots [201]. This paper basically presents a small multiple (each plot is called a track) with a control timeline for defining the context zones (periphery plots) and focus zone (focus plot), thus allowing to compare the distributions of various features.

Considering each model, its hyperparameters and the results in terms of performance metrics as a tabular dataset, parallel coordinates [145] are a convenient way to visualize and discover possible patterns from high-dimensional data using the original data. Although the focus is on DL models and experiments full of hyperparameters and different models, HiPlot [103], created by Facebook, offers a visualization adapted for model comparison which may also be useful for comparing ML models as well (HiPlot is ready to create the chart from a simple array of dictionaries). In addition, model filtering and axis manipulation options offer greater flexibility than traditional static plots. However, the value of this visualization for Feedzai is unclear given that Random Forest models, for example, are usually resilient in relation to the chosen hyperparameters [229], and the number of models generated can benefit from tables/plots that data scientists are more used to, for example. Perhaps for Gradient Boosting algorithms [36] and projects in which many models are trained — and that any performance gain is significant — HiPlot has a say. The TensorBoard HParams Dashboard [1] also offers similar functionality, as well as Optuna [4], a hyperparameter optimization package (for ML and DL).

For fraud detection, but targeted at fraud analysts, VaBank [179] offers a three-way interface to help characterize fraudulent patterns (in banking datasets). In this tool, transactions are represented by glyphs that encode a set of common and relevant data fields. With this in mind, the first panel, the Transaction History view, is a kind of *binned scatterplot* (there are also interactive histograms for each axis), where the glyphs are arranged according to the associated amount of money (Y-axis) and time (X-axis). The other two panels offer two different projections (a matrix/grid and a force-directed graph) of the results of a SOM (a dimensionality reduction method) to complement the first panel.

In terms of presenting information about models and reporting, model cards [188] (a kind of one-pager) offer a detailed framework. Some sections, at least, could be useful to provide a clear and standardized summary of the models that a data scientist will be using and what practices a data scientist should keep in mind for reporting, since information about a particular model may be used in the future as a hypothesis to be evaluated against others, for example. In the same vein, with a special focus on data ethics, there is deon [71], a well-founded CLI tool to generate a checklist to enrich the documentation for DS projects.

2.3.4 Takeaways

There are some general conclusions to highlight:

- The discrimination threshold plot from Yellowbrick [30] is one of the options directly adapted in order to provide a plot that shows the overview of the various performance metrics (considering the range of classification thresholds) for the sake of threshold tuning.
- Precrec's idea of partial ROC/PR curve charts [258] (already used at Feedzai) will also be adapted for MevaL since there is an interest in analyzing in more detail a certain subspace (the FPR values of interest are usually small values, up to 5%, for example).
- Of the tools/packages for Model Evaluation analyzed, the one closest to a production-ready tool is WIT, given its constant development and flexibility since it was made available. Yellowbrick [30] is also a very stable Python package, especially for use in conjunction with scikit-learn [217]. In addition, Yellowbrick also powers PyCaret's Model Evaluation (or Analysis) capabilities [6] with its charts (PyCaret is a low-code ML package for Python). As a side note, PyCaret also uses scikit-plot [203] for the lift and cumulative lift curve charts.
- In terms of granularity/Model Evaluation categories, there seems to be no trend, at least one clearly visible. Over time, the overall motivation, in terms of understanding and/or improving the performance of ML models, seems to remain the main high-level motivation. The main difference is in the specificity and extent of the proposed solutions. Nevertheless, different granularities and interactivity are very useful.
- In general, all papers present out-of-the-loop (OOTL) solutions, that is, solutions parallel to the typical development environment used by data scientists. There are no proposals that present an Interactive Machine Learning (IML) system that allows the creation of an e2e ML pipeline and has a significant set of functionalities in terms of Model Evaluation, for example.
- Within the sample of papers analyzed on Model Evaluation with visual support, there is a general gap in the way the evaluation is made. Some papers have a good

number of participants, compare the results with statistical tests and have made an assessment over time, as in the case of WIT, for example, but, in general, the proposals were tested only in the context of User Acceptance Testing (UAT) with simple tasks and datasets, without the complexity of the real world. Although these works are developed in an academic context, there is a bit of a contradiction here: is it really necessary a tool with such functionalities to evaluate models trained in toy contexts?

- Neither visual tool (except WIT) has a maturity/popularity level close to the packages used in DS/ML in a second tier (being the first tier for scikit-learn, for example). In terms of packages, there are Orange, Yellowbrick, and PyCaret as pertinent (and actively maintained) options for use as second-tier packages.
- Some features seem interesting to test, but their value is still unclear. These features are the odds ratio [147], Manifold's geo(spatial) feature view [168], and Prospect's incorrectness versus label entropy plot [215].
- In addition, although two possible statistical tests have been identified to compare classification models [245], it is still necessary to further study if these are the most suitable tests for the intended objective.
- The ROC curve packages [92, 135, 162, 250, 257, 258] and the periphery plots [201] are tools to keep in mind to adapt and use in the improvement of the charts for ROC curve analysis and model decay assessment considering different subsets of data, respectively.
- For cases where the development of model-specific features is a priority, the temporal confusion matrix [172] and HiPlot [103] are two simple, yet powerful ideas to adapt. Appendix B contains a prototype of a binary temporal confusion matrix.
- Currently, the DL Model Evaluation subfield is very active, as can be seen through the survey prepared in 2018 [111], for example.
- In 2019, the number of papers making the code available increased considerably (uncommon before 2019).
- In 2019, the first papers combining visual tools and bias/fairness in ML began to emerge (FairSight, FairVis, and partially WIT). Aequitas appeared in 2018 but underwent a review in April 2019. These works are consistent with the current concern with bias in data and ML models [13].
- Many of the visual tools presented "ignore" the presentation of model performance in the most simple and traditional way, such as a number for Recall, for example. Perhaps it is for the sake of brevity and objectivity.
- Some of the older papers (approximately pre-2016 papers) seem to have, at least in part, a motivation for improving the confusion matrix concept.

- The rule-based option, RuleMatrix, shows, at least OOTB, that rule-based methods can be computationally costly.
- Most papers come from agnostic development contexts, while two papers are linked to healthcare [147, 148] and one to biomass prediction [350].
- Some of the papers feature visual tools whose target group are also Domain Experts (DEs) who do not master DS/ML. However, most proposals aim to help data scientists.
- There is no visible gap between papers looking to address model comparison or not.
- Only two papers highlight change tracking functionality, and only one paper devotes a paragraph to talk about the flexibility of the proposed solution for imbalanced datasets [8].
- In terms of dimensionality reduction techniques for high-dimensional DV, only t-SNE [175] is used [3, 172, 352] (VaBank uses SOM [179]). t-SNE is a non-linear technique for dimensionality reduction. Intuitively, it tries to minimize the divergence between two distributions [216]: (1) a distribution that measures pairwise similarities of the input instances and (2) a distribution that measures pairwise similarities of the corresponding low-dimensional points.
- When looking at vendors of DSML platforms [100, 186], it is possible to notice that, in addition to performance metrics, prediction/model runtime/speed is also a metric to consider.

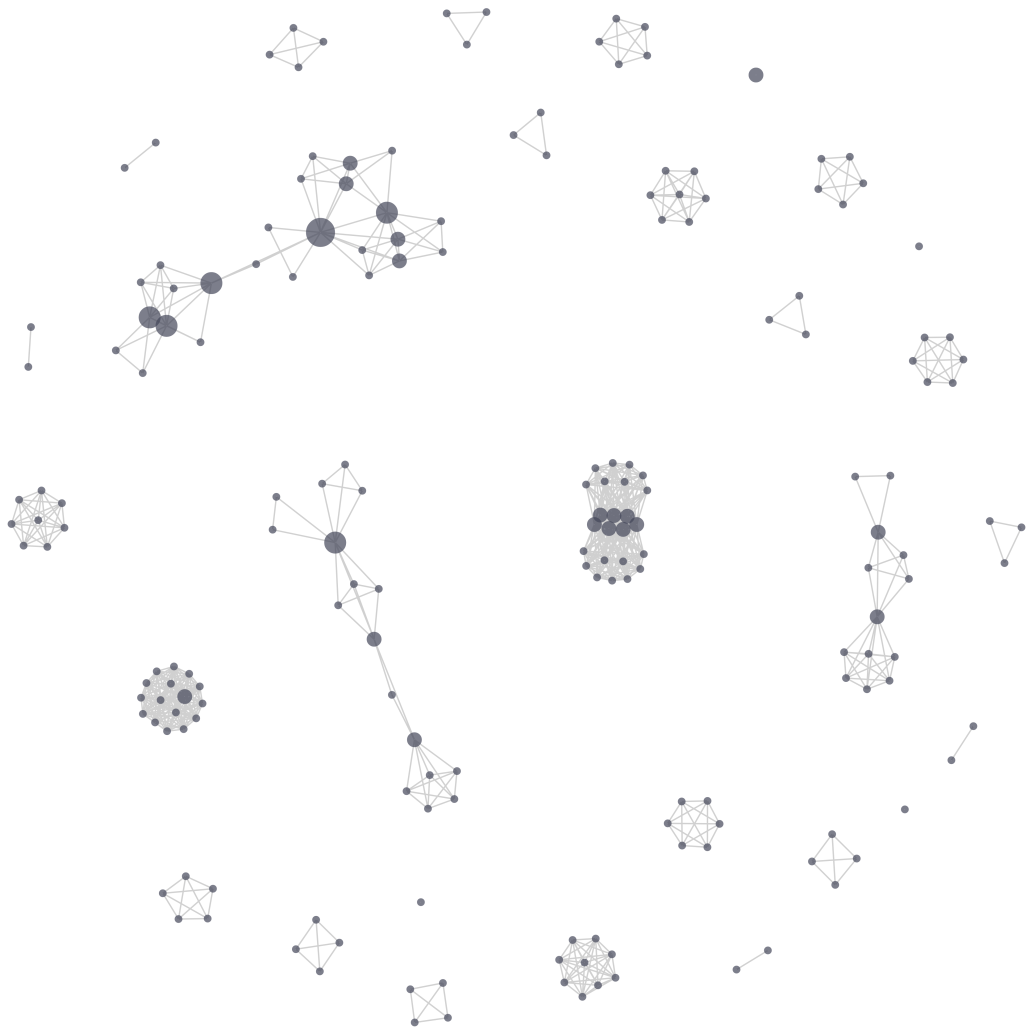


Figure 2.15: Coauthorship network based on a sample of 46 papers and 186 authors discussed in the Literature Review section. These papers are focused on DV or have at least one DV component considered significant. Isolated nodes, that is, nodes referring to the unique authors of certain papers (nodes without edges) are maintained in this graph. In this sample, the author with the most papers is Steven M. Drucker (four papers [12, 112, 215, 272] in total). An interactive version (with tooltips) is publicly available online [212]. The script (and some additional information) used to generate this graph can be found in Appendix S.

Chapter 3

MevaL, a Python Package for Visual Model Evaluation

3.1 Introduction

In order to introduce MevaL, the Python package for visual Model Evaluation developed as the solution to this effort, and its inner workings, this chapter is dedicated to the concrete definition of the scope and requirements of the project, as well as to the top-bottom description of the package. Thus, each section will cover the following:

1. Requirements Elicitation: this section is dedicated to describing and consolidating the requirements identified for MevaL derived from the input collected. This section is also devoted to organizing and contextualizing all pre-development moments of interaction with data scientists in order to qualitatively validate some ideas and iterate MevaL.

2. Tech Stack: this section is dedicated to the technologies and packages that support MevaL.

3. Package Architecture: this section is dedicated to elaborating on the structure and organization of MevaL (as a Python package).

4. Aesthetics and Configuration: this section is dedicated to the style specifications for the plotting part of MevaL, as well as some configuration considerations for pandas.

5. Features: this section is dedicated to detailing the data processing and plotting capabilities available in MevaL. The description of each implemented chart can be found in this section.

6. Deployment: this section is dedicated to the process and status of moving MevaL to production, that is, to integrate the developed package in Feedzai's DS environment and make it widely available to all Feedzai's data scientists.

3.2 Requirements Elicitation

From the beginning, there was a clear research question with a hazy path: how can we improve, accelerate, visually support, and diversify Feedzai's current Model Evaluation capabilities to enable data scientists to boost their daily work and the quality of their models? In simpler words, how could we help Feedzai's data scientists to validate their ML models?

In order to clarify this path until the definition of small concrete objectives to guide the development of the project, this section will start by addressing the process of collecting input and ending with the list of requirements collaboratively assembled to be respected.

3.2.1 User Interviews and Input Gathering

Initially, in order to align the research question with a concrete action plan, two sources of input were leveraged (in addition to the literature review): Feedzai's data scientists (the main target group) and Feedzai's Knowledge Management (KM) system (the scalable way to access the main target group over time using historical documentation).

In order to collect input from Feedzai's data scientists, semi-structured user interviews (aiming at insights, not statistics) with six Research data scientists (working or with previous experience in ML) and five Customer Success data scientists working on different projects were designed and conducted (Table 3.1). Given that there are two major personas (or profiles) at Feedzai working in DS, it was decided to consider both at this stage to collect the desired input about the workflow and their Model Evaluation needs, as well as to break any unconsciously established assumptions — and ultimately to decide the persona for this project given the considerable heterogeneity between the two (there is also a possible third persona, the *external* data scientist, from Feedzai's clients, who was not considered). The first interview, with a Research data scientist, also served as a pilot in order to improve the agenda and receive feedback on facilitation. The sample size was decided based on the opinion shared by experienced people at Feedzai (supported by previous initiatives), as well as the cost-benefit perspective that the cumulative input of five people may be sufficient [208, 270]. In addition, this (approximate) number was considered especially due to the time available, either for conducting the interviews themselves, either to compile the results, and the impact of scheduling interviews lasting between 45-60 minutes with different people busy with their normal responsibilities. In this way, the interviews were designed with the help of some User Experience (UX) materials [25, 69, 204, 209, 273] and the guidelines can be found in Appendix C.

Table 3.1: Department and starting date at Feedzai of interviewees. User interviews took place between December 30, 2019 and January 16, 2020.

Department	Starting Date
Customer Success	May 2018
Customer Success	October 2018
Customer Success	January 2019
Customer Success	June 2019
Customer Success	September 2019
Research	January 2018
Research	February 2018
Research	October 2018
Research	October 2018
Research	May 2019
Research	July 2019

Simultaneously, the interviews also made it possible to clarify some open (technical) questions when this project started:

- Should we extend the Model Evaluation capabilities available on Pulse or develop a solution from a clean slate?
- Is a Python package a reliable option?
- What types of charts do data scientists use regularly (if any)? And for what?
- Are there any needs that could benefit from a visual representation and that are well defined?

The main lessons (both general and actionable) to keep in mind are:

- One of the tasks that Customer Success data scientists need to do is check specific breakdown fields and see different performance metrics in different data subgroups. A breakdown field, in this context, is a feature of a dataset or its own set of unique values (or just a subset of them) by which a dataset is *divided* to treat the derived slices as individual datasets to be compared (since the performance on these slices is relevant in addition to the overall performance). So, it is important to allow flexibility in terms of the number of breakdowns and/or models allowed per table/visualization. In addition, this task is mostly made outside of Pulse (via computational notebooks and/or scripts).
- There is significant variability between workflows (considering different projects/-clients). The use of Pulse, compared to computational notebooks and scripts, varies significantly between Customer Success data scientists and projects.
- Pulse is Feedzai's *go-to-production* platform, with useful features to (at least) perform the first screening of the models, but the requirements demand the flexibility provided by custom code.

- Although possible new features, in an ideal scenario, should be implemented on Pulse, it is feasible to consider creating a Python package (at least for Feedzai's data scientists). However, some requirements must be incorporated by design: in terms of usability, the package must be stable, easy to use, modular and be equipped with good tutorials and documentation.
- No ML Bias and Fairness audit needs were identified.
- ML Interpretability methods are useful for communicating with clients and for speeding debugging.
- The duration and pace at which projects are carried out can vary, which correlates with faster Model Evaluation and with slower and more in-depth Model Evaluation. However, given the Big Data context in which Feedzai operates, it is important to keep in mind the computation time required to iterate and that this can influence the project to be slower and more static, but with a not so deep Model Evaluation — each project is a unique project. Somewhat independent of this, is the fact that there is always more that could be done and data scientists have limited time. The preparation of data and features, as well as environment configurations, takes a considerable fraction of the available time and strongly influences the time for other actions. Therefore, it is important to provide tools for evaluating models as OOTB as possible so that the focus is on hypothesis testing rather than preparation for testing hypotheses.
- There are (approximately) two major phases in client projects: the initial phase, with only historical data and without a functioning production environment, and the maintenance and iteration phase, over time, after the first model goes into production (there is a kind of calibration period between both when the model is in shadow mode). From the second phase, (online) model monitoring is a fundamental aspect and, with that, the return to the (offline) DS environment for possible iterations and/or model retraining. This implies, therefore, that new Model Evaluation procedures are made with the snapshot of the old model in mind. In this way, there is a new reference point for several decisions that must be carefully considered.
- FI is very important and used (to simplify models by reducing the number of features, for example).
- The use of model-specific methods to evaluate ML models is rare (to understand how splits are being made in a RF model, for example).
- Feedzai provides a system that is not only composed of ML models but also (explicit and manually defined) rules (or just rules, depending on the use cases). Thus, in certain projects, it is also fundamental to evaluate the performance of the system as a whole, in a flat or hierarchical way.

To complement the user interviews, some content of the Feedzai's KM system was also checked and documented in order to collect potential ideas, to understand part of the reasoning for the current state, and to confirm if there were, in the past, possible ideas that did not work (or were still open). Pulse, Feedzai's e2e ML platform, was also set up locally, tested, and some conclusions drawn, which were later addressed in a meeting with a Customer Success data scientist. This meeting also served to collect some input in a similar way to the interviews.

Finally, a meeting was also conducted with the principal responsible for a similar initiative at Feedzai that resulted in a Python package (and computational notebook) for Exploratory Data Analysis (EDA). This meeting mainly served to address the various stages of development and to highlight the importance of creating and validating a mockup (the recommended format was that of a computational notebook) before moving on to coding (more information in the Mockup section). As a side note, flexibility was one of the criteria that motivated the development of a Python package for EDA.

3.2.2 Persona

The main persona for this work is the Customer Success data scientist at Feedzai. This narrowing occurred for the following reasons:

- First of all, the work, at least at the level of Model Evaluation, is significantly more standardized (in a simplified way, Model Evaluation is a performance metric-centric job at Feedzai) for Customer Success data scientists (compared to Research data scientists).
- In Research, the focus is on defining heterogeneous proofs of concept (PoCs) that fail easily or show potential quickly. The main objective is to beat a pre-established baseline (*Is this new method better or not?*) and ensure that the experimental setup is adequate so that new alternatives can be generalized for Feedzai. So, it's difficult to have a setup or package that would be general enough to work for everything — proper support is on the side of guidelines and processes.
- Due to the limited time and the lack of centralized and programmable tooling for Model Evaluation (except the features available on Pulse), the impact of a package like MevaL could be quite significant for the daily work of Customer Success data scientists.
- Given the communication aspect existing in DS projects, between peers and between data scientists and clients, a package designed with consistency in terms of charts in mind can speed up the sharing and exchange of information. In addition, at Feedzai, Customer Success data scientists already use charts in presentations and to share results, for example.
- On the other hand, the consistency, as well as the centralization of a general package (around the context of financial crime detection), may allow its extension in the future, in addition to ensuring that all Customer Success data scientists have access to the same options ready to be used.

In general, this profile and the associated work have the following characteristics:

- The Customer Success data scientist is focused on obtaining better results, based on pre-established performance metrics, and on fitting a solution for a specific use case.
- Depending on the project, the Customer Success data scientist has variable time to devote to Model Evaluation.
- In general, the Customer Success data scientist knows the relevant data subgroups that he/she needs to take into account when evaluating models *a priori*.
- In addition to in-house platforms, the Customer Success data scientist also uses scripts and/or computational notebooks (although preference varies from person to person).
- The Customer Success data scientist knows how to use the main tools of the Python-based DS tech stack (adopted at Feedzai), namely pandas, Jupyter, NumPy, and PySpark (Spark Python API).

3.2.3 Scope

In order to clarify what MevaL is and what it is not, the package fits as a reliable tool according to a set of attributes that make up its scope:

- MevaL is a Python package designed and tested to be used in JupyterLab environments.
- MevaL is intended for post-modeling, pre-production evaluation (or, simply, Model Evaluation). However, MevaL can also be used to compare new and old ML models, that is, offline models and models in production.
- MevaL is single model-first. This means that the API for each chart is designed for one model at a time, even though some charts can be adapted for more than one model — for now, comparing models can be done using different charts plotted side by side through different calls (in a way inspired by the small multiple concept [29] adapted to a computational notebook layout).
- MevaL is a visually supported package. Although, in practice, it is necessary to have a data processing component to improve the user experience and ensure that there is a sufficiently small representation of the original data (with an expected maximum of 10,000 rows, with only a part of these rows being plotted per chart) and with the necessary transformations for analysis, all other features have DV in their core.
- MevaL is designed and tested to work, in the first place, in Feedzai's DS environment, respecting its dependencies. Basically, MevaL is an environment-dependent package.

- MevaL is a model-agnostic package. In other words, MevaL works from datasets that can be loaded and analyzed. In addition, it is raw data-friendly by having an optional data processing pipeline. These raw datasets may or may not have been sampled.
- MevaL is oriented towards binary classification problems.
- MevaL is structured mainly for the second phase of Model Evaluation, after the first screening of a possibly large number of models (mainly through a leaderboard-like, metric-oriented table, like the table generated by the mljar-supervised package [189]), where the maximum expected number of models is five (the comparison of 2-3 models is expected to be the most recurrent scenario). This number may be higher for short-term projects (10-15), for example, but time will not allow such an in-depth analysis — however, MevaL, with some of its features, can be partially used in the process as well.
- MevaL is table-aware. In other words, MevaL sees tables (more specifically pandas DataFrames) as visualizations. This is particularly useful for the first screening of models, based on different performance metrics.
- MevaL follows three simple, yet well-defined and purpose-based (dataset) schemas, that is, it follows a (reduced) set of rules for the definition of a schema suitable for performance metrics, predicted scores or feature importance. The details of these schemas can be found in the Data and Data Processing section.
- MevaL is a threshold-driven package, that is, given the importance of adjusting and analyzing various classification thresholds, sometimes even more than a threshold for the same model, depending on the value of a particular feature, for example, all relevant charts can be prepared according to a certain classification threshold (there is also a specific chart to visualize the performance metrics over the range of classification thresholds). In this way, MevaL does not assume anything about the classification threshold to be inspected — it only provides an API that allows the user to prepare the appropriate chart for a specific classification threshold, explicit or derived from the value of a performance metric for some cases, as well as a processing pipeline that includes a range of thresholds in the metrics dataset to be analyzed.
- MevaL is a low-code package, with a set of functions and classes that are easily usable and that require only the appropriate parameterization by the user (thus seeking to speed up the experimentation cycle, in addition to lowering the barrier to leverage DV). In this way, MevaL abstracts the implementation details, allowing the user to focus on what he/she wants.
- MevaL is a package designed for data scientists, that is, designed for knowledgeable people, who have the sensitivity to interpret the results and know how to critically interpret the data shown by MevaL. However, this principle does not deny that

different data scientists will have different levels of experience/backgrounds and that, at least initially, not all data scientists will be acquainted with all the charts. Thus, in addition to knowing the target audience, it is essential to ensure that the documentation and tutorials provided are adequate to support all Feedzai's data scientists.

3.2.4 Requirements

From the content discussed in the interviews, as well as all the material collected, it was possible to establish a set of requirements to guide the development of MevaL, as well as the questions that the package seeks to answer explicitly. To adjust the requirements defined after a first iteration, a meeting with stakeholders was scheduled in order to present the findings and finalize the list of requirements. Subsequently, each requirement has been carefully described in a product requirements document (PRD)-like document to provide a kind of single source of truth and a way to share this list with potential stakeholders.

That said, the list of requirements (or design goals) consists of the following items:

- R01: Data readiness
- R02: Flexibility
- R03: Modularity
- R04: Pluggability
- R05: Usability
- R06: Work-based relatability

The first requirement (*R01: Data readiness*) is concerned with the availability of data ready to be plotted (and analyzed), either from a raw scored dataset (that is, from a dataset with the predicted scores of a ML model and the features used), or from a dataset processed independently with a valid schema. In other words, MevaL must support a mechanism that allows extracting a sufficiently small representation with the relevant information from one or more raw datasets (each one for a specific model), that is, it must support a data pre-processing pipeline that is easily triggered and performant (it needs to be fast enough according to Feedzai standards so that this component does not become a clear bottleneck). On the other hand, starting from a pre-specified schema, MevaL must also be able to use the data present in datasets already processed by extraneous mechanisms. In this way, MevaL can accommodate different workflows, and datasets previously processed and persisted. In a nutshell, MevaL must be ready to plot data from different datasets coming from different sources and transformations.

The second (*R02: Flexibility*) and third (*R03: Modularity*) requirements come from shared motivation for the entire package at the structural level, but are applied to different granularities. On the one hand, R02 is concerned with a user experience that is customizable according to the needs in each individual case, that is, R02 is concerned that each feature has workable defaults and the possibility of gradually customizing more parts; on the other hand, R03 is concerned with decoupling the individual building blocks that holistically constitute MevaL, that is, with the modular interpretation and use that must be possible with each subpackage and module. In addition, R02 also seeks to ensure that the plotting API for each feature is as consistent as possible, while R03 enables the use of different subsets of features freely and MevaL's own extensibility for the future (without discouraging the application of the package as a whole).

From a more contextual point of view, the fourth requirement (*R04: Pluggability*) is concerned with the easy integration of MevaL into Feedzai's current DS environment and that any data scientist is able to use the package as another tool that he/she uses daily. This environment, in terms of front-facing details, is made up of Pulse, JupyterLab, and a set of Python-based dependencies, with the current supported Python version being Python 3.6. Thus, MevaL must be aligned with these technical requirements in order to be integrable from the beginning — at the end of the day, MevaL is a tool for Customer Success data scientists.

As expected, the fifth requirement (*R05: Usability*) connects a number of key points for the adoption and use of a package like MevaL. That said, MevaL must be grounded on an intuitive and exemplified API, appropriate documentation, an entry-level notebook, and duly explained visualizations, so that the learning curve is of rapid progress. This requirement is also linked to the first one (*R01: Data readiness*) in the sense that data preprocessing should not be an obstacle to the adoption of the package.

Last but not least, the sixth requirement (*R06: Work-based relatability*) addresses the duality between existing and actually used charts and tasks that would benefit from new visual representations (according, above all, to user interviews). From a more practical point of view, MevaL should support existing charts for Model Evaluation at Feedzai and should provide charts designed with the potential to cover specific tasks that may benefit from visual support. From the user interviews, no evidence was found on the usefulness of looking for visual alternatives for Model Evaluation tasks that are not currently carried out at Feedzai. That said, this requirement is just an umbrella for a set of specific tasks, or design goals, that integrate the current Model Evaluation procedures — these options are described in the Model Evaluation Topology section and together they make up a kind of taxonomy for Model Evaluation.

3.2.4.1 Prioritization (Preview)

After the user interviews, and in the same document mentioned above, a first crude list of priorities was also defined that helped shape the development phase (the first tier turned out to be the focus of this master's thesis):

1. **Tier 1** (the must-have features): features that fill the current Model Evaluation Topology/functional requirements (with particular attention to the design of tables

and informative summaries). FP/FN Analysis is considered the least priority task at this tier as it involves a strong component of Data Exploration (and Feature Engineering). Within this tier, features can be ordered in terms of complexity (ascending order) and current adoption (descending order). The main idea is to start with the features that are easier to implement and that Feedzai's data scientists already use, such as ROC curves, and go deeper into the possible available options.

2. **Tier 2** (the nice features to have): model-specific features for RF and LightGBM models, *incorrectness versus label entropy* plot [215], Data Exploration/Feature Engineering-oriented features for FP/FN Analysis, and ensemble modeling via non-adversarial Model Comparison.

3. **Tier 3** (the features to explore if possible): metadata weight comparison (to compare the weight of different models using the number of trees, the number of features they use, among other aspects) and geospatial analysis (Manifold's Geo Feature View is an example [168, 349]).

Focusing on **Tier 1**, the associated prioritization was designed to conform to the order approximately dictated by the following factors:

1. **Complexity**: the simplest features, both in terms of implementation and in terms of use, consisting of a smaller number of parts, should be the first to be implemented. Complexity will not be measured objectively by any metric. Although, throughout several iterations, a given feature may become more complex than another, it will be considered an estimate of complexity based on the current perspective.

2. **Current adoption**: the features that are already implemented on Pulse, as well as the features related to them and identified through the user interviews, should be implemented first. Current adoption can also be seen as a proxy for the usefulness of features in the short term.

3. **Availability of ready-to-use files**: since, through Pulse, it is possible to obtain a set of files ready to consume, the features that can use them by design should be implemented first. The features that require manipulating the original data file scored by Pulse should be addressed later. In the end, it turned out to be the other way around (more information in the Data and Data Processing section).

4. **Consumption-oriented**: the initial approach should be focused on the consumption of existing files, that is, MevaL should load the relevant files. Thus, features that solely depend on existing files *a priori* should be implemented first. The features that need to communicate with Pulse for the generation of new files (that must be later consumed) should be implemented later.

5. **DV-oriented**: features whose charts or tables are the main element should be implemented first, compared to features in which the most visual parts are a complementary or null element.

3.2.5 Mockup

In addition to the theoretical settlement for MevaL, a mockup was also developed, in the form of a (Jupyter) notebook (with images and some text), aimed at showing the possible capabilities of the API and simulating the approximate usability that a data scientist would have with MevaL. It is important to highlight that the mockup was developed from the perspective of the API/package as a flexible tool to be used according to the user's preferences, that is, the mockup did not serve to simulate what may be, step by step, a standardized Model Evaluation procedure for Feedzai.

So, in order to collect some (initial) feedback on the possible concrete features to be implemented in MevaL, five (mockup review) conversations/interviews (the guidelines can be seen in Appendix V) were conducted with Customer Success data scientists (this number was defined in a similar way to that presented in the User Interviews and Input Gathering section). Of these five conversations, three were with data scientists previously interviewed and two were with data scientists whose first contact with this project was during the conversation. The motivation to collect this feedback, in addition to keeping the development of MevaL close to its future users, was related to the need to carry out the first screening, as well as to collect a more concrete list of ideas than that obtained during the interviews about possible customizations for the features, so that they are designed with the needs of data scientists in mind. These conversations also served to address some relevant questions raised in the final part of the conversations, as well as to ascertain whether there was any missing fundamental feature according to what Customer Success data scientists currently use/do (with the awareness that the sample of the conversations may not be fully representative).

That said, the mockup approached the following structure (the chart images and helper text were arranged in a narrative based on questions and answers):

1. Imports
2. Configs
3. Question (Markdown)
4. Feature/Chart (mock)
5. Answer (Markdown)

Finally, from these conversations, some preliminary new ideas emerged, such as:

- Create and update the standard notebook for Model Evaluation as new features are developed and implemented in the Python package. This should complement an easy-to-use API with a clear statement of functionality. The notebook can also serve as an introduction or tutorial for the API, as well as a first standard version of Model Evaluation.
- Implement a summary of the models and datasets to be analyzed.

- Implement interactive legends or a similar mechanism to show/hide information.
- The score distribution chart must have an interactive mechanism to change the bin width or its function must have an argument to define it flexibly (a small multiple can be an option as well).
- Implement the strip plot (considering a version for Threshold Tuning as well) and add a tooltip to each strip.
- Confidence intervals can also help in communicating with clients.
- Include tooltips in the classification threshold plot. Also, allow to zoom in on the Y-axis or limit the values.
- Consider that performance-based tables and charts, when being used for Threshold Tuning, will have to be flexible to allow Customer Success data scientists to check different thresholds for different breakdown fields.
- Implement a parameter for the amount (cost-sensitive) field.
- More important than a ROC curve chart is a flexible partial ROC curve chart.

3.3 Tech Stack

For MevaL to be possible, a set of specific technologies (packages and extensions) was leveraged (mainly Python-based) in order to guarantee the most robust development possible and in line with the practices followed at Feedzai (not to mention the suitability for the development of the desired features). Although most of this stack was built to match Feedzai's DS environment, all technologies used are open source. These technologies can be divided into three major groups (a list of them, as well as the versions used and their project-specific purpose, can be found in Table 3.2 and Table 3.3, respectively):

1. Plotting (*Front-end*) tools: since MevaL is a package for evaluating ML models with visual support, it is necessary to have suitable technologies to build and show different charts.
2. Data and processing (*Back-end*) tools: since MevaL is a data-driven package, dependent on certain consolidated representations from the original data used to train/test a given ML model, as well as its output, it is necessary to provide a way (pipeline) to map the low-level raw data to the higher-level information in order to ensure the extraction of insights in a practical way. In addition, implicitly, it is also necessary to consider certain data structures that allow data (from external datasets) to be handled and programmatically maneuvered, that is, a format for representing (tabular) data (in Python).
3. Code quality tools: since MevaL is a dynamic package to be used by data scientists in order to help them develop ML models for their projects, it is necessary to ensure that its codebase is as robust, stable, and consistent as possible. In other words,

from an agnostic perspective in terms of functionality, it is essential to enrich the package with a set of heterogeneous validations and standardized formatting that will increase the quality/readability of the code (from a practical point of view, it is very difficult, perhaps impossible, to compare the code being developed with a large list of good practices/warnings constantly) and facilitate future maintenance/extension of the package itself. Thus, although part of this work needs to be done by hand, it is composed of a set of cumbersome, error-prone, and non-scalable steps in general — the ideal scenario to use certain tools (from logical and stylistic linters to unit testing frameworks) that help in the execution of various quality assurance tasks in an (approximately) automatic way.

Regarding the plotting side, Altair was the chosen package (the other possible and supported option was Matplotlib). Altair is a declarative visualization package for Python that provides an API for generating web-based Vega-Lite charts (which are automatically embedded in JupyterLab notebooks). In other words, Altair is like an alternative Python-based syntax [195] that generates Vega-Lite specifications (*chart blueprints*) — in the end, the visualizations are still rendered with Vega-Lite (JupyterLab comes with built-in support for Vega and Vega-Lite [196]). Thus, Altair is part of the Vega ecosystem — Vega is a JavaScript Object Notation (JSON)-based declarative language for describing and creating interactive visualizations (Vega-Lite is the high-level counterpart) [213] inspired by Leland Wilkinson's influential *The Grammar of Graphics* [339] (building blocks for DV). The Grammar of Graphics (GoG) is a well-established framework for composing charts from different independent parts [317], streamlining the mapping between a dataset and the available visual encodings (it is also at the foundation of ggplot2 [315], a declarative visualization package for R). That said, the reasons (partly due to the ecosystem in which Altair operates) that fueled this choice were as follows (in no particular order):

- In addition to Altair being supported in Feedzai's DS environment, the other in-house MevaL-like package for EDA uses Altair.
- Altair provides OOTB interactive charts (through a high-level interaction grammar entangled with the rest of the grammar/methods [264]), extending the possibilities in a similar way to existing ones for producing static charts. In addition, it is also possible to add OOTB tooltips, a way to complement the chart based on the user's interests (via mouseover) through table-like information on the different plotted marks. A tooltip is a tabular-like text box [290] that provides extra information upon user-driven mouse hovering.
- Since Altair is the visualization package for Python currently used by Feedzai's DV team (and it is the package used in the EDA package), there is significant know-how at Feedzai about it.
- Altair has sensible, workable defaults, as is the case for color palettes [264], thus allowing implicitly to leverage its declarative vein.

- The Vega ecosystem, where Altair is inserted as a front-facing package for Python (the stack goes like Altair → Vega-Lite → Vega → D3 [164]), is strongly backed up by research [105]. After the first version of Vega in 2013 [263] (and other prior actionable works, such as Prefuse [38] in 2005 and D3 [38], Vega's cornerstone/kernel and one of the most powerful plotting tools out there [164], in 2011), the first research paper [266] on language design/implementation appeared in 2014, followed by a set of other works, such as Vega-Lite [264] in 2017 and Draco [199] (a Vega-Lite-based visualization design tool) in 2019, culminating, for now, in two research papers accepted at IEEE VIS 2020 (a visualization conference that took place in October 2020) on smart labeling [139] (already available from version 5.16.0 of Vega [106]) and animation [137] (two of Altair's current weaknesses). The last two works also show the active development experienced in this ecosystem (Altair is at the end of the pipeline but it is also in active development).

- Altair's charts can be easily embedded in Hypertext Markup Language (HTML) files (an interesting medium to create and share reports, for example) as fully-functioning charts (in terms of interactivity, for example) or statically (as an image). In other words, it is possible to use a Vega-Lite chart, created with Altair, in other environments outside the Python and Jupyter ecosystem.

- In addition to providing the ability to create multiple individual charts and combine them into a single (layered or concatenated) multi-view/compound chart, Altair also offers a kind of chart parameterization mechanism to generate multi-view (faceted or repeated) charts (via certain methods/operators). In other words, from the typical code block to define a *unit* chart, it is possible to obtain multiple charts (as a multi-view chart) according to the variables used. This convenient API, smoothly integrated with the rest, for creating varied views of a dataset (for a particular type of chart) is particularly useful for small multiples (significantly leveraged by MevaL) and scatterplot matrices [225].

- Altair brings Vega-Embed [198], a small package for embedding charts in web environments and customizing some options, enabled by default. From a purely practical point of view, Vega-Embed adds a button (in the upper right corner of each chart) and a dropdown menu with some actions (triggered manually) useful for the user, such as the option to export a Portable Network Graphics (PNG) or Scalable Vector Graphics (SVG) image.

- Altair has a consistent, readable and easy-to-use API underpinned by the Vega-Lite (and Vega) design principles and the way it clearly instantiates the GoG [169].

- Under Altair, at the Vega level, lies an efficient system architecture previously benchmarked [265]. In simple and non-exhaustive terms, Vega efficiently updates a chart according to new input events or data changes, highlighting its reactive nature (in addition, there is also a separation of concerns between specification and evaluation) [169].

As for data and processing tools, the two main ones are pandas and PySpark, the de facto packages (at least currently) for data [246] (in-memory analytics) and data-intensive [237] (distributed computing) contexts powered by Python (Spark is a more comprehensive tool, an all-in-one Big Data project, that can also be used with R, Java, and Scala, for example). In addition, they are also two of the standard tools at Feedzai for DS and ML. The need to have data processing tools that allow the transition from large raw datasets to small carefully thought-out datasets is also linked, from a practical point of view, to one of Altair's design principles/constraints: limiting the number of rows in a dataset that will be embedded in the chart specification generated from Altair in 5000 [301] (by default). This mechanism is particularly tailored for pandas DataFrames, as the respective data is converted to JSON and included in its entirety in this specification, that is, in the Vega-Lite *scaffolding* that defines a Altair chart from its API. Above all, although it is totally possible to remove this restriction in a fruitful way, this idea attracts holistic thinking for the entire solution, from raw data to the notebook, and draws attention to potential problems in the size of notebooks and performance, especially in presence of multiple plots (this restriction is maintained by default in MevaL). In terms of implementation, it also invites a careful choice of a subset of processed data, if applicable, at the time of plotting. Moreover, supporting these two tools are NumPy and SciPy, as they are used for specific data transformations and for statistical computation.

In terms of code quality tools, in addition to efficient support for the iterative development and maintenance of a certain level of quality, there are some idiosyncrasies regarding certain tools that deserve to be highlighted:

- nbQA [91], by enabling static code analysis and code formatting of the cells of a Jupyter notebook through a simple, one-liner command, was particularly useful for running Black on development and demonstration notebooks. This formatting allowed, for example, to break function calls with a significant number of parameters on different lines, facilitating their readability and customization.
- Pyroma [247] made it possible to compare the *setup.py* file, the metadata-based build script, which is essential for creating Python packages, with a list of best practices, allowing this non-priority file to be improved effortlessly.
- flake8-docstrings [15] was especially useful to ensure that the Python docstrings followed the NumPy convention [304] (the convention recently adopted at Feedzai for Python packages like MevaL).

As a side note, since the Feedzai's DS environment does not have one of the dependencies necessary to prepare Altair charts as images programmatically, a JavaScript-based workaround was developed and briefly used. This component is detailed in Appendix G.

Table 3.2: Each version of the different technologies leveraged for the development and maintenance of MevaL. Some of the options have two versions as these dependencies have been updated according to Feedzai's DS environment. At the base of this stack is Python 3.6.10. The version of *markdownlint* corresponds to the version used for the Visual Studio Code extension.

Tech	Version
Altair [297]	2.4.1
Bandit [232]	1.6.2
Black [161]	19.10b0 and 20.8b1
Flake8 [353]	3.7.9 and 3.8.3
flake8-bugbear [160]	20.1.4
flake8-docstrings [15]	1.5.0
isort [55]	4.3.21
JupyterLab [140]	0.34.12 and 1.2.6
markdownlint [17]	0.37.1
mypy [165]	0.780
nbQA [91]	0.3.5
NumPy [99]	1.14.5 and 1.18.5
pandas [180]	0.23.4 and 0.24.2
Pipenv [224]	2018.11.26 and 2020.8.13
Pyroma [247]	2.6
PySpark [346]	2.3.1 and 2.4.5
pytest [149]	3.6.3 and 5.4.3
SciPy [303]	1.1.0 and 1.4.0
Vega-Embed [198]	3.30.0

Finally, JupyterLab is like the workbench, that is, in this work, it is used as the main platform for the execution of Model Evaluation procedures, given not only its use in Feedzai (as an extensible platform for routine DS tasks and manual or automatic reporting [279]) and the community in general [128] but also its native support to show (interactive) visualizations. On the other hand, JupyterLab, together with Visual Studio Code [185], were the source-code editors used during this project (JupyterLab was used, mainly, to visually test the charts created). The technical specification of the laptop used during development can be found in Appendix M.

3.4 Package Architecture

The general project structure for MevaL can be seen in Figure 3.1. As a Python package to be distributed internally (and developed locally), the structure followed is similar to that of a typical Python package [234, 294] (mainly based on the user guide provided by the Python Packaging Authority [235], the *official* working group responsible for maintaining a core set of resources for Python packaging). However, this structure is extended to that of a DS project [70] as well, since MevaL, in addition to being a package, is also a

Table 3.3: Overview of the purpose of each of the technologies used for the development and maintenance of MevaL.

Tech	Purpose
Altair [297]	Plotting
Bandit [232]	Security linting
Black [161]	Code formatting
Flake8 [353]	Linting
flake8-bugbear [160]	Linting (it extends Flake8 with extra warnings/error codes)
flake8-docstrings [15]	Docstring convention checking
isort [55]	Import ordering
JupyterLab [140]	Computational notebook environment
markdownlint [17]	Markdown linting
mypy [165]	Optionally-enforced static type checking
nbQA [91]	Running Python code quality packages on Jupyter notebooks
NumPy [99]	Data manipulation
pandas [180]	Data structuring, processing, and manipulation
Pipenv [224]	Dependency and environment management
Pyroma [247]	Python package metadata (*setup.py* file) checking
PySpark [346]	Data structuring and processing
pytest [149]	Unit testing
SciPy [303]	Statistical calculations
Vega-Embed [198]	Chart embedding and image saving (used by default in Altair)

repository and, therefore, will contain some folders and files that will not necessarily be packaged/distributed but that can be consulted and used from the repository (like demo notebooks, for example). That said, there are some opinionated details that complement/contrast the examples found online:

- The *notebooks* folder contains the computational notebooks for development and showcase, as well as the respective supporting datasets (relatively small raw datasets and small processed datasets). These two types of artifacts are in the same folder given the purpose shown by the two types of notebooks hosted here.
- Pipenv is the environment isolation tool of choice, as it is the option used in Feedzai's main DS-oriented Python package. The other options on the table were Virtualenv and Conda [237].
- The list of adopted Python Package Index (PyPI)/Trove classifiers (metadata to categorize a given Python package) can be found in Appendix E.
- The *changelog* file (a chronologically ordered list of changes [153], basically) was updated based on the name of the closed issues for each of the merge/pull requests to the main branch of the MevaL repository.
- The versioning of the package is based on the Semantic Versioning 2.0.0 [227] specification (without pre-releases).

- All modules, except those that refer to specific types of charts (submodules), are located under the inner *meval* folder (package), regardless of whether they are *internal* modules or *front-facing* modules (all can be imported and used by the user, but some are expected and others are not). This structure was followed in order to keep the folders (subpackages) for the visual parts to be used directly by the user (the *style* subpackage was also structured in this way since it also contains some user-friendly helper functions to change the general theme for Altair charts, for example). One option studied was to create a folder called *internal* to place all modules for internal use, that is, all non-API modules, as is the case with the PyCaret package [6]. The *utils* module is also expected to be rarely used by the user.
- In the case of the *tables* module, which contains a set of Python decorators to customize the rendering of different pandas DataFrames (more specifically, a simple function created by the user to show pandas DataFrames), it was decided to separate it from the above structure designed for the OOTB charts and related helper functions, as this module is operated in a very specific way.
- Within the text files, there is the *.editorconfig* file which aims to facilitate the maintenance of the style of certain files, as is the case with the *LICENSE* file, the *Makefile*, and the Python files themselves (especially at the level of insertion of a new empty line at the end of each file or not, indentation style, and indentation size). This file is part of a project called EditorConfig [340] that provides a file format and numerous text editor plugins for code-wise consistency purposes.

Each subpackage, that is, each of the folders inside the inner *meval* folder, contains one or more files (or Python modules), each with a class for a particular type of chart. The nomenclature for this division is further explored in the Model Evaluation Topology section.

The *charts.py* file, on the other hand, contains all the functions that allow the construction of the different Altair charts available in MevaL, as well as the definition of some components (or partial charts) to be used by these charts. These functions are very close, in theory, to the "recipe" functions present in the altair_recipes package [222] or the high-level functions available in the Starborn package [267] for various types of charts (each function is named after a chart, basically).

As a side note, there is no file dedicated to custom exceptions since MevaL takes advantage of Python's built-in exceptions (like *ValueError*, for example).

To wrap up this part, MevaL can also be summarized by the following (overall) code metrics (calculated using the Radon package/CLI tool [154]):

- Cyclomatic/McCabe's complexity (*radon cc -e tests/* -s –total-average .*)
  - Average complexity (considering 267 blocks, that is, 267 classes, functions, and methods): A (2.850187265917603).

```
meval ... Root folder for the project/repository
├── <Pipenv files> ... Development environment files
├── <Python packaging files> ... The files for making a Python package and manag-
│   ing its distribution like the setup.py file
├── <Text files> ... Non-code files like the changelog/history or the README
├── assets ... Images and other miscellaneous files
│   └── <Asset> ... UML diagram, for example
├── meval ... MevaL package (root folder for the importable package)
│   ├── base.py ... Abstract base classes and evaluation interface for MevaL
│   ├── charts.py ... Functions for creating Altair charts
│   ├── constants.py ... Reusable constants for MevaL
│   ├── logging.py ... Default logger for MevaL
│   ├── metrics.py ... Functions for performance metrics and confidence intervals
│   ├── preprocess.py ... Data processing pipeline for MevaL
│   ├── tables.py ... Styling decorators to customize pandas DataFrames
│   ├── utils.py ... Utility (general-purpose) functions for MevaL
│   ├── feature_importance ... FI subpackage
│   │   └── <chart>.py ... Module with the class for a given type of chart
│   ├── model_decay ... MD subpackage
│   │   └── <chart>.py
│   ├── score_distribution_estimation ... SDE subpackage
│   │   └── <chart>.py
│   ├── style ... Aesthetics and configuration for Altair and pandas
│   │   └── <config.py>
│   ├── threshold_tuning ... TT subpackage
│   │   └── <chart>.py
│   ├── uncertainty_estimation ... UE subpackage
│   │   └── <chart>.py
│   └── visual_performance_analysis ... VPA subpackage
│       └── <chart>.py
├── notebooks ... Development and showcase data and computational notebooks
│   ├── data
│   │   └── <data>.csv
│   └── <notebook.ipynb>
├── scripts ... Convenient Bash scripts for various tasks
│   └── <script>.sh ... Launch JupyterLab locally, for example
└── tests ... Unit test suite
    └── <test>.py
```

Figure 3.1: Approximate project structure for MevaL. The *__init__.py* files for the package and each subpackage are omitted. The *preprocess.py* file contains all the functions to preprocess raw datasets and create datasets (pandas DataFrames) ready to be plotted.

  - Complexity of the best blocks: A (1).
  - Complexity of the worst block: D (22).

- Maintainability Index (*radon mi -e tests/* -s .*)
  - All files are ranked A, except the file with all Altair charts (*charts.py*) given its large number of lines (2841 excluding docstrings). In the future, this file should be separated into different modules.
  - Ranking (and score) of the best files: A (100.00).
  - Ranking (and score) of the two worst files: C (8.99) and A (33.54).

- Raw metrics (*radon raw -e tests/* -s .*)
  - Lines of code (LOC): 9722.
  - Logical lines of code (LLOC): 2378.
  - Source lines of code (SLOC): 6760.
  - Comments: 505.
  - Single comments: 522.
  - Multi: 1000.
  - Blank: 1440.
  - Comment statistics: 5% (C % L), 7% (C % S), and 15% (C + M % L).

In particular, the data processing module (*preprocess.py*) has the following Halstead metrics/complexity measures [335] (*radon hal meval/preprocess.py*):

- η_1: 13.
- η_2: 102.
- N_1: 65.
- N_2: 117.
- Program vocabulary: 115.
- Program length: 182.
- Calculated (estimated) program length: 728.6931012169267.
- Volume: 1245.8791892718764.
- Difficulty: 7.455882352941177.
- Effort: 9289.128661188843.
- **Time required to program** (coding time derived from the *Effort* measure): 516.0627033993801 seconds ($\approx$ 8-9 minutes).
- Number of delivered bugs: 0.41529306309062547.

3.5 Aesthetics and Configuration

MevaL, as a visualization and DS package, is adjusted (by default) with a set of aesthetic/style options for Altair, as well as some custom settings for pandas. All of this configuration is loaded when importing this Python package (however, given the flexibility of Altair and pandas, all of these decisions can be reversed/changed if necessary). On the other hand, there are some standard options consciously subscribed to, such as Altair's categorical color scheme (Tableau 10 [283]).

At the core of the aesthetic choices for MevaL, the following items stand out:

- Arial is the preferred (sans-serif) typeface (or simply font) for MevaL (in other words, it is the *umbrella* font for the computer fonts that will be used by default). The choice of Arial was mainly due to its familiarity, its good (arguable) design, and its ubiquity — Arial is a cross-platform, web-safe font [178] (the initial choice was Roboto, but this font is not Ubuntu-friendly).
- The initial and generic width and height values for charts available in MevaL are 300 pixels. Although different widths and heights can benefit distinct charts (it is possible to change both values in general, as well as to change them individually), these numbers were decided, mainly, based on the provided chart size, suitable for 13-inch (or larger) laptops and for the JupyterLab interface.
- The Y-axis title is positioned, by default, horizontally (zero rotation angle) in the upper left corner of the chart, near the end of the axis. This positioning, in contrast to the more traditional version where the title is rotated and centered vertically along the Y-axis, seeks to facilitate the reading of this title [41, 90, 286, 295].
- Regarding color, MevaL leverages three different color palettes, as shown and described in Figure 3.3. In addition to the two available ones in Altair, there is a customized and heterogeneous color palette tailored for non-data elements of the chart (such as axes and grid lines, for example), for secondary encoded data (as vertical reference lines, for example), and for semantic distinction (thus complementing the other color palettes). Taking the first four monochromatic colors (Figure 3.3), White is used, for example, as the background color and as the stroke color in the marks that are used on the vertical axes of a slopegraph to mitigate visual clutter; Light Gray is used, for example, for shaded areas that highlight a part of the ROC/PR/Gain space of a ROC/PR/Gain curve chart and for the review area of a score distribution chart; Gray is used, for example, for direction markers/strips, that is, to show whether the change in the continuous value associated with a category is positive or negative according to a reference (double encoding using colors and marks); finally, Black is used, for example, as a more aesthetically pleasing color for chart guides and text, as well as for threshold markers. As for Green and Red, these are used semantically with a positive and negative character (to compare differences), as well as a proxy for non-fraud and fraud, respectively. The

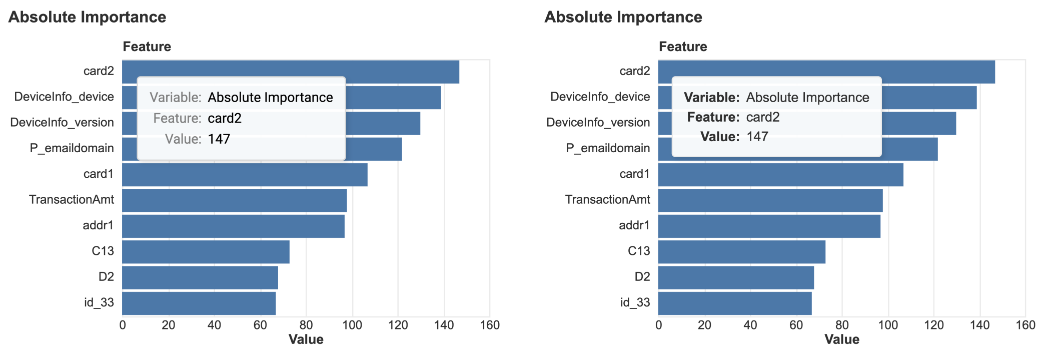
application of these color palettes will become clear throughout the Features section.

- As a final note on color, it is important to highlight that the chosen Green is a *bluish* tone of green [255]. As evidenced by the comparison of Figure 3.4 and Figure 3.5, the choice of a binary set of dichotomous colors like this needs to be done carefully so that the color palette is accessible to as many people as possible, including people with some kind of red/green color deficiency [256]. In this way, with this bluish green, it is possible to maintain the semantic meaning typically attributed to these two colors at Feedzai (green for non-fraud and red for fraud), while this color palette is more accessible than one with a *normal* tone of green.
- Based on the points that discuss color, it is also important to note that MevaL is ready to be used OOTB in a light mode interface. This decision was mainly due to the default light theme of JupyterLab, as well as a custom theme for JupyterLab created at Feedzai, also light [279].

Hereupon, the options described above, as well as other non-data elements, constitute a custom Altair theme that serves as a *skin* for the charts. In this context, an Altair theme (or simply theme [288]) is a set of top-level chart configurations [302] applied together (via a Python dictionary-based theme registry) to (globally) customize the appearance of the charts developed in Altair (within any Python session/notebook).

In parallel, MevaL also adopts the concept of a tooltip theme to refer to Cascading Style Sheets (CSS)-driven aesthetics for Altair tooltips (since these characteristics cannot be changed from the top-level chart configurations, as is the case for axes and legends, for example [300]). Although the tooltip is expressible as a visual encoding channel, that is, the content of the tooltip can be defined and customized through the usual API, its logic and style are addressed with the Vega Tooltip plugin [197] that is integrated into Vega-Embed [198], the convenience wrapper for rendering the charts used by Altair. This decoupling between tooltips and Vega/Vega-Lite specifications assembled with Altair is in line with one of Vega's design principles: Vega and Vega-Lite are designed to operate declaratively and agnostically in relation to the renderer or platform where the charts will be shown — this way, given that the tooltip is a feature that works in browsers but does not work in SVG files, for example, it is not fully integrated into Vega [194]. So, it is necessary to define a HTML/CSS snippet/string (it is necessary to rely on web-based technologies), based on a set of standard CSS properties [278], and load it separately so that the tooltip theme inherits the same colors and the same font as Altair's main theme. For this, two functions are used, *display()* and *HTML()*, from IPython [220] (*display* module), the Jupyter kernel to work with Python (JupyterLab dependency). The current disadvantage of this method is that the tooltip theme disappears when the user clears the notebook's entire output, that is, the displayed tooltip theme is transient. This method is non-invasive, in the sense of persisting changes in CSS properties of the JupyterLab interface, for example, but it should be active during the entire JupyterLab session to work robustly. In addition, considering Altair version 2.4.1, the respective style, or CSS, is not injected into a possible HTML file by default, so it needs to be monkey patched. To overcome the last problem, MevaL provides a helper function to save a certain chart as an HTML file that monkey patches the Jinja HTML template [252] used by Altair to include the necessary CSS properties, as well as to add an extra argument needed in the *save* method (Appendix P).

That said, MevaL has a custom theme (inspired by those available in theming packages [210, 311]) that is loaded when importing the package, as well as a custom tooltip theme (a comparison between the default tooltip theme and this custom one can be seen in Figure 3.2). In addition, MevaL provides some custom helper functions (Appendix H) that allow the verification of the loaded theme (and the list of available themes), changing the default width and height of charts, the loading of the default Altair theme, and the activation of another available (Altair or tooltip) theme (the user can also employ the Altair API itself to create new themes from scratch).



(a) Altair's default tooltip theme (*light* theme).

(b) MevaL's custom tooltip theme.

Figure 3.2: A comparison between Altair's default tooltip theme and the custom theme provided by MevaL. The customized theme offers improved contrast between the text and the background, in addition to ensuring that the same colors and font are used for consistency purposes.

Moreover, Altair (more specifically Vega) supports two types of (HTML-based) renderers: HTML5 Canvas and SVG. Both have advantages and disadvantages, depending on the application, but in the case of MevaL, it was decided to use the SVG renderer as the default one (instead of Canvas). Keeping the plotting context in mind, this choice was motivated by the fact that SVG is vector-based, providing better image quality (it is resolution-independent, so to speak) and *infinite* zooming, even though, in theory, Canvas provides improved rendering performance and scalability based on the number of charts [54]. Considering the average size of the notebooks created by Feedzai's data scientists in the sample analyzed and the way Jupyter notebooks are stored (they are stored in a JSON-based format, ignoring the greater complexity of the HTML Document Object Model (DOM), the representation of a web page in the browser, originated by the SVG format, for example), the benefits of Canvas are not a priority — if necessary, the renderer type can be changed with a single line of code.

Regarding the axis labels (for both the X-axis and the Y-axis) and some tooltip fields, it is noteworthy that some number/time format patterns (or *abbreviations*) are applied to ensure the most compact, human-readable, and consistent representation possible [274]. Thus, the main standard format patterns (the week starts on Monday by default) for MevaL are:

- For axes with ratio-based performance metrics, the values are formatted as percentages, with two significant digits and without the (possible) insignificant trailing zeros. The other float numbers follow the same format, but without the percentage part. To balance the lower precision for the axis labels, tooltips are formatted with four significant digits.
- Integers, such as confusion category values or certain FI values, follow decimal notation with an International System of Units (SI) prefix and without insignificant trailing zeros.
- The numbers that represent positive or negative differences, as is the case in the difference in ranking of features, are signed (a plus sign for zero or positive numbers and a minus sign for negative numbers).
- For time-based labels and tooltip fields where the month, day and year are relevant (mainly in tooltip fields), the format applied is "<full month name> <zero-padded day of the month>, <year with century>" ("November 30, 2020", for example). Relevance depends on the time resolution/discretization present in a given feature, given that MevaL currently supports discretization at the day, week, and month levels.
- For time-based labels and tooltip fields where the month and year are relevant (mainly in tooltip fields), the format applied is "<full month name> <year with century>" ("November 2020", for example).
- For time-based labels and tooltip fields where the month and day are relevant (mainly in labels), the format applied is "<full month name> <zero-padded day of the month>" ("November 30", for example).

Finally, MevaL also controls some global pandas options, especially display-related options (these options will impact the HTML representation used by IPython to show pandas DataFrames on Jupyter notebooks). These adjustments are intended to remove the metadata that is printed at the end of the pandas DataFrame (dimensions), adjust the horizontal behavior of the pandas DataFrame (no limit on columns, but they have a smaller maximum size, so that the user can view the entire pandas DataFrame, even if it is necessary to scroll horizontaly, on a *row*) and raise an exception when the user tries to change a certain value (via a chained assignment) on a copy of a slice from a pandas

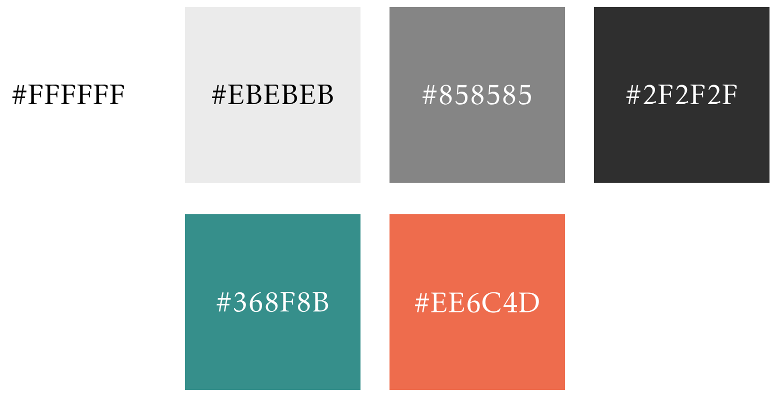


Figure 3.3: The custom color palette for MevaL (in hexadecimal format), used in conjunction with the Tableau 10 color palette [283] (Altair's default categorical color palette [200]) and the Blues one (Altair's default sequential single-hue color palette [200]).

DataFrame, thus preventing manipulations with unexpected results. The specific options and the respective values can be found in Appendix J.

3.6 Features

Throughout this section, all charts available in MevaL will be shown and described (Appendix T contains a gallery with more examples). Figure 3.6 contains an overview of all types of charts, grouped by subpackage, which MevaL currently supports. In addition, the data layer that supports MevaL will also be covered.

As general considerations about MevaL's charts (these details do not apply to the pandas-based content in the Tabular Performance Analysis section), they tend to follow/inherit from two abstract base classes (two classes that act as *starter blueprints* of the chart classes), and one of the chart class constructor arguments (or instance variables) is constant and currently only supports one value. As for the abstract base classes (Code Listing 3.1), these classes were created for two main reasons (apart from good practices in object-oriented programming (OOP) and code organization): (1) ensure a minimally consistent plotting API for the various charts with a group of standard basic arguments and methods that allow a set of actions to be carried out, such as showing the charts; (2) facilitate future integration of new visual extensions into the package from a common and easily interpretable basis for the developer. As for the argument mentioned earlier, this is the argument that defines the plotting engine (or package) to be used (*plotting_engine* in Code Listing 3.1). Currently, only Altair (*"altair"*) is supported for all charts, but in the future, other engines may be integrated into MevaL. This option is particularly interesting to allow a more natural fit of charts created with other technologies where it is possible to create certain types of charts that are not possible with Altair, for example. In addition, this approach has the advantage of maintaining the same interface that wraps the possible visualization packages that could be supported (another approach would be to have a separate module or class for each chart according to its plotting backend as it is the case

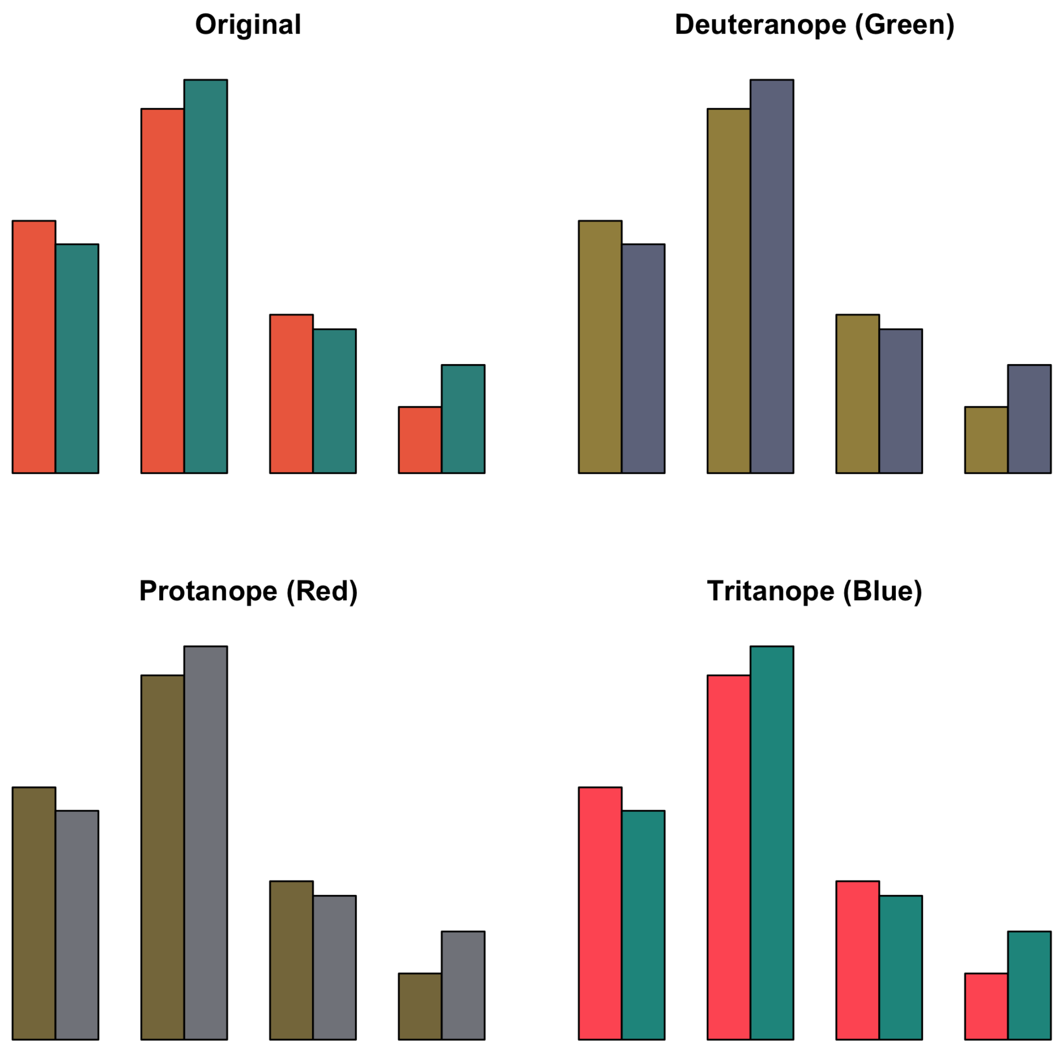


Figure 3.4: How colorblind users (approximately) see the original colors of the chart in the upper left corner. In this example, where colorweak users are not considered (they have less severe anomalies but in the same line as colorblindness [256]), deuteranope, protanope, and tritanope are different kinds of dichromats, so they can only rely on just two primary colors [327] instead of three (the color in parentheses corresponds to the non-functional color). Considering the Green and Red chosen for MevaL, it is still possible to categorically distinguish both colors, although care must be taken with the nomenclature used in terms of colors and semantic meaning. These charts were generated with the help of the colorspace R package [348] and its implementation of a physiologically-based model for simulating color vision deficiency [176] (considering the maximum possible severity, that is, dichromacy). The respective script can be found in Appendix F.

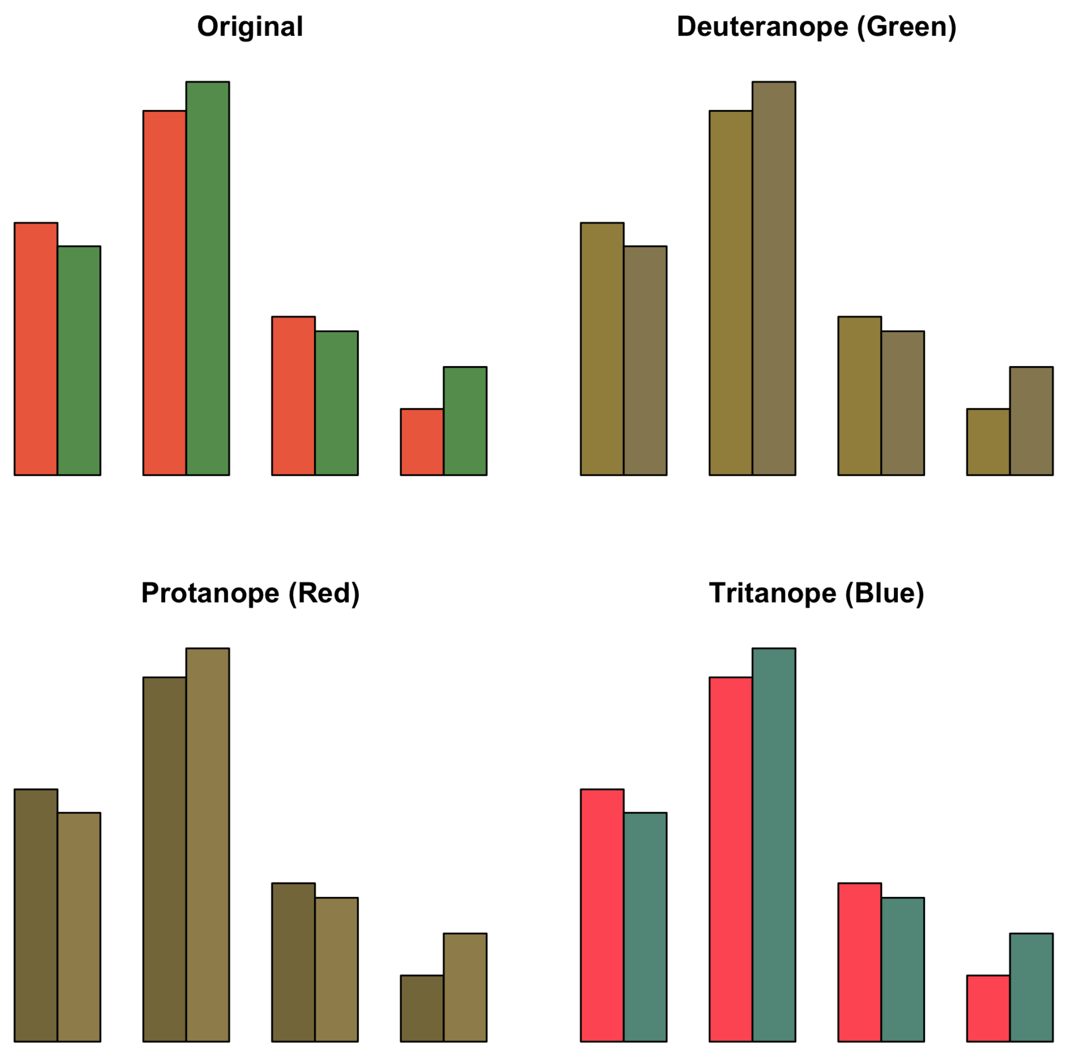


Figure 3.5: A scenario similar to that of Figure 3.4, but now with another starter tone of green. In practical terms, both colors, although slightly distinguishable, can no longer be categorically distinguished — they look like two brownish colors from a sequential color palette now (except for tritanopes). These charts were generated with the help of the colorspace R package [348] and its implementation of a physiologically-based model for simulating color vision deficiency [176] (considering the maximum possible severity, that is, dichromacy). The respective script can be found in Appendix F.

of the *plot* and *iplot* modules in the spectrum_utils package [35], for example). Since this argument is constant and for the sake of simplicity, it will be omitted hereinafter.

Listing 3.1: The two abstract base classes for the charts available in MevaL. The *encode()* instance method is the method responsible for creating a given chart from a given input (according to data, visual encodings, and other options), with or without some kind of manipulation in between (it is a method for internal use and it is expected to be used only once in the class constructor method). The *show()* instance method is responsible for simply returning the previously created chart (stored in an instance variable) to be displayed in JupyterLab (it is a method for external use and it is expected to be used more than once). In addition to the methods present in this snippet, these classes also define getters and setters with input validations. The *UAltairChart* type can be found in Appendix H.

```python
from abc import ABC, abstractmethod
import pandas as pd
from .constants import UAltairChart


class OneDimChart(ABC):
    def __init__(
        self,
        data: pd.DataFrame,
        xvar: str,
        xscale: str = "linear",
        plotting_engine: str = "altair",
    ) -> None:
        self.data = data
        self.xvar = xvar
        self.xscale = xscale
        self.plotting_engine = plotting_engine

    @abstractmethod
    def encode(self) -> UAltairChart:
        pass

    @abstractmethod
    def show(self) -> UAltairChart:
        pass


class TwoDimChart(OneDimChart):
    def __init__(
        self,
        data: pd.DataFrame,
        xvar: str,
        yvar: str,
        xscale: str = "linear",
        yscale: str = "linear",
        plotting_engine: str = "altair",
    ):
        super().__init__(
            data=data, xvar=xvar, xscale=xscale, plotting_engine=plotting_engine,
        )

        self.yvar = yvar
        self.yscale = yscale
```

Complementing the previous paragraph, there are also other general (secondary) points to clarify. All marks, such as points and bars, have associated tooltips (assuming that charts are used in their interactive versions). The tooltips allow showing (extra) details (via a kind of table) on demand (when hovering) defined based on the available data fields (the variables of the dataset, basically).

On the other hand, charts with two or more subcharts (so-called small multiples or compound charts in Altair), are arranged in a grid with a maximum of two subcharts per row (by default), that is, these charts will have an arbitrary number of rows but only two columns (in the case of a row with only one subchart, it will be aligned to the left). The only exception to this arrangement is the faceted score distribution chart, where each row refers to a bin width and the number of columns will depend on the number of breakdown values if a breakdown field is specified. This value for the number of columns was chosen based on the JupyterLab interface (which has a collapsible left sidebar) and its use on laptops (with a diagonal display size of 13.3 inches or relatively similar) so that both subcharts by row appear at the same time on the screen (however, in certain cases, it may be necessary to scroll up and down to see all the subcharts).

Regarding interactivity, only a subset of the techniques available in Altair are leveraged, since the version used (2.4.1) has significantly more limited mechanisms than the most recent version (4.1.0). In this way, the main techniques adopted are tooltips (small *pop-up windows* that provide extra information when the mouse cursor hover over a mark or a data item [344]), interactive legends (in practice, this functionality is made possible by another chart in the upper right corner that simulates a chart legend, as Altair does not support interactive legends natively in version 2.4.1), and linked selections (in particular for the FI comparison and breakdown strip charts) [264]. The last two techniques allow multiple selection, that is, it is possible to select several chart elements to highlight (these can be added/removed by clicking on them while holding the Shift key [297]). In addition to these techniques, brushing (or interval selection) is also used in the (optional) control chart located below the temporal chart for Model Decay in order to simulate a *focus + context* approach that allows exploring certain subspaces of the main chart in greater detail [201]. On the other hand, techniques such as zooming, panning, and query widgets (like sliders or dropdown selection lists, for example) are not used due to the more limited options available, compared to the most current version of Altair, and because they do not provide a user experience as intended. Although the first two options are quite

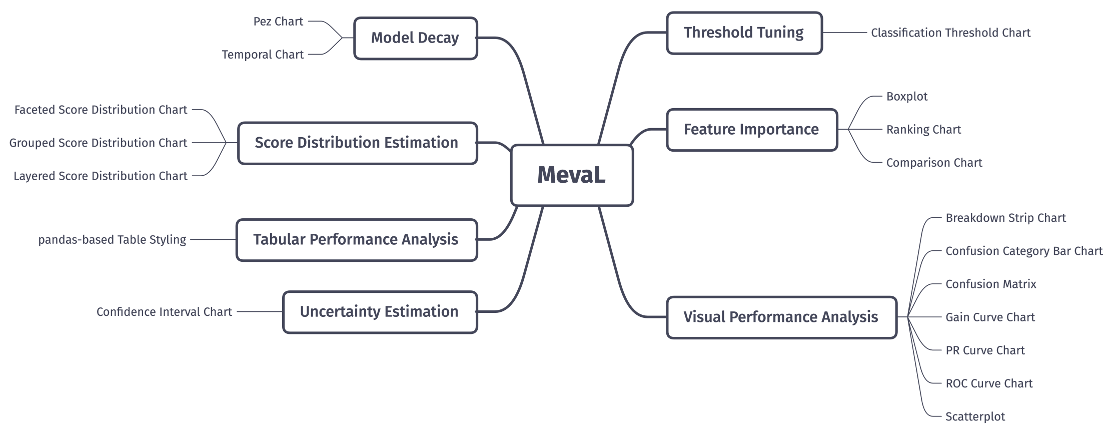

MevaL
Model Decay
Pez Chart
Temporal Chart
Score Distribution Estimation
Faceted Score Distribution Chart
Grouped Score Distribution Chart
Layered Score Distribution Chart
Tabular Performance Analysis
pandas-based Table Styling
Uncertainty Estimation
Confidence Interval Chart
Threshold Tuning
Classification Threshold Chart
Feature Importance
Boxplot
Ranking Chart
Comparison Chart
Visual Performance Analysis
Breakdown Strip Chart
Confusion Category Bar Chart
Confusion Matrix
Gain Curve Chart
PR Curve Chart
ROC Curve Chart
Scatterplot

Figure 3.6: Overview of the chart types available in each MevaL subpackage.

interesting for curve charts (in the ROC curve chart, most of the time, there is a subspace of greater interest, as is the case of the subspace for low FPR values) and the third option to have a slider that would allow the data scientist to quickly change the threshold to be considered for certain charts (such as the classification threshold chart), it is not possible to clear the selection by double-clicking (it is necessary to run the notebook cell again to return to the initial state, for example) and these query widgets (provided by Altair as input bindings) lack user-defined options to make them clearer about their purpose through textual labels, as well as API-driven styling options, for example. The techniques adopted fall into the categories of Abstract/Elaborate (tooltips), Filter (interactive legends and linked selections) and, in a way, Connect (brushing) of an established taxonomy [264, 344]. In the future, other techniques may be more easily adopted after upgrading the Altair version.

The Dataset section of the User Validation and Case Study chapter contains information on the data (including FI data) used for the visualizations in this section. In addition, the images for the charts were obtained using the procedure described in Appendix I (they faithfully represent the charts that appear as output from JupyterLab cells).

3.6.1 Model Evaluation Topology

The Model Evaluation Topology (Figure 3.7) is a visual representation of the different objectives and tasks for Model Evaluation. In a nutshell, it is a diagram that helps to organize the charts available in MevaL in different subpackages that serve distinct purposes. It is divided into the main need, that of evaluating models to obtain feedback, four objectives (Table 3.4), and ten tasks. The motivation to develop this kind of hierarchical conceptualization is twofold:

1. First, to have a way to communicate as standardized as possible about the Python package and its features, since the features will be instantiated from this topology.

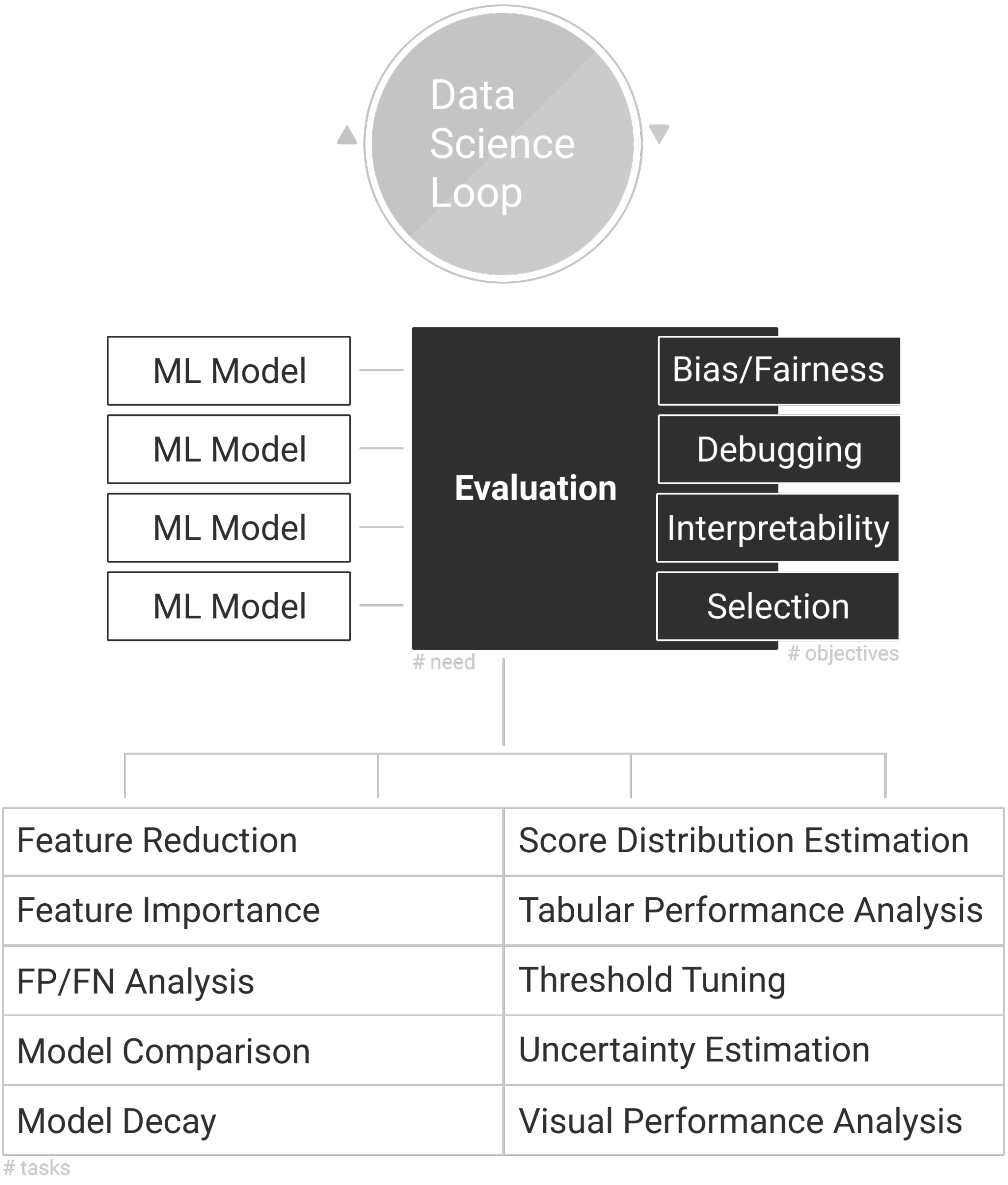


Figure 3.7: Model Evaluation Topology.

2. Second, to frame the design of the features to be implemented based on specific tasks and objectives, given that, not all features can be as flexible as a two-dimensional chart for two arbitrary performance metrics, for example.

That said, the tasks that constitute the lowest level of the Model Evaluation Topology are the following:

- **Feature Reduction**: Features or actions related to reducing the number of features being used. The reduction in the number of features is done via Feature Selection

Table 3.4: Model Evaluation objectives.

Objective	Possible Question
Bias/Fairness	How can I confirm that one of my subgroups of interest is not being harmed in relation to the rest?
Debugging	How can I properly estimate the performance of my model?
Interpretability	How can I find out what are the most important features for my model?
Selection	How can I choose between these two models with similar performance?

and not via Dimensionality Reduction, that is, a subset of the original features is used. An alternative name could be Feature Subsetting.

- **FI**: Features or actions related to computing the feature importance. They don't provide actionable information by themselves, so they appear combined with FP/FN Analysis and Feature Reduction, for example.
- **FP/FN Analysis**: Features or actions related to assessing regions of the feature space where the model is failing.
- **Model Comparison**: Features or actions related to the comparative diagnosis and selection of models based on performance and other relevant metrics. They are related to Performance Analysis, and the models can be obtained from different hypothesis classes and/or hyperparameter sweeps.
- **Model Decay**: Features or actions related to offline inspecting performance metrics over time. They are related to Model Comparison and the idea of measuring things in different time windows. An alternative name could be Model Degradation.
- **Score Distribution Estimation**: Features or actions related to checking and comparing the prediction score distribution for various datasets and/or models. It is also relevant for models in production, where new "ground truth" labels may or may not be available.
- **Tabular Performance Analysis**: Features or actions related to checking and comparing business and performance metrics in a tabular or numeric format.
- **Threshold Tuning**: Features or actions related to adjusting the classification threshold. This impacts threshold-dependent metrics.
- **Uncertainty Estimation**: Features or actions related to complementing point estimates with information on variability and statistical significance.
- **Visual Performance Analysis**: Features or actions related to checking and comparing business and performance metrics with visual support through charts.

That said, all tasks are covered by charts and other features from MevaL except FP/FN Analysis. This type of feature-based, model-driven analysis implies crossing the original data with the results obtained from a model in order to try to identify patterns in the incorrectly classified instances or how they vary compared to the well-classified ones, for example (leading to the generation of new ideas for features or other options that may allow to improve the previous results). So, this exercise would imply considering the original datasets as one of the datasets plottable by MevaL and not just as the source of raw data to be potentially processed in smaller formats and with aggregated information. In addition, given the large size of these original datasets and the scope of the package for analyzing performance metrics, prediction scores, and FI values (processed datasets away from the feature space), this inclusion would lead to a significant increase in complexity and necessary computational resources. Thus, the FP/FN Analysis was considered as future work. Lastly, it is also important to note that although it is totally possible to compare different models, MevaL is, at the moment, a single model-first package (more information in the Scope section) and although the charts available for the FI task help to select subsets of features manually, MevaL does not support any pre-modeling feature selection method or direct mechanism that allows selecting a subset of features and training a new model, for example. MevaL is a package focused on Model Evaluation *per se*, that is, on facilitating the analysis of the score-based results obtained from the development of ML models. In this way, it is a package that works from raw/processed data and does not communicate with Pulse, that is, it does not interact programmatically with the tool that allows training new models.

With the tasks presented in mind, it would also be possible to consider Subgroup Discovery. This task includes features or actions related to the inspection of different breakdowns (based on distinct fraud concepts and/or feature values). However, given that, based on the user interviews, the subgroups are usually known *a priori*, it will only be considered the associated granularity (subgroup) for other tasks, as is the case of Tabular Performance Analysis.

Regarding the Bias/Fairness objective, it was not addressed simply because there is a whole team at Feedzai working on issues related to this vast topic. Thus, it was decided not to include particular features with this objective in mind.

3.6.2 Data and Data Processing

First of all, MevaL is designed to operate on (one or multiple raw) domain-specific tabular datasets (called *main datasets* from now on to facilitate the distinction between this type of dataset and the FI ones) consisting of a set of input (raw or engineered) features/variables, a (binary) target column, and a prediction score column (from the raw output of a given classification ML model) for each row (transaction in most cases). Within the set of features, there is a subset of these with semantic value for the problem in question, such as the feature for the transaction amount (a weighing feature), the feature for the timestamp, and different categorical breakdown features that *divide* the dataset into relevant subgroups. In addition, the package also digests (small) datasets of FI (absolute or relative/percentage) values (each row refers to a different feature), obtained from the internal workings of an algorithm or agnostically, to help users extract insights that can help them understand the data, the model, and how to improve the model [42] (this dataset is optional). The data scientist can analyze one or two of these datasets as it is possible to compare the set of FI values (and ranking positions) obtained for the same set of features and for the same *unit of measurement*. The single processed dataset is similar to these raw datasets, if they are long-format/tidy [316] datasets, with a few extra columns to help include additional information in the relevant charts. On the other hand, if the raw datasets are wide-format/wider [318] datasets, where each feature will be in a different column or the different types of FI values (absolute and relative/percentage, for example) are in different columns, instead of one column for the type and one for the value, for example, MevaL will transform them into a long-format dataset as well (basically, making a dataset longer leads to an increase in the number of rows and a decrease in the number of columns [318]). For now, MevaL only accepts files in comma-separated values (CSV) format.

Given the typically large size of these raw domain-specific datasets and the lack of summary statistics due to their finer granularity, MevaL also offers a data processing pipeline (on top of PySpark and pandas) to derive two sufficiently small datasets (called *functional datasets* when they are referred to together) to serve as a basis for the visual part: (1) a wide-format [318] (approximately) dataset dedicated to performance metrics (performance metrics and confusion categories are unstacked to facilitate their plotting and mapping between visual encodings and concrete column names/accessors, as well as their comparison using a table, all in a single representation); (2) a long-format/tidy [316] (approximately) one dedicated to prediction scores aggregated in different ways. One of the main differences between both datasets is that, in the first case, scores are interpreted as possible classification thresholds and are (merely) used to obtain certain performance metrics, while in the second case, scores are actually interpreted as predicted scores and there are interest in understanding its distribution.

Running this pipeline is optional (for both types of source files or for just one of them, and, in the case of the main dataset, it is possible to process it only to derive one of the two functional datasets) since MevaL accepts datasets loaded (via pandas) and previously processed by MevaL or other tools, as long as each follows a relatively soft predefined schema (or input format). Figure 3.8 and Figure 3.9 briefly outline MevaL's data-oriented input and output. However, given the efficiency of the data processing pipeline and to keep the API (and the logic behind it) as simple as possible, the data scientist needs, for each model/raw dataset, to specify a processed dataset for each one of them in each of the optional fields for datasets already processed (this mechanism serves primarily to fully load previous work quickly). In other words, it is not possible to combine paths for files and datasets (DataFrames) that have already been processed arbitrarily, since the processing is triggered based on the paths for specified datasets and if the field for each functional dataset is empty or not (the data scientist must pass a processed dataset for each model/raw dataset, or pass none at all and process everything). Thus, if the data scientist has two datasets that he/she wants to analyze, but only one of them is already processed, he/she must leave the fields for the processed datasets empty and pass the paths to the respective files (note that both these fields work separately to trigger the data processing pipeline or not, as well as the one for the FI dataset). In addition, except for FI datasets which are totally optional, the data scientist must specify at least one path to a main raw dataset, one path to a main raw dataset and one of the respective functional datasets, or both functional datasets (these fields cannot all be empty).

The flexibility present in this pipeline is mainly due to two reasons:

1. First, this flexibility combines with the possibility of persisting the (small) processed datasets, so that they can be used by different people, at different times, without the need to process the data again (even if this pipeline is relatively efficient, this will not be as fast as loading this data into memory).
2. Secondly, this flexibility makes it easy to use other tools that allow to achieve the same final product as the pipeline implemented in MevaL, something that can be useful in contexts where the available tools may be different.

That said, given that there are three possible output datasets, each will have its own schema or basic set of columns. The first one, for the performance metrics dataset, is based on the following (boilerplate) schema:

- model (identifier)
- breakdown field
- breakdown value
- classification threshold
- TP
- FP
- TN
- FN
- lower endpoint of the 100(1 - α)% confidence interval
- performance metric
- upper endpoint of the 100(1 - α)% confidence interval

The confusion category columns (TP, FP, TN, and FN) are optional, as they can be removed from the final version of this (performance metrics) dataset using a Boolean argument. In addition, if the data scientist specifies a cost field (as the transaction amount field, for example) and wants to compute the supported cost-based (weighted) versions

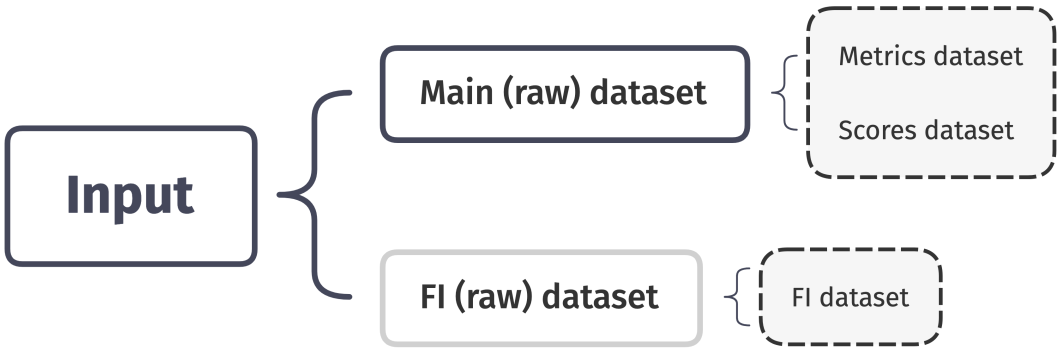


Figure 3.8: Data input to MevaL. MevaL accepts two types of source files: the main dataset (with the train/test data and the predicted scores) and the FI dataset (optional). MevaL supports multiple main datasets, merging them into unique functional datasets that can be queried with this dimension in mind as well, and one or two FI datasets paired with a specific model. Being raw files, these datasets will trigger the data processing pipeline and give rise to the functional datasets in Figure 3.9. However, it is also possible to replace these datasets with ones previously processed in different combinations, and the processing will only occur partially (if the data scientist has a pre-processed metrics dataset, the pipeline will only process the main dataset in order to generate a new scores dataset, for example). MevaL distinguishes between raw and processed datasets based on the type of object used as an argument (ignoring the fact that multiple datasets are specified through a sequence-based list or tuple): if the object is a string-based Python dictionary (the path to the file corresponds to the key and the file name corresponds to the value), it means that the data scientist has a dataset that he/she wants to process; if the object is a pandas DataFrame, it means that the data scientist has a dataset already processed (in addition, if the object is a PySpark DataFrame, it will be converted to a pandas DataFrame).

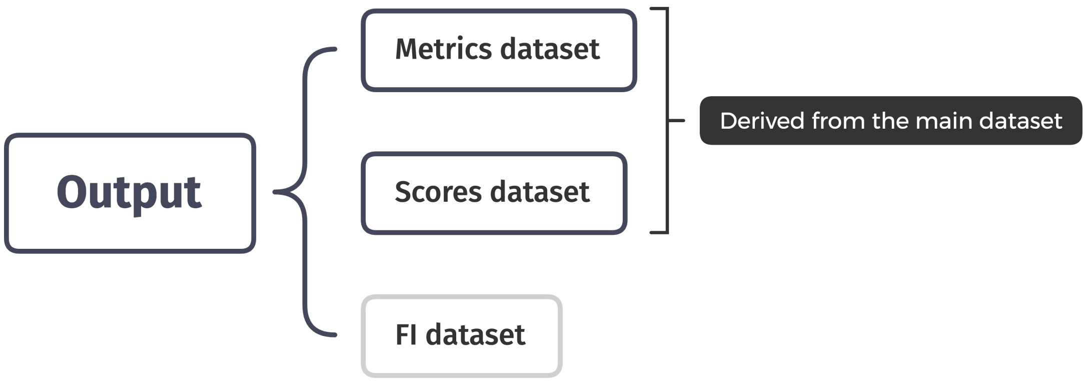


Figure 3.9: Data output from MevaL. Based on the input in Figure 3.8, MevaL will give access to two or three single-purpose, multi-task functional datasets that can be visualized and analyzed with the help of the package.

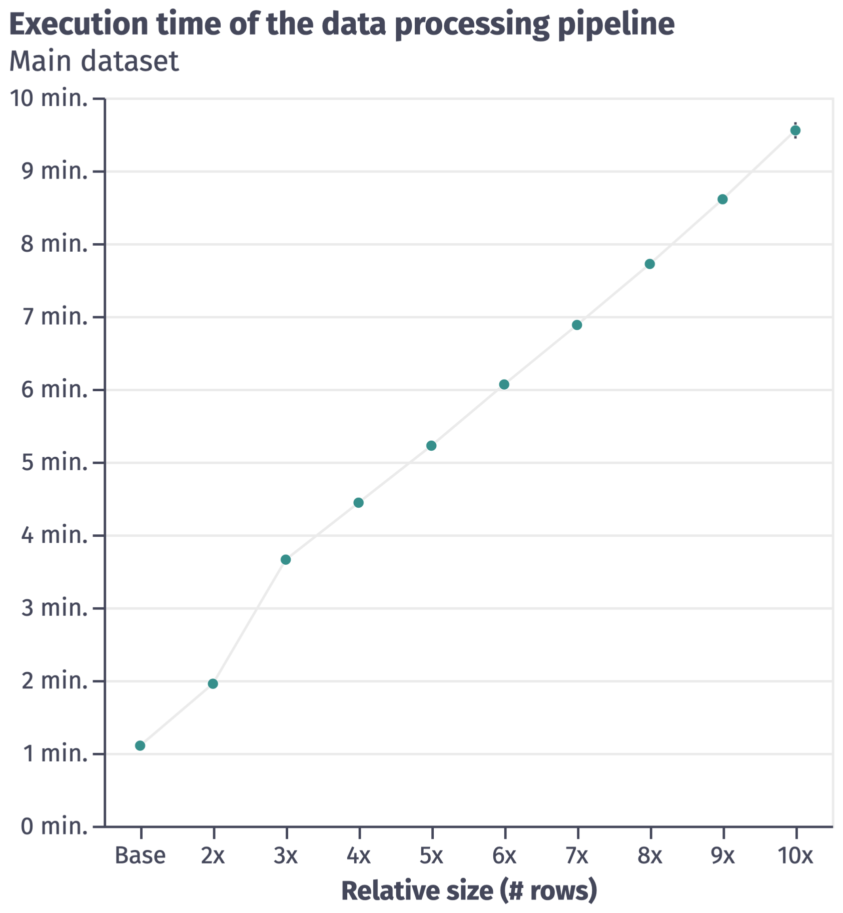


Figure 3.10: Execution time of the data processing pipeline for the main dataset considering different variations of the (base) main dataset obtained by sampling with replacement (between approximately 120,000 rows and 1,200,000 rows). The points/circles correspond to the average values obtained considering 20 independent repeats. The vertical error bars correspond to the associated standard deviations, being only visible in the last case and negligible in all of them. The relative size is always based on the dataset described in the Dataset section and Appendix K. For the base main dataset, the data processing pipeline takes about 1 minute (with a memory usage of approximately 1.2 GB), while for a similar dataset with 10 times more rows, it takes between 9 and 10 minutes (with a memory usage of approximately 2 GB). These execution times (with an approximately linear trend) are sufficiently efficient and workable for a development context. More information on this performance analysis can be found in Appendix L.

of the performance metrics of interest, there will also be confusion category and performance metric columns for them. The confidence interval-related columns will also only be added if the data scientist wants to compute this complement to the point estimate obtained for each of the supported performance metrics of interest. For now, this feature is only available for (count-based) FPR, Precision, and Recall. That said, the performance metrics currently available natively (their definitions can be found in the Data Science and Machine Learning section) on MevaL are (these metrics were selected based on the ones currently used by Feedzai's data scientists or because they are appropriate for imbalanced datasets):

- (Count/Cost-based) Alerts
- (Count-based) Alert Rate
- (Count-based) F_1 Score, $F_{0.5}$ Score, and F_2 Score
- (Count/Cost-based) FPR
- (Count-based) Geometric Mean
- (Count-based) MCC
- (Count/Cost-based) Precision
- (Count/Cost-based) Recall

On the other hand, the rows whose value is *overall* for the breakdown field and breakdown value columns correspond to the overall performance of the model including all instances of the original dataset. More generally, the *overall* keyword always refers to the complete (original) dataset in which all instances are included in the calculations (this value also corresponds to the minimum representation of each dataset derived from the main one since the addition of breakdown fields/values is optional). As for the second output dataset, the aggregate scores dataset, it is based on the following (boilerplate) schema:

- model (identifier)
- breakdown field
- breakdown value (distribution)
- prediction score (original or truncated)
- count/frequency
- percentage

Prediction scores are, by default, in the range between 0 and 1 and are maintained in this range after processing. Each breakdown value, for each breakdown field, is treated as a distribution, that is, the sum of the percentages of each score in each breakdown value adds up to 100%. In addition, since scores can have a large number of decimal places (more than 10, for example), there is an option to truncate them, that is, to keep only a certain number of decimal places without any rounding. In this way, the number of unique values is significantly reduced (producing a reduced aggregate scores dataset, particularly important with the addition of breakdown fields and values), without sacrificing the usually desired precision (2 to 4 decimal places). Finally, the last dataset, the FI dataset, it is based on the following (boilerplate) schema:

- feature name
- FI variable name ("Absolute Importance" and "Relative Importance", for example)
- FI value
- FI method identifier
- rank
- difference in ranking compared to the other FI method
- difference in FI value compared to the other FI method
- *not_percentage*
- *box*
- *absolute_change*

The last five columns are only computed/required if there are two results of FI methods to be compared. The ranking is currently calculated using the ordinal ranking method [336] (the *first* value for the *method* parameter in pandas [180]), that is, each feature is assigned a different value. This method is used to ensure that there is a different ranking position for each feature, even if the FI value is the same (the respective axes must behave as categorical-oriented axes). The difference-based columns are only computed if the data scientist specifies two datasets with the results of two different methods/runs of the same method. On the other hand, the last three columns are helper columns specifically for the slopegraph (or comparison chart) that can be plotted to compare two FI methods (more information can be found in the Feature Importance section). The *not_percentage* column is a Boolean column to indicate whether FI values should be interpreted as percentages or not on the Altair side. The *box* column contains Unicode box-drawing characters [326] to encode groups of features (features with the same FI value) on the Y-axis of the slopegraph. Lastly, the *absolute_change* column corresponds to the annotations to be added in the slopegraph containing a summary for the features that have a difference in ranking higher than a specific threshold. Note that the last two columns are only used in the version of this chart designed for a reduced number of features (or for a chart whose height is large enough and does not imply that the entire chart is visible on the screen at the same time).

From a high-level point of view, the (full) data processing pipeline for main datasets is described by the following steps (the respective pseudocode can be found in Algorithm 1, Algorithm 2, Algorithm 3, and Algorithm 4):

1. Standardize/Enrich some of the input arguments (if necessary). This step is responsible for the following actions: Algorithm 2, Line 2
    a) Ensure that the threshold(s) to analyze, performance metric(s), and breakdown(s) are in a sequence (list or tuple, preferably).
    b) Standardize the name of performance metrics.
    c) Concatenate the paths and filenames for each score file/main raw dataset (in order to create path strings).
    d) Split breakdowns into two objects, one for the fields and one for the user-defined values. This division is intended to facilitate the use of each part individually in certain functions.
    e) Define the threshold listing strategy as *"scores"* if the scores are not truncated, thus considering only the scores in the raw dataset as threshold steps.
2. Create a *SparkSession* (the *entry point* for using PySpark) or obtain an existing one. A2, L3
3. Prepare two empty lists that will serve as containers for each type of processed dataset (performance metrics dataset and aggregate scores dataset). A2, L4-5
4. For each score file: A2, L6-17
    a) Read the score file using PySpark. A2, L7
    b) Minify the dataset (PySpark DataFrame) by selecting only a subset of columns, that is, transform the dataset into a minified version with just the columns of interest (based on the semantic and breakdown fields). The columns for the predicted scores and the cost/amount (if specified) are cast to double data type. Predicted scores may also be truncated. A2, L8
    c) If the timestamp field is one of the specified breakdown fields, add temporal columns extracted from the timestamp field (according to the supported and specified options/breakdown values) with these new resolutions in order to facilitate future manipulations of the minified dataset. Basically, in this way, the discretized temporal columns (for day, week, and/or month) can be operated as breakdown columns. In addition, also add the breakdown values corresponding to the discretizations applied to the list of breakdown fields (and remove the timestamp field itself). A2, L9-12
    d) For the overall results, group the predicted scores and aggregate them by computing the number of actual (count-based and cost-based, if specified) positives and negatives (according to the target field). This step will result in a relatively small pandas DataFrame (an example of this DataFrame can be seen in Table 3.5), with a column for each unique score associated with a column for the number of actual positives, a column for the number of actual negatives and,

if specified, a column for the number of actual positives weighed by the associated cost/amount and, finally, a column for the number of actual negatives weighed by the associated cost/amount. When grouping the scores, it is possible to sum each of the associated binary integer values obtained according to the equality between the target field values and the adopted (weighted or not) target/non-target value. This dataset serves to compute the final performance metrics dataset, as well as the aggregate scores dataset ready to be plotted (scores work as scores and also classification thresholds). Moreover, this function is fundamental for the efficiency of the pipeline, since, in this way, it is not necessary to go through the entire dataset repeatedly to compute these summary statistics for each classification threshold, for example — a sufficiently (memory-wise) small representation is generated and it is possible to rely on pandas-based (column-wise) vectorized operations to easily compute the final datasets (more precisely, a dataset of this type is also generated for each breakdown field and value, and in the end, all these datasets are concatenated into one). $\boxed{\text{A4, L2}}$

e) For the overall results, prepare the aggregate scores dataset by computing the total instances for each score as well as the respective percentages. $\boxed{\text{A4, L3}}$

f) Before computing the (count-based and cost-based, if specified) performance metrics of interest, for the overall results, extend the list of available scores/thresholds, sort them in descending order (more precisely the pandas DataFrame), and compute the confusion categories. The thresholds to be considered are extended with a possible set of thresholds specified by the user (if they are not already in the dataset) and/or with the missing values in an evenly spaced range of values between 0 and 1 coming from a power of 10 (calculated according to the number of decimal places after truncating the scores). On the other hand, by ordering the dataset before calculating the confusion categories, scores closer to 1 will be at the top, while scores closer to 0 will be at the bottom, which allows these values to be computed in an efficient (for pandas) and cumulative way. For example, the TPs for the 0.99 score, the first score of a hypothetical dataset, are all the (actual) positive values summed up previously, that is, the positive values in the respective column of this row of the dataset. As for the 0.98 score, the second score of a hypothetical dataset, the (actual) positive values correspond to the values of its row, as well as those of the previous row, that is, the number of positives in the score 0.99, since the true positives are all instances whose scores are equal to or higher than a certain classification threshold (and so on). This logic is similar for FPs and the (actual) negative column. In this way, it is possible to calculate two of the confusion categories (TP and FP) by cumulatively adding the values of the previously computed positive and negative columns, and the other two confusion categories (FN and TN) can be obtained by subtracting the total number of positives and the total number of negatives to the number of TPs and the number of FPs, respectively. $\boxed{\text{A4, L4}}$

g) For the overall results, compute the performance metrics of interest (and confidence intervals if requested). If confidence intervals are also calculated, each trio of columns for one of the supported performance metrics will be reordered so that the column for the point estimate is in the middle between the lower endpoint and the upper one. $\boxed{\text{A4, L4}}$

h) For each breakdown field and breakdown value (each breakdown value represents a subset of data that should be considered as a *full dataset*): $\boxed{\text{A4, L5-28}}$

   i. Repeat the steps described between **4. d)** and **4. g)** but now considering that the original dataset is also grouped by one breakdown field at a time (at this moment, the breakdown values are still in this column and not in a separate one), and the aggregate scores dataset and the relevant performance metrics will be computed for each relevant breakdown value. A column for breakdown fields and another for breakdown values are also added to help disambiguate the results. $\boxed{\text{A4, L7-18}}$

   ii. Concatenate all generated (partial) datasets into one and fill in the null values with the *overall* keyword. Note that it is each type of dataset that is merged into one, that is, there are an intermediate scores dataset and another one for performance metrics. $\boxed{\text{A4, L19-28}}$

i) Add a column for the model identifier (currently, the identifier corresponds to the file name without the extension). $\boxed{\text{A2, L14}}$

j) Reorder the columns for each type of dataset so that the leftmost columns are the model identifier column, the breakdown field column (a kind of identifier too), and the breakdown value column(and the threshold column for the performance metrics dataset as well). $\boxed{\text{A2, L14}}$

k) Add each type of dataset, for a particular model, to its outer list (**3.**). $\boxed{\text{A2, L15-16}}$

5. Concatenate all per-model datasets, and return the final aggregate scores dataset and the final performance metrics dataset ready to be used. $\boxed{\text{A2, L18}}$

For cases where only a part of the above pipeline is needed, it is *simplified* and only the relevant steps will be executed to generate the aggregate scores dataset or the performance metrics dataset (more specifically, steps unique to one or the other dataset will not be performed in the simplified functions). In contrast, the data processing pipeline for FI datasets consists of the following approximate steps (the respective pseudocode can be found in Algorithm 5):

Table 3.5: The first five rows of an example main dataset after the aggregation and sorting step.

fraud_score	P	N
1.00	0	0
0.99	88	2
0.98	127	16
0.97	100	4
0.96	94	3

Algorithm 1 Input and output of the data processing pipeline for main datasets

```
Input:
    score_files – The unstructured paths to the main raw dataset(s)
    target_field – The name of the target column
    target_value – The positive target value
    non_target_value – The negative target value
    score_field – The name of the predicted score column
    timestamp_field – The name of the time column
    breakdowns – The name(s) of the breakdown column(s), as well as the optional value(s)
    to be considered for each specified breakdown column
    thresholds_to_analyze – Set of extra classification thresholds to consider
    performance_metrics – The performance metrics of interest to be calculated
    compute_cost_metrics – Whether to calculate the cost-based counterparts of the confu-
    sion categories and performance metrics of interest
    compute_ci_for_metrics – Whether to calculate the binomial proportion confidence
    interval via normal approximation for each supported performance metric
    keep_confusion_categories – Whether confusion categories should be maintained in the
    performance metrics dataset
    sep – Delimiter to use for score_files
    timestamp_fmt – The format string to represent the timestamp_field
    cost_field – The name of the cost/amount column
    truncate_score_decimals – The number of decimal places to consider for the discretiza-
    tion of the predicted scores
    threshold_listing_strategy – The method for structuring the classification thresholds to
    be considered for computing the performance metrics dataset
Output:
    Performance metrics dataset
    Aggregate scores dataset
```

Algorithm 2 Data processing pipeline for main datasets

```
1: function PROCESS_SCORES(Input @ Algorithm 1)
2:     normalized_args ← NORMALIZE_INPUT_ARGUMENTS(score_files, performance_metrics, thresholds_to_analyze, breakdowns, truncate_score_decimals, threshold_listing_strategy)
3:     spark ← GET_OR_CREATE_SPARK_SESSION()
4:     per_model_metrics ← empty list
5:     per_model_scores ← empty list
6:     for all input_path ∈ normalized_args.score_files do
7:         raw_scores_df ← READ_SPARK_DF(input_path, sep)
8:         minified_scores_df ← MINIFY_RAW_SCORES(raw_scores_df, target_field, score_field, timestamp_field, cost_field, normalized_args.breakdown_fields, truncate_score_decimals)
9:         if timestamp_field ∈ normalized_args.breakdown_fields then
10:            minified_scores_df ← ADD_TIME_COLUMNS(minified_scores_df, timestamp_field, timestamp_fmt, normalized_args.breakdown_values)
11:            normalized_args ← UPDATE_TIME_BREAKDOWN_FIELDS(normalized_args, timestamp_field, normalized_args.breakdown_fields, normalized_args.breakdown_values)
12:        end if
13:        metrics_df, scores_df ← PREPARE_METRICS_AND_SCORES(minified_scores_df, score_field, target_field, target_value, non_target_value, normalized_args.performance_metrics, compute_cost_metrics, compute_ci_for_metrics, keep_confusion_categories, cost_field, truncate_score_decimals, normalized_args.threshold_listing_strategy, normalized_args.thresholds_to_analyze, normalized_args.breakdown_fields, normalized_args.breakdown_values)
14:        metrics_df, scores_df ← ADD_MODEL_ID_AND_REORDER_COLS(metrics_df, scores_df, input_path)
15:        per_model_metrics.APPEND(metrics_df)
16:        per_model_scores.APPEND(scores_df)
17:    end for
18:    return CONCAT_DFS(per_model_metrics), CONCAT_DFS(per_model_scores)
19: end function
```

1. Standardize/Enrich some of the input arguments (if necessary). This step is responsible for the following actions: $\boxed{\text{Algorithm 5, Line 2}}$

   a) Ensure that the feature and FI column names are in a sequence (list or tuple, preferably).

   b) Concatenate the paths and filenames for each FI file/FI raw dataset (in order to create path strings).

   c) Obtain the unique name for the feature name (*feature_col_name*, FI variable name (*variable_col_name*, and FI value (*value_col_name* columns according to pre-established constants for cases in which the dataset needs to be transformed into a long-format one (basically when there is more than one feature name and/or FI variable column) or according to the names for these columns (except for the FI value column, called *value* by default, that does not need to

Algorithm 3 Input and output of the wrapper function for intermediate steps in the preparation of each partial functional dataset

Input:

minified_scores_df – A PySpark DataFrame with only the columns of interest
score_field – The name of the predicted score column
target_field – The name of the target column
target_value – The positive target value
non_target_value – The negative target value
performance_metrics – The performance metrics of interest to be calculated
compute_cost_metrics – Whether to calculate the cost-based counterparts of the confusion categories and performance metrics of interest
compute_ci_for_metrics – Whether to calculate the binomial proportion confidence interval via normal approximation for each supported performance metric
keep_confusion_categories – Whether confusion categories should be maintained in the performance metrics dataset
cost_field – The name of the cost/amount column
truncate_score_decimals – The number of decimal places to consider for the discretization of the predicted scores
threshold_listing_strategy – The method for structuring the classification thresholds to be considered for computing the performance metrics dataset
thresholds_to_analyze – Set of extra classification thresholds to consider
breakdowns_fields – The name(s) of the breakdown column(s)
breakdowns_values – The optional value(s) to be considered for each specified breakdown column

Output:

Partial performance metrics dataset for a ML model
Partial aggregate scores dataset for a ML model

be defined by the data scientist).

2. Create a *SparkSession* (the *entry point* for using PySpark) or obtain an existing one. $\boxed{\text{A5, L3}}$
3. Create a blank dataset (pandas DataFrame) for the final FI dataset (all processed files are appended to the same final pandas DataFrame). $\boxed{\text{A5, L4}}$
4. For each FI file (maximum of two files per model): $\boxed{\text{A5, L5-16}}$
   a) Read the file into a temporary/partial pandas DataFrame, cast the necessary fields, and melt it if necessary (unpivot it from wide to long format). If it is not necessary to melt it, the FI column is divided/unfolded into two columns: one for the name of the FI variable (*variable_col_name*), according to the original name of the column, and another one (*value_col_name*) for the FI values themselves. This step is done for the sake of consistency (there are cases in which there may be more than one type of results for the same FI algorithm/run, as is the case for algorithms/runs that produce absolute and relative values, for example). $\boxed{\text{A5, L6-12}}$

Algorithm 4 Wrapper function for intermediate steps in the preparation of each partial functional dataset

```
 1: function PREPARE_METRICS_AND_SCORES(Input @ Algorithm 3)
 2:     aggregated_scores_df ← AGG_SCORES_FOR_BASE_METRICS(minified_scores_df, target_field, target_value, non_target_value, score_field, compute_cost_metrics, cost_field)   ▷ Overall
 3:     scores_df ← PREPARE_SCORES(aggregated_scores_df, score_field)   ▷ Partial aggregate scores dataset
 4:     metrics_df ← COMPUTE_METRICS_AND_CI(aggregated_scores_df, score_field, performance_metrics, compute_cost_metrics, compute_ci_for_metrics, keep_confusion_categories, truncate_score_decimals,threshold_listing_strategy, thresholds_to_analyze)   ▷ Partial performance metrics dataset
 5:     metrics_bfield_dfs ← empty list
 6:     scores_bfield_dfs ← empty list
 7:     for all $bfield \in breakdown\_fields$ do
 8:         bvalues_to_compute ← GET_BREAKDOWN_VALUES(minified_scores_df, bfield, breakdown_values)▷ Get the list of unique values for a breakdown field if there are no user-defined values
 9:         metrics_bvalue_dfs ← empty list
10:         scores_bvalue_dfs ← empty list
11:         for all $bvalue \in bvalues\_to\_compute$ do
12:             scores_bvalue_df ← PREPARE_SCORES(aggregated_scores_df, score_field, bfield, bvalue)
13:             scores_bvalue_df ← ADD_BFIELD_AND_BVALUE_STANDALONE_COLS(scores_bvalue_df, bfield, bvalue)   ▷ Identity-like columns whose values are the name of the breakdown field and the breakdown value, respectively
14:             metrics_bvalue_df ← COMPUTE_METRICS_AND_CI(aggregated_scores_df, score_field, performance_metrics, compute_cost_metrics, compute_ci_for_metrics, keep_confusion_categories, truncate_score_decimals,threshold_listing_strategy, thresholds_to_analyze, bfield, bvalue)
15:             metrics_bvalue_df ← ADD_BFIELD_AND_BVALUE_STANDALONE_COLS(metrics_bvalue_df, bfield, bvalue)   ▷ The original breakdown field column is also removed
16:             metrics_bvalue_dfs.APPEND(metrics_bvalue_df)
17:             scores_bvalue_dfs.APPEND(scores_bvalue_df)
18:         end for
19:         metrics_bfield_df ← CONCAT_DFS(metrics_bvalue_dfs)
20:         scores_bfield_df ← CONCAT_DFS(scores_bvalue_dfs)
21:         metrics_bfield_dfs.APPEND(metrics_bfield_df)
22:         scores_bfield_dfs.APPEND(scores_bfield_df)
23:     end for
24:     metrics_df ← CONCAT_DFS(metrics_df, metrics_bfield_dfs)
25:     scores_df ← CONCAT_DFS(scores_df, scores_bfield_dfs)
26:     metrics_df ← FILL_NULL_VALUES(metrics_df, "Overall")
27:     scores_df ← FILL_NULL_VALUES(scores_df, "Overall")
28:     return metrics_df, scores_df
29: end function
```

   b) Create the ranking field and sort the pandas DataFrame (ascending). The *first* argument is used to ensure that there is a different ranking position for each feature, even if the feature importance value is the same. If null values exist, they will be located at the end. $\boxed{\text{A5, L13}}$

   c) Create the field for the FI method identifier (composed of the path and the name of the respective file). $\boxed{\text{A5, L14}}$

   d) Add the partial DataFrame to the final DataFrame (**3.**). $\boxed{\text{A5, L15}}$

5. Create a helper Boolean field (*not_percentage*) to define, in Altair, what formatting is indicated for the FI values (axes and tooltip), that is, if the formatting pattern of these values should contain the % type value in order to be multiplied by 100 and complemented with the percent sign (FI values can be integer values or percentage values). The values that will be considered as percentages will be the real values between 0 and 1 (inclusive), while the values that will not be considered as percentages will be, above all, the integer numbers (this is verified holistically to disambiguate possible small values within a range of values where the upper limit is greater than 1). $\boxed{\text{A5, L17}}$

6. If two FI files are specified, do the following: $\boxed{\text{A5, L18-23}}$

   a) Create the field for the difference in ranking compared to the other FI method. $\boxed{\text{A5, L19}}$

   b) Create the field for the difference in FI value compared to the other FI method. $\boxed{\text{A5, L20}}$

   c) Compute the last two helper fields for the slopegraph that can be plotted to compare two FI methods. The *box* field contains Unicode box-drawing characters [326] to encode groups of features (features with the same FI value) on the Y-axis of one of the versions of the slopegraph. On the other hand, the *absolute_change* field corresponds to the annotations to be added in the same slopegraph containing a summary for the features that have a difference in ranking higher than a specific user-defined threshold. An example of one of these annotations is *card1: 4.37% → 4.63% (+ 0.27%)*. The annotations are placed on the right side of the respective chart, so the change is structured from left to right (from the first method to the second method according to the order of the methods in the argument passed by the data scientist). $\boxed{\text{A5, L21-22}}$

7. Return the final pandas DataFrame (FI dataset). $\boxed{\text{A5, L24}}$

For now, this data processing pipeline assumes that both possible FI datasets contain the same set of features. A many-to-many FI method/result comparison may require different runs of this pipeline (via the *EvaluationManager* class presented in the Entry Point section) in order to generate different datasets with 2 of the methods/results to be

Algorithm 5 Data processing pipeline for FI datasets

Input:
fi_files – The unstructured paths to the FI raw dataset(s)
feature_cols – The name of the feature name column(s)
fi_cols – The name of the FI results column(s)
sep – Delimiter to use for *fi_files*
Output:
FI dataset

```
1: function PROCESS_FI(Input)
2:     normalized_args ← NORMALIZE_INPUT_ARGUMENTS(fi_files, feature_cols, fi_cols)
3:     spark ← GET_OR_CREATE_SPARK_SESSION()
4:     fi_df ← empty pandas DataFrame
5:     for all input_path ∈ normalized_args.fi_files do
6:         raw_fi_df ← READ_PANDAS_DF(input_path, sep)
7:         raw_fi_df ← CAST_NUMERIC_COLS()
8:         if LEN(normalized_args.feature_cols) > 1 or LEN(normalized_args.fi_cols) > 1 then
9:             raw_fi_df ← WIDE_TO_LONG_FORMAT(raw_fi_df, normalized_args.feature_cols, normalized_args.fi_cols, normalized_args.feature_col_name, normalized_args.variable_col_name, normalized_args.value_col_name) ▷ Features and/or FI variables in different columns move to different rows (single column)
10:        else
11:            raw_fi_df ← UNFOLD_FI_COLUMN(raw_fi_df, fi_cols, normalized_args.variable_col_name, normalized_args.value_col_name)
12:        end if
13:        raw_fi_df ← COMPUTE_ORDINAL_RANKING(raw_fi_df, normalized_args.variable_col_name, normalized_args.value_col_name)
14:        raw_fi_df ← ADD_FI_ID(raw_fi_df, input_path)
15:        fi_df ← CONCAT_DFS(fi_df, raw_fi_df)
16:    end for
17:    fi_df ← TAG_NON_PERCENTAGE_COL(fi_df, normalized_args.variable_col_name, normalized_args.value_col_name)
18:    if LEN(normalized_args.fi_files) = 2 then
19:        fi_df ← COMPUTE_RANKING_DIFF(fi_df, normalized_args.feature_col_name, normalized_args.variable_col_name, normalized_args.value_col_name) ▷ Differences are propagated, so the reference can be any of the methods
20:        fi_df ← COMPUTE_VALUE_DIFF(fi_df, normalized_args.feature_col_name, normalized_args.variable_col_name, normalized_args.value_col_name) ▷ Differences are propagated, so the reference can be any of the methods
21:        fi_df ← ADD_BOX_DRAWING_CHARS_COL(fi_df, normalized_args.variable_col_name, normalized_args.value_col_name) ▷ Per FI identifier
22:        fi_df ← ADD_CHANGE_ANNOTATIONS_COL(fi_df, normalized_args.feature_col_name, normalized_args.value_col_name)
23:    end if
24:    return fi_df
25: end function
```

compared (it is more flexible if the data scientist only wants to see, individually, the FI ranking by method/execution).

3.6.3 Entry Point

Now that the expected input and output datasets have been presented, as well as the implemented data processing pipeline, it is relevant to address the *entry point* of this package, that is, the *EvaluationManager* class that serves the purpose of aggregating all the metadata/variables/functional datasets for a Model Evaluation procedure, as well as triggering the data processing pipeline (if necessary). Although this class acts as the starting point for using MevaL, is possible to use the plotting subpackages without instantiating this class, for example (for the sake of flexibility). That said, the *EvaluationManager* has the following mandatory/optional attributes (instance variables/class constructor parameters/*__init__()* method parameters):

1. ***target_field***: The name of the target column ("ground truth" labels).
2. ***target_value***: The positive target value (the fraud label usually).
3. ***score_field***: The name of the predicted (positive) score column.
4. ***timestamp_field***: The name of the time (timestamp/datetime) column (composed of UNIX/POSIX timestamps or datetime-like values).
5. ***performance_metrics***: The performance metrics of interest to be calculated. The performance metrics are validated and standardized/sanitized according to a Python dictionary (similar to the sanitize_ml_labels package [46]).
6. ***score_files***: A dictionary whose keys identify the path and values the main raw dataset filename(s). The default value is *None* (null value).
7. ***fi_files***: A dictionary whose keys identify the path and values the raw FI dataset filename(s). The default value is *None*.
8. ***metrics_dfs***: The processed/schema-compliant performance metrics dataset(s) (preferably pandas DataFrames) ready to be plotted. PySpark DataFrames are converted to pandas DataFrames. The default value is *None*.
9. ***agg_scores_dfs***: The processed/schema-compliant aggregate scores dataset(s) (preferably pandas DataFrames) ready to be plotted. PySpark DataFrames are converted to pandas DataFrames. The default value is *None*.
10. ***fi_dfs***: The processed/schema-compliant FI dataset(s) (preferably pandas DataFrames) ready to be plotted. PySpark DataFrames are converted to pandas DataFrames. The default value is *None*.
11. ***breakdowns***: The name(s) of the breakdown column(s) according to the main raw dataset, as well as the value(s) to be considered for each specified breakdown column(s) (if no value is specified via a Python dictionary, all distinct values will be

considered by default, except the null value). If one of the breakdown columns coincides with the ***timestamp_field***, the supported (discretization/resolution) values are *"day"*, *"week"*, and *"month"*. This sequence should be a list or a tuple, preferably.

12. ***non_target_value***: The negative target value (the non-fraud label usually). If no value is specified, the target value will serve as the reference (or key) to find the corresponding non-target value within a pre-established dictionary. The default value is *None*.
13. ***cost_field***: The name of the cost/amount column. The default value is *None*.
14. ***truncate_score_decimals***: The number of decimal places to consider for the discretization of the predicted scores. If no integer value is specified, the predicted scores will be grouped according to each available single value. This parameter, together with the ***threshold_listing_strategy*** parameter, influences the number of classification threshold steps considered. If the value specified for this parameter is 2 (all scores will be truncated and will only have 2 significant decimal places at most), the number of threshold steps will be 100 (evenly spaced) if the ***threshold_listing_strategy*** is *"range"* or equal to the number of unique score values after truncation if the ***threshold_listing_strategy*** is *"scores"*, for example. The default value is *2*.
15. ***compute_cost_metrics***: Whether to calculate the cost-based counterparts of the confusion categories and performance metrics of interest. The default value is *False*.
16. ***compute_ci_for_metrics***: Whether to calculate the binomial proportion confidence interval via normal approximation for each supported and specified performance metric (FPR, Precision, and Recall). The default value is *False*.
17. ***keep_confusion_categories***: Whether confusion categories should be maintained in the final performance metrics dataset or not. The default value is *True*.
18. ***timestamp_fmt***: The format code-based string to represent the ***timestamp_field*** (e.g., *"%Y-%m-%d %H:%M:%S"*). This parameter also accepts two custom strings, *"ms"* and *"s"*, used to define the format of UNIX/POSIX timestamp values in milliseconds and seconds, respectively. The default value is *"ms"*.
19. ***threshold_listing_strategy***: The classification threshold listing strategy, that is, the method for structuring the classification thresholds to be considered for computing the performance metrics dataset. If *"scores"*, only the unique scores (after truncation, if applied) present in the dataset will be considered. If *"range"*, all thresholds in a uniform/evenly spaced interval between 0 and 1 (inclusive) with 10 raised to ***truncate_score_decimals*** values will be considered. In addition, if the scores are not truncated (***truncate_score_decimals*** is set to *None*), MevaL will be forced to only consider the scores in the main raw dataset as threshold steps (*"scores"* strategy). The default value is *"range"*.

20. ***thresholds_to_analyze***: Set of specific classification thresholds to consider for analysis (these may or may not already be in the list of generated classification thresholds).

21. ***feature_cols***: The name of the feature name column(s). This field is used exclusively to process the FI dataset(s).

22. ***fi_cols***: The name of the FI value column(s). This field is used exclusively to process the FI dataset(s).

23. ***score_files_sep***: Delimiter to use for ***score_files***. This separator is applied to all files (there was no need to make this parameter more flexible, as only some projects need a different field separator for their CSV files, something followed as a kind of convention within a project). The default value is ",".

24. ***fi_files_sep***: Delimiter to use for ***fi_files***. This separator is applied to all files (there was no need to make this parameter more flexible, as only some projects need a different field separator for their CSV files, something followed as a kind of convention within a project). The default value is ",".

The type hints [254] for each of the above parameters can be found in Appendix N. In addition, the *EvaluationManager* class also has instance methods for orchestrating the computation of functional datasets, getters and setters with input validations, and, also, an instance method that prints a summary of the functional datasets. This simple summary includes the identifier for each dataset (the name of the respective property), the number of rows, the number of columns, and the total memory usage.

Regarding the *breakdowns* parameter, one of its possible arguments (or one of the parts of its argument) has particular characteristics. If one of the defined breakdown fields coincides with the *timestamp_field*, it is necessary to include which temporal discretization values should be associated with this field, that is, how the timestamp/datetime values must be truncated to be interpreted as breakdown values similar to the others. This step is important to ensure a viable resolution, both in practical/technical terms (the number of unique values is significantly reduced, for example) and in terms of granularity of interest to Feedzai's data scientists. Thus, MevaL supports three types of discretization: *day*, *week*, and *month*. In practice, with these discretizations, datetimes will be generated where only the day and/or month and year will be significant (and new columns will be added with the name of each of the selected discretizations). This mechanism is implemented with a Spark User-Defined Function (UDF) where two methods from Python's *datetime.datetime* class are used that originate *naive* objects in terms of timezone information, which makes these methods time-wise agnostic in relation to the machine(s) they will run on: *utcfromtimestamp()* (for timestamps in milliseconds and seconds) and *strptime()*. The Delorean [345] and datetime_truncate [14] packages also offer forms of truncation similar to that implemented in MevaL.

3.6.4 Feature Importance

In a nutshell, the concept of FI summarizes a set of techniques (or, sometimes, the values themselves), intrinsic or extrinsic to the models, which assign a kind of score to each of the features according to their utility for performing the proposed task. In this context, in addition to providing compressed and global information on the behavior of a model [192], these methods are particularly useful for Feature Reduction, that is, to find the minimum subset of features required to develop a valid ML model. Therefore, to help the data scientist analyze the results of (at least) one (or two at the same time) Feature Importance methods, MevaL provides three types of custom charts: the ranking chart, the comparison chart, and the boxplot.

3.6.4.1 FI ranking chart

The FI ranking chart (Figure 3.11) is a horizontal bar chart that aims to facilitate getting the big picture of the distribution of FI values, as well as comparing features individually or in subgroups obtained from their position in the ranking. The choice of a horizontal bar chart instead of vertical was due to its approximation to a table/list and its reading from top to bottom (or vice versa), typical of a ranking, as well as the possibility of arranging the axis labels for the features in an easily readable and natural way (with a zero rotation angle and without truncating the text too much) [11]. Also, this chart is ordered in descending order of FI values. That said, the parameters of the respective class are:

1. ***data***: The FI dataset (pandas DataFrame) to be plotted.
2. ***fi_id***: The identifier of the FI method/results to be plotted.
3. ***xvar***: The column name with the values for the X-axis (FI values).
4. ***yvar***: The column name with the values for the Y-axis (feature names).
5. ***catvar***: The column name with the existing FI variables, that is, the value/category types (absolute and/or percentage/relative, for example).
6. ***category***: The name of the category or type of FI values to be plotted. A different chart is generated for each of the specified categories.
7. ***mode***: The chart mode, that is, the type of chart to display. The supported values are *"bar"* (horizontal bar chart) and *"lollipop"* (horizontal lollipop chart). The default value is *"bar"* (the most common design for all data scientists at Feedzai).
8. ***top***: The number of top features to be plotted. This parameter can be used in conjunction with the *bottom* parameter to generate a chart with a combination of top and bottom features (if the *bottom* parameter is 0, only a number of top features are shown). The default value is 0 (all features are considered if *bottom* is also 0).

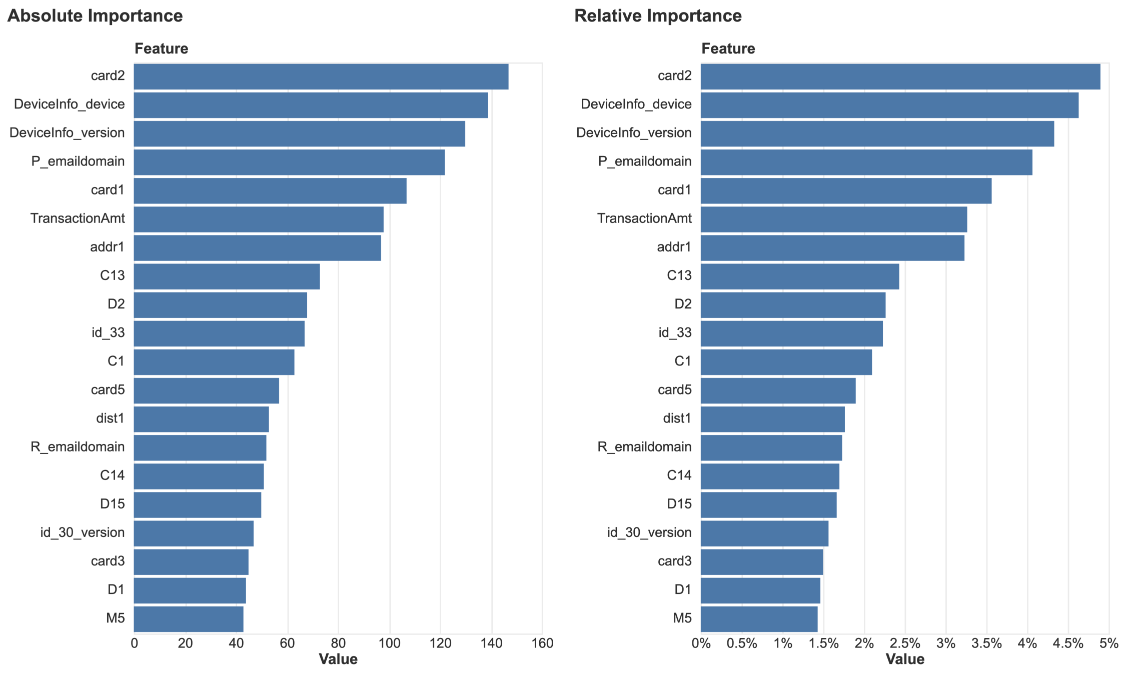


Figure 3.11: FI ranking chart. This example shows the top 20 features for the two types of available FI values.

9. ***bottom***: The number of bottom features to be plotted. This parameter can be used in conjunction with the *top* parameter to generate a chart with a combination of top and bottom features (if the *top* parameter is 0, only a number of top features are shown). The default value is 0 (all features are considered if *top* is also 0).

The lollipop *mode* (Figure T.1) is designed for cases where there are a large number of features to display (for one or more FI categories) and several bars with similar heights/FI values, and these have a small width so that the height of the chart is relatively reduced (avoiding, or at least mitigating, the need to scroll up and down to inspect the chart, for example) [53, 81, 110, 114, 205]. Thus, this version seeks to avoid the Moiré (optical) effect and make the chart visually less aggressive [81, 110, 114]. This chart works similarly to the bar chart, combining a line anchored on the Y-axis and a dot at the end to indicate the value to retain, although it may be more difficult to retrieve this value since it is the center of the dot that effectively marks it (this design is less accurate than the straight edge of a bar and half of the dot surpasses the specific value it represents) [81, 110]. However, according to one study (a crowd-sourced experiment using Amazon Mechanical Turk with approximately 150 participants), there were no significant differences between bar and lollipop charts (in terms of accuracy and response time) [113].

3.6.4.2 FI comparison chart

The FI comparison chart (Figure 3.12) is a mixed chart that aims to provide an overview of the difference in results (or lack thereof) between two FI methods/runs. This chart can be used to complement or replace the typical manual work of comparing FI results through spreadsheets. That said, the parameters of the respective class are:

1. ***data***: The FI dataset (pandas DataFrame) to be plotted.
2. ***versus***: A sequence (list or tuple) with the identifiers of each of the FI results to be compared. The order of this sequence matters, since the first identifier is considered as the reference.
3. ***xvar***: The column name with the values for the main X-axis (FI identifiers).
4. ***yvar***: The column name with the values for the Y-axis (FI values).
5. ***fvar***: The column name with the feature names.
6. ***catvar***: The column name with the existing FI variables, that is, the value/category types (absolute and/or percentage/relative, for example).
7. ***category***: The name of the category or type of FI values to be plotted. A different chart is generated for each of the specified categories.
8. ***rankvar***: The column name with the ranking positions.
9. ***rank_threshold***: The ranking difference threshold to include features on the chart. Features whose absolute difference in ranking is greater than or equal to the integer value passed as an argument are shown. This parameter can be used in conjunction with the *value_threshold* parameter, in order to create a predicate that includes one condition *and* the other. A value between 3 and 5 is a good starting value for this parameter (according to the input and experience of Feedzai's data scientists). A higher value may be symptomatic of a problem or that the results are from two different algorithms, such as LightGBM and a DL algorithm, for example. The default value is 3.
10. ***value_threshold***: The FI value difference threshold to include features on the chart. Features whose absolute difference in value is greater than or equal to the integer/real value passed as an argument are shown. This parameter can be used in conjunction with the *rank_threshold* parameter, in order to create a predicate that includes one condition *and* the other. The default value is *None*.
11. ***show_feature_names***: Whether to show the name of the features, instead of the ranking positions, on the leftmost vertical axis of the version of this chart tailored to a reduced number of features. The default value is *False*.
12. ***show_second_chart***: Whether to show the complementary (rightmost) chart to the slopegraph, the main chart to compare the overall results in terms of change of ranking. The default value is *True*.
13. ***label_char_limit***: The maximum length of labels for feature names (in pixels). If the length of a feature name is greater than this value, the text is truncated and the ellipsis string ("...") is used in response to this limit. The default value is 80 (pixels).

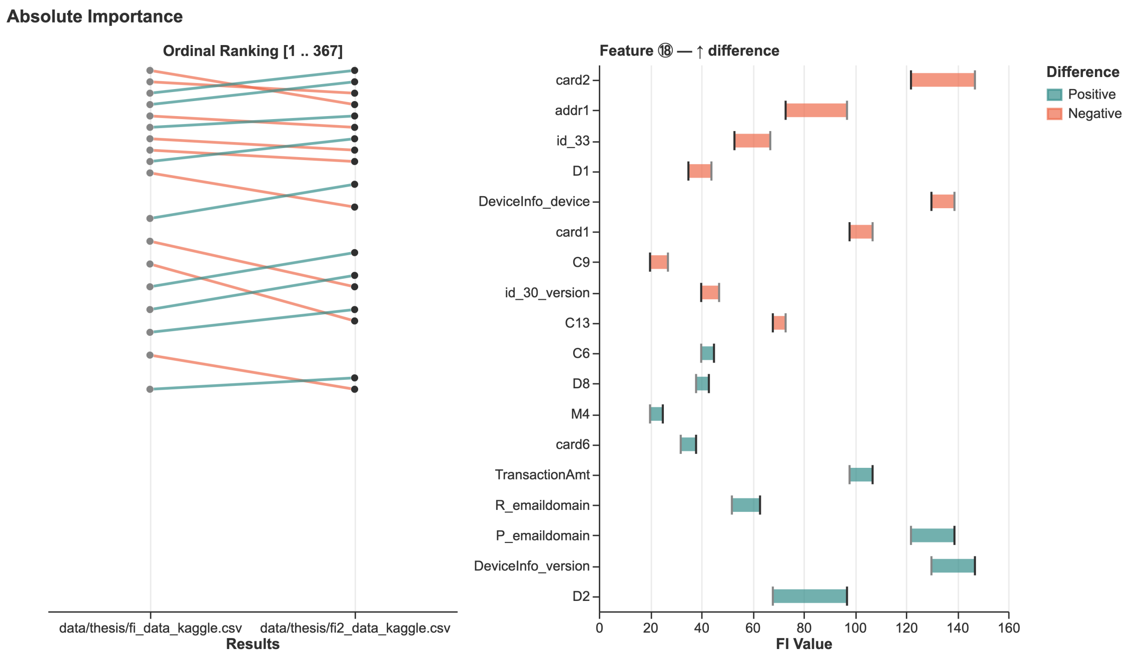


Figure 3.12: FI comparison chart. This example shows the differences between two FI (absolute) results, with special attention to the 18 features whose ranking difference is greater than or equal to 1 and whose difference in value is greater than or equal to 5. Features are sorted in ascending order of difference in (signed) value. Although there are changes in the features closer to the bottom, these are not highlighted because, in practice, the value change is zero or very close to that.

14. ***right_yaxis_sort***: The sorting criteria for the features of the chart complementary to the slopegraph. They can be ordered by absolute difference in value (*"absolute_value_diff"*), alphabetical order (*"alphabetical"*), difference in value (*"value_diff"*), according to the values of the first result (*"first_result"*), and according to the values of the second result (*"second_result"*). The default value is *"value_diff"*.
15. ***right_yaxis_order***: Based on the sorting criteria (*right_yaxis_sort* parameter), sort in ascending (*"asc"*) or descending (*"desc"*) order. The default value is *"asc"*.
16. ***mode***: The chart mode, that is, the type of chart to display. The supported values are *"slope&bar"* (a combination of a slopegraph and a horizontal grouped bar chart for a reduced number of features), *"slope&strip"* (a combination of a slopegraph and a Cleveland dot-based ranged strip chart [37, 62]), and *"dot"* (an adaptation of the Cleveland dot/ranged/arrow chart [37, 62, 63]). The default value is *"slope&strip"*.

In the standard version of this chart (*slope&strip*), color plays a central role in making the chart easier to interpret (Figure 3.12). For both subcharts, two (binary) categorical color schemes are applied: (1) the color scheme consisting of MevaL's gray and MevaL's black colors (double-)encodes the FI results, with the former referring to the first result (the reference) and the latter to the second one; (2) the color scheme composed of MevaL's green and MevaL's red colors (double-)encodes, respectively, positive (ranking increases or higher FI values) and negative (ranking decreases or lower FI values) differences, with the first result specified as an argument serving as a reference. Subchart-wise, the first subchart (the slopegraph on the left) is dedicated to the ranking for each feature explicitly displayed. Thus, in each of the vertical axes, there is a dot that indicates the position in the ranking of each of the features (the first place in the ranking corresponds to the topmost position on the Y-axis), and the change from one result to the other is hinted by the connecting lines between the same features (and their colors). So, this subchart allows at least two readings: (1) one more general, from each of the vertical axes, where one looks at the ranking order of the explicit features and the surfaced gaps/clusters; (2) another more granular, considering the features individually, when inspecting the connecting slopes and which ones went up or down (or reveal no changes). Even if, for a given criterion, there are no significant changes or there are few changes for a relaxed one, this finding can also be important for the data scientist [138]. As for the chart on the right (the ranged strip chart) and its apparatus, the Y-axis title contains some supplementary information, such as the number of features in accordance with the specified threshold(s) (such as Unicode circled numbers [330, 331], obtained via Python's *unicodedata* built-in module, or simple numbers, if the number of features is greater than 50) and the ordering criteria for them. Since this chart is a ranged/gap chart [62], the ticks (instead of dots) encode the values obtained in each FI result per feature (with the color distinguishing the results), with a colored band that connects them, in case there is any difference in value, to facilitate visual inspection of the differential magnitude. This design pays more attention to the distance between the values than to the values themselves [62], although the data scientist can consult the tooltips associated with each tick to verify the specific values in question (Figure T.14). The arrow version (*dot*) of this comparison chart [63] works in a similar way to this subchart, using colored arrows to indicate the sign/direction of the change (Figure T.3).

3.6.4.3 FI boxplot

The FI boxplot (Figure 3.13) aims to show the approximate distribution of the FI values, based on five summary statistics (five-number summary), as well as to enable the comparison of this distribution between the results available in a given dataset. This boxplot is a boxplot with whiskers from minimum to maximum and without outliers encoded as individual points (the idea is to show a general picture of the distribution of values in which there is not a clear outlier concept). That said, the parameters of the respective class are:

1. ***data***: The FI dataset (pandas DataFrame) to be plotted.
2. ***xvar***: The column name with the values for the X-axis (FI identifiers).
3. ***yvar***: The column name with the values for Y-axis (FI values).

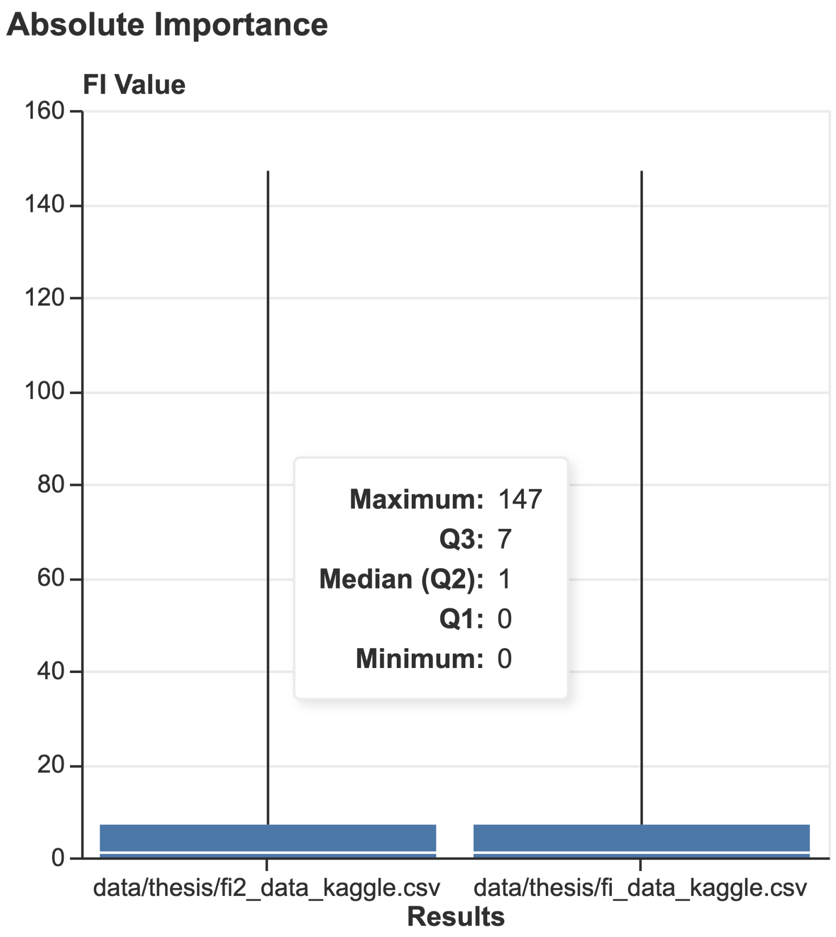


Figure 3.13: FI boxplot. In this example, it is possible to see that the values are concentrated in a relatively low range of values. The value of the third quartile (the median of the upper half of this dataset) is just 7 (the maximum value is significantly higher).

4. ***catvar***: The column name with the existing FI variables, that is, the value/category types (absolute and/or percentage/relative, for example).
5. ***category***: The name of the category or type of FI values to be plotted.

3.6.5 Model Decay

3.6.5.1 Temporal chart

The temporal chart (Figure 3.14) is an interactive line chart that provides a visual way of checking how performance, according to a given metric, varies over time. In other words, this chart shows several points for which performance has been estimated, instead of considering only a single value that covers the entire time span covered by a given dataset. In this way, this chart seeks to show the decay of the performance of a model over time and whether this decay is abrupt after a certain moment or if the model is somewhat conservative and performance is slowly decreasing, for example. This chart seeks to help the data scientist to select a model, as a given model may perform better in the first moments of a test dataset, but with a significantly quick drop in performance after that, while another, with a worse performance than the previous one, but which maintains a similar performance for a longer time (depending on the lifetime of each model in production, it may be more appropriate to choose a more conservative model, for example). In addition, it can also be useful to identify periods when performance has fluctuated significantly and encourage future analysis. That said, the parameters of the respective class are:

1. ***data***: The performance metrics dataset (pandas DataFrame) to be plotted.
2. ***xvar***: The name of the time discretization (*"day"*, *"week"*, or *"month"*) for the X-axis.
3. ***yvar***: The name of the performance metric column(s) to be plotted. A different chart is generated for each of the specified performance metrics.
4. ***at_value***: The classification threshold of interest (if the *reference* argument is *"threshold"*) or the reference value for the *reference* argument performance metric for which the classification threshold should be obtained. In the case of the second behavior, the classification threshold is obtained depending on the value of a given reference performance metric (if the exact value does not exist in the dataset, the closest smaller value is used). This option is important for the data scientist because, in many cases, they are guided by a certain reference value (or upper bound) for a given performance metric and do not directly use the classification threshold.
5. ***mode***: The chart mode, that is, the type of chart to display. The supported values are *"line"* (interactive line chart) and *"pez"* (a simple pez chart, a kind of one-dimensional temporal heatmap [310]). The default value is *"line"*.
6. ***show_points***: Whether to show the points for which there are estimates or just the line. The default value is *False*.
7. ***show_zoom_line_chart***: Whether to show the interactive sparkline/control timeline [201] coordinated with the main line chart that serves to select specific time periods and display them in more detail. The default value is *True*.
8. ***show_pez***: Whether to show the pez chart as a complementary chart at the bottom. The scale domain of this chart is between 0 and 1, showing a perspective based on the entire domain supported by each ratio-based performance metric, while the Y-axis of the line chart is adapted according to the concrete range in the dataset. The default value is *False*.
9. ***ref_line***: The specific value of a performance metric to be highlighted in the line chart (in particular for single charts), through a horizontal line, or highlight the general performance (for each metric), considering the entire dataset, if the value is the string *"overall"*. The default value is *None* (no horizontal line).

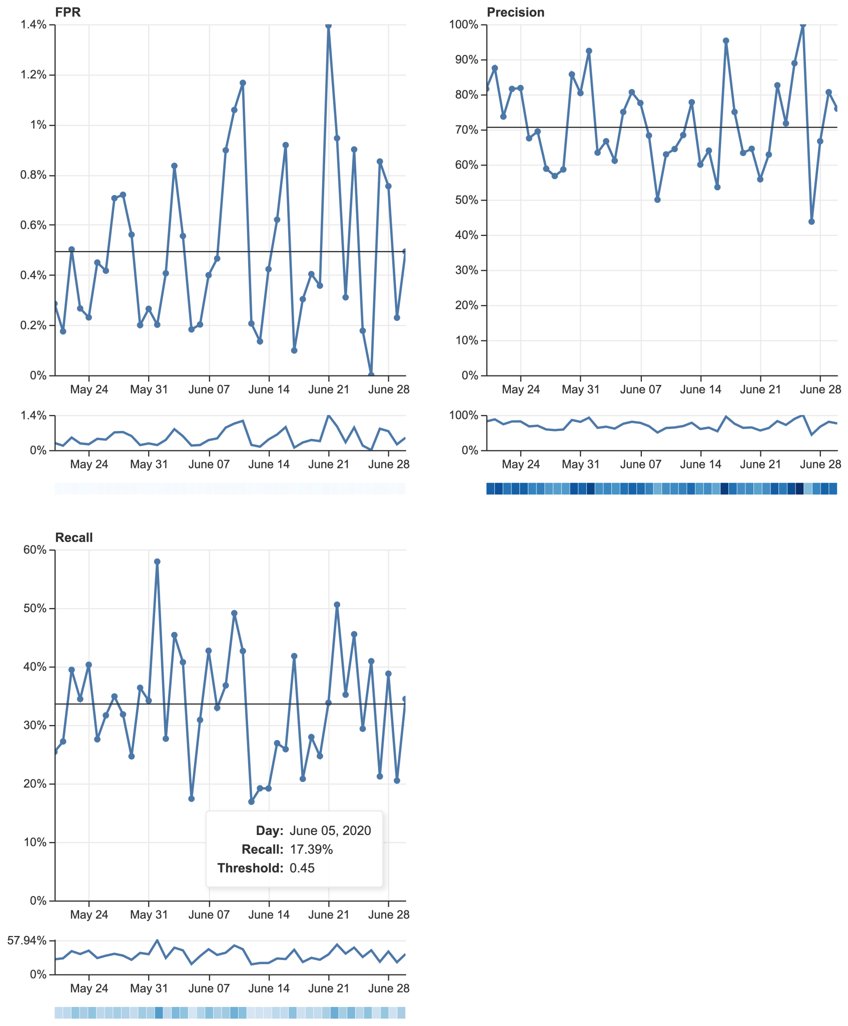


Figure 3.14: Temporal chart for Model Decay assessment.

10. ***reference***: The performance metric to be used as a reference to obtain the classification threshold for which to select the values to be plotted or simply *"threshold"* in order to specify a classification threshold using the *at_value* argument. The default value is *"threshold"*.

3.6.6 Score Distribution Estimation

3.6.6.1 Score distribution chart

The score distribution chart (Figure 3.15) is a histogram [213] for visual inspection of the approximate distribution of the predicted scores by a ML model, globally or for each value in a breakdown field. At Feedzai, this type of chart is mainly used to cover the following subtasks (in addition to its use for communication purposes):

- Qualitatively check the expected skewness of the fraud (positive) and non-fraud (negative) distributions.

- Detect anomalies.
- Define alert and automatic decline thresholds.
- Inspect the fraud/non-fraud "ratio" in the various bins.

That said, the parameters of the respective class are:

1. ***data***: The aggregate scores dataset (pandas DataFrame) to be plotted.
2. ***xvar***: The column name with the values for the X-axis (the predicted scores, or, more precisely, each single predicted score, truncated or not).
3. ***yvar***: The column name with the values for the Y-axis (the count/frequency or percentage of each score).
4. ***breakdown_field***: The breakdown field for which to plot each of the approximate distributions of the respective (breakdown) values (ignoring the overall distribution). The default value is *None* (just show the overall distribution).
5. ***breakdown_values***: The subset of values for each of the specified breakdown fields, in order to limit the number of distributions to be shown. The default value is *None* (consider all breakdown values).
6. ***yscale***: The scale type of the Y-axis. The supported values are *"linear"* (linear scale) and *"log"* (logarithmic scale). In the case of the *"log"* scale, a logarithmic transformation is applied to the original values (the values of the input domain) for the Y-axis. This type of scale can be useful when the data vary over several orders of magnitude, allowing *zoom in* on the various bins, even if they are very small [264]. The default value is *"linear"*.
7. ***show_overall***: Whether to show the overall histogram when a breakdown field is specified. The default value is *True* (this parameter is ignored if no breakdown field is specified).
8. ***bin_width***: The width of each bin. It is possible to define more than one value in a sequence (list or tuple), and a chart (or set of chart, if a breakdown field is specified) is generated for each value. Bin widths can be specified in the range [0, 1]. The default value is *0.1*.
9. ***threshold_marker***: The value, considering the continuous scale of the X-axis, to be highlighted. If more than one threshold marker is specified, numeric labels will only appear, in addition to the vertical line, if the difference in the thresholds is greater than or equal to 0.1 (so as to avoid visual clutter). Threshold markers can be specified in the range [0, 1]. The default value is *None* (no threshold markers).
10. ***threshold_marker_dashed***: Whether to make the threshold marker line dashed instead of solid. The default value is *False*.

11. ***show_threshold_area***: Whether to show the areas divided by two threshold markers (without the numeric labels) and the percentage of instances in each of them, as well as a grouped bar under each histogram to show how the percentage of instances is divided ignoring the width of the areas (in the score space). Each of the three areas/segments is colored with the respective color for non-fraud (negative class), uncertain/to be reviewed, and fraud (positive class), respectively. These threshold-based areas are useful for defining alert and automatic decline thresholds, for example. The default value is *False*.
12. ***mode***: The type of histogram for the score distribution by breakdown value, that is, when a breakdown field is specified (it is assumed that there will be two or more distributions to be plotted). There are three different supported types: *"facet"* (a faceted histogram, that is, a histogram for each different distribution), *"group"* (a grouped histogram, like a grouped bar chart, where the continuous X-axis scale is broken so that the bars of each distribution are side by side for each bin), and *"layer"* (a layered/overlaid/overlapping histogram). Usually, this chart helps to compare distributions considering fraudulent (positive) and non-fraudulent (negative) instances (according to the true labels) separately. The layered and grouped versions are optimized for two breakdown values, as is the case when choosing the (binary) target field as the breakdown field, for example. The default value is *"layer"* (the most common version used at Feedzai).
13. ***shared_yscale***: Whether to share (forcibly, since the scales can coincide) the Y-axis between the charts for the same bin width. In other words, a single scale for the Y-axis is adopted. It only works for *"group"* (nevertheless, two axes are drawn because this version can grow a bit horizontally, thus facilitating individual reading) and *"facet"* (only the leftmost Y-axis is drawn) modes (with *show_threshold_area* set to *False*). The default value is *False*.

The possibility of defining a list of bin widths, instead of just one, stemmed from the feedback shared by data scientists. Depending on how a data scientist bins, it is possible to have slightly different pictures of how the actual underlying distribution is (Figure 3.17, for example). So, it is more robust to look at different bin sizes. Oftentimes, the variation between distributions will be in the score range from 0 to 0.1. In this way, with this flexible parameter, a data scientist can "zoom in" on this specific section of the distributions and see if there is a different *curve* in this left part, for example.

When the Y-axis scale is not shared (in the grouped and faceted breakdown versions), that is, each subchart has an independent Y-axis scale per row (the scales can end up being the same, depending on the data), the logging message in Figure 3.16 is displayed by default (thus providing additional information about the histograms). This feature, a prototype of the idea of using logging or something similar to alert the data scientist to relevant details of each chart generated, was implemented using the built-in *logging* module, a custom logger (interface) and the *INFO* logging level. The idea was inspired

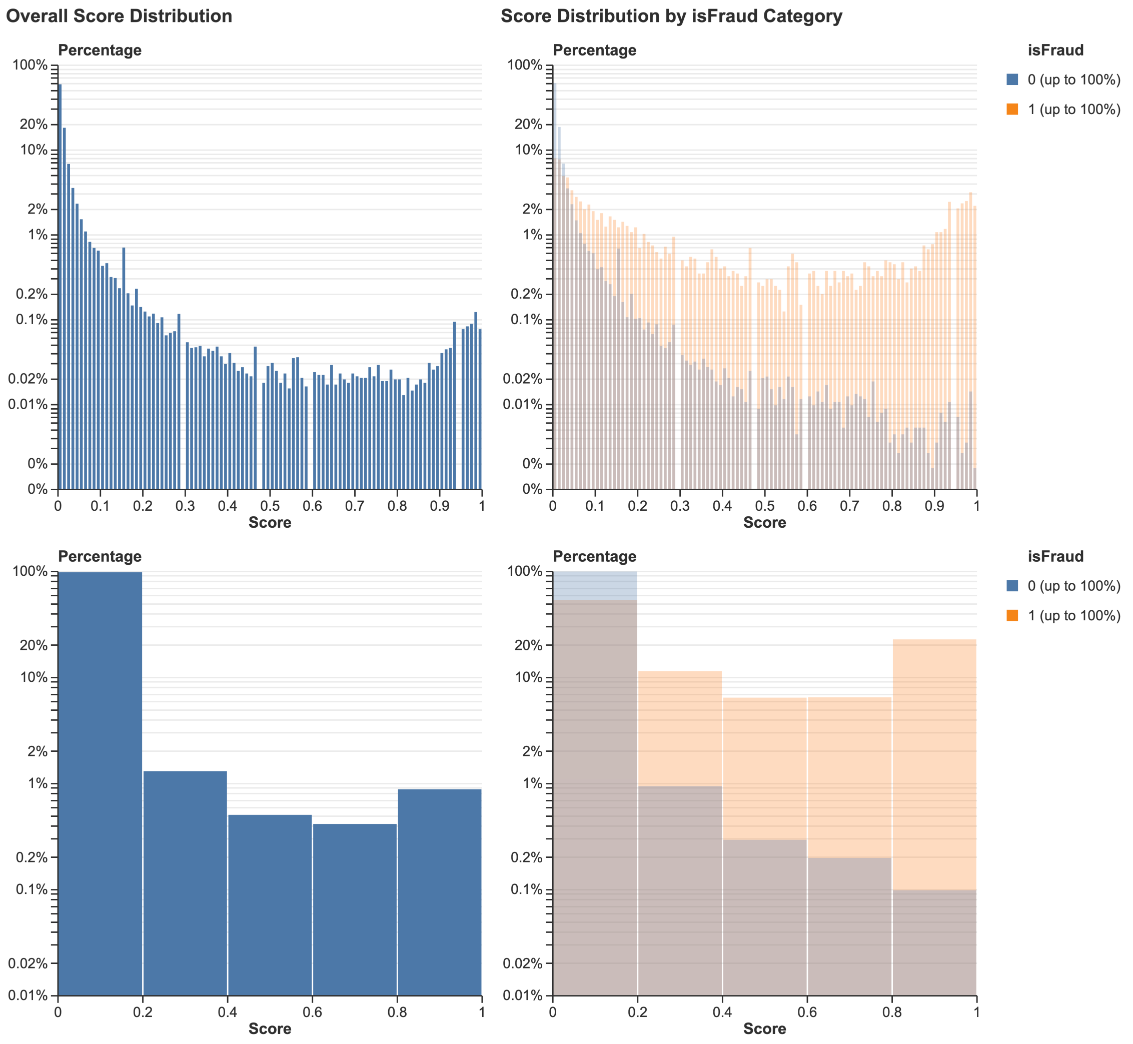


Figure 3.15: Layered score distribution chart. In this example, the second column shows the distribution of the scores obtained for the fraudulent instances and for the non-fraudulent instances separately (according to the target labels, not with the classification obtained from a model and a threshold). The first row has a bin width of 0.01, while the second one has a bin width of 0.2. The scale of the Y-axis is a logarithmic scale.

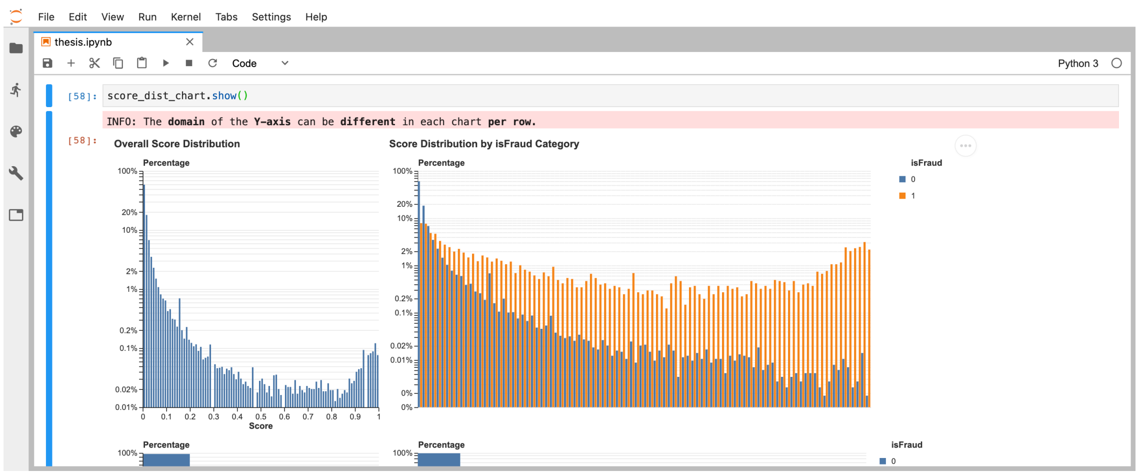

```
[58]: score_dist_chart.show()
```
```
INFO: The domain of the Y-axis can be different in each chart per row.
```

Figure 3.16: When showing a grouped or faceted score distribution chart with independent Y-axis scales, MevaL displays the following (logging) message to alert the data scientist to be careful when trying to compare distributions directly: *INFO: The domain of the Y-axis can be different in each chart per row.*.

by VisuaLint [115], a recent technique for surfacing chart building errors *in situ* (like the red wavy underlines for spelling mistakes in certain word processor programs), and the concept of code linting (especially in terms of the messages that appear on the terminal).

In Figure 3.17, there is an example of the threshold areas, plotted in the background of each subchart, as well as at the bottom of each one of them. In the background, since these areas are defined using score-based threshold markers, they are limited to cover the respective score subspace, regardless of the number of instances in each area. In addition to tooltips with precise information on the percentage of instances in these areas, it is also possible to visually contrast the their size with the histogram bars. On the other hand, at the bottom of each subchart, the single thin bar functions as a normalized stacked bar chart, showing the proportion of instances in each of the three plotted areas.

As a side note, although this chart can be used in Model Monitoring contexts, with or without labels for the instances, its design focuses on offline Model Evaluation where labels exist. Also, no density estimation mechanisms (like kernel density estimation, one of the methods to estimate the underlying probability density function from observed data [308]) have been implemented since Feedzai's data scientists typically look only at bars when relying on visual inspection of these charts.

3.6.7 Tabular Performance Analysis

3.6.7.1 Table styling

In order to facilitate the analysis and communication of tables, MevaL stands on the shoulders of pandas and provides a set of declarative-like functions that allow the customization of the visual styling for the familiar and easy to use tables rendered by this package. To make this possible, MevaL leverages the CSS-based Styling API for DataFrames (Series

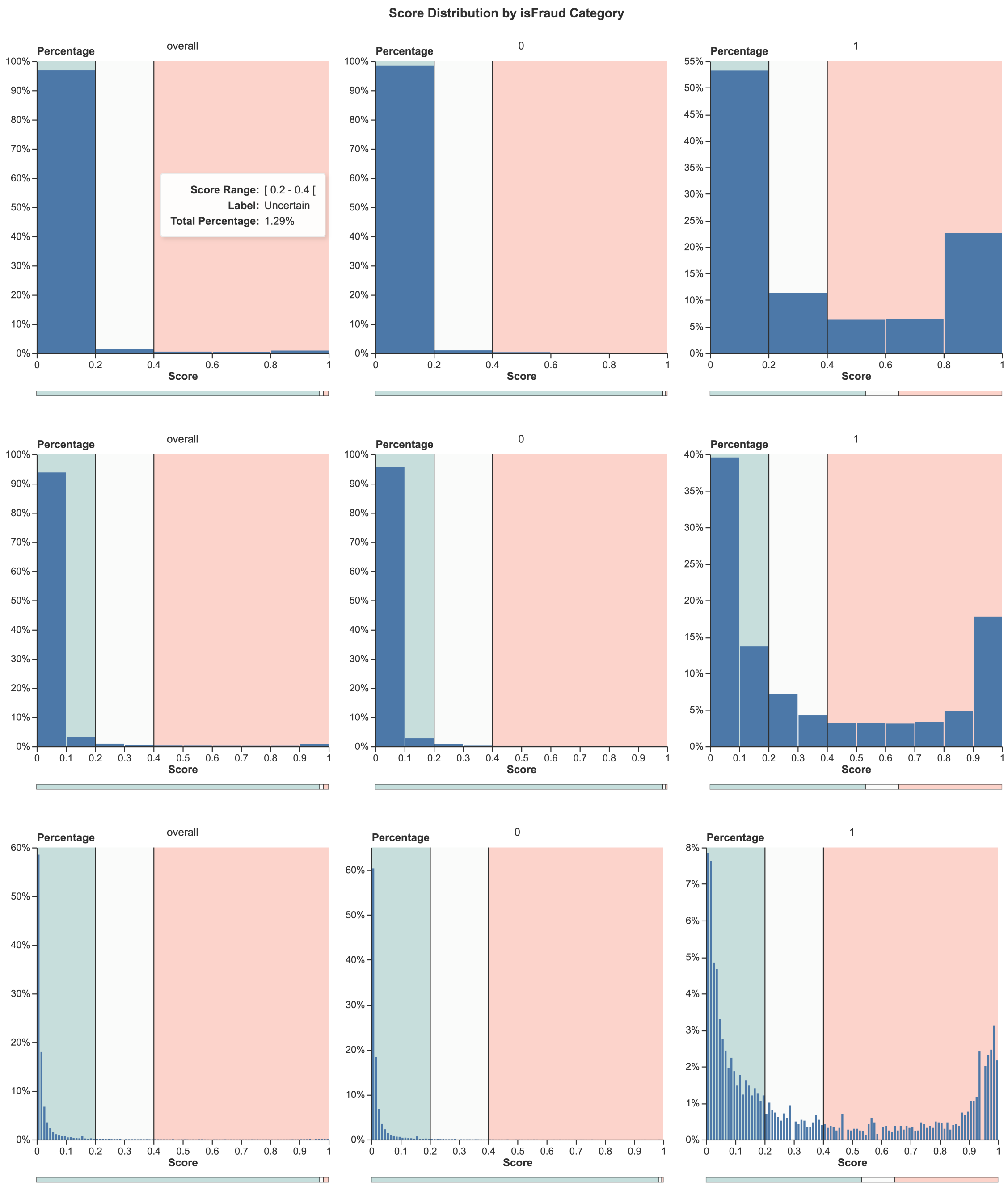


Figure 3.17: An example of the threshold areas for the (faceted) score distribution chart. The scores 0.2 and 0.4 were the scores chosen to depict the threshold markers. In addition, it is also possible to see an example of a tooltip for one of the areas in the first subchart. All bars have the same color, instead of one color per distribution, to prevent visual clutter. Note that the Y-axis scales tend to be different for each subchart.

must be converted to DataFrames) available in pandas. So, from a pandas DataFrame object, (hard or conditional) formatting (at different levels, such as at the cell level, at the column level, at the row level, among others) can be applied using the *style* property. This property returns a *Styler* object, which can be incrementally formatted (via different built-in methods) to define the desired style for the table rendered by pandas.

That said, MevaL provides a set of Python decorators (a decorator is a function that takes another function and extends its behavior without explicitly modifying it) that can be applied to a function whose purpose is merely (at the very least, since the function can perform other actions, such as certain data transformations, for example) to return and display the pandas DataFrame passed as an argument. An example of a decorator implemented in MevaL to round up float values (more precisely to define a certain precision in terms of the number of decimal places) and choose the presentation type is as follows:

Listing 3.2: A decorator to round up the float values displayed on a pandas DataFrame. An important aspect of this function, which allows multiple decorators to be applied to the same function in any order, is the try-except block (all implemented decorators follow the structure of this example). Since the *style* property is a property of the *DataFrame* class only, and not of the *Styler* class, the decorator must call the relevant style method depending on whether the object in question is a *DataFrame* or a *Styler* previously formatted by another decorator.

```python
from functools import wraps


def get_round_cols(data):
    return list(data.select_dtypes(include=["float"]).columns)


def round(decimal_places=2, value_format_type="f"):
    def decorator_round(func):
        @wraps(func)
        def wrapper_round(*args, **kwargs):
            value = func(*args, **kwargs)

            try:
                cols = get_round_cols(value)
                return value.style.format(
                    f"{{:.{decimal_places}{value_format_type}}}", subset=cols
                )
            except AttributeError:
                cols = get_round_cols(value.data)
                return value.format(
                    f"{{:.{decimal_places}{value_format_type}}}", subset=cols
                )

        return wrapper_round

    return decorator_round
```

To use this decorator and others, it is only necessary to apply them (and optionally pass the necessary arguments as in any other function), in any order (they are independent of the order), to a display function like the following:

Listing 3.3: An example of a decorator applied to a function that will serve to show an arbitrary pandas DataFrame with a precision of three decimal places.

```python
from meval import tables


@tables.round(3)
def displayer(df):
    return df
```

With this in mind, MevaL has a set of 14 decorators ready to be used:

1. ***round()*** (Figure T.18): Decorator for rounding float columns (more specifically, to set their precision in terms of decimal places). This decorator has two parameters:

   a) *decimal_places*: The number of decimal places. The default value is *2*.

   b) *value_format_type*: An identifier (according to Python's format specification mini-language [236]) for the desired presentation format for floats. The default value is *"f"* (fixed-point notation, that is, all floats have exactly the same number of digits).

2. ***bar()*** (Figure T.19): Decorator for (diverging) bar chart in the background of the specified columns/features. It can be particularly useful for looking at FI datasets (where the FI values are arranged in a column). This decorator has one parameter:

   a) *cols*: The name(s) of the column(s)/feature(s).

3. ***caption()*** (Figure T.20): Decorator for table caption (in the upper left corner above the table). This decorator has one parameter:

   a) *text*: The text for the table caption.

4. ***border_rows()*** (Figure T.21): Decorator for horizontal bordering (its logic is different from that used in the *border_cols()* decorator). This decorator has three parameters:

   a) *col*: The name of the column/feature that defines the location of the horizontal borders based on their unique values (assuming the pandas DataFrame is ordered by this column/feature because, otherwise, the borders will be added at the end of the pandas DataFrame between rows with different values).

   b) *sort*: Whether to order the pandas DataFrame by the specified column before applying the borders. The default value is *False*.

   c) *order*: Sort ascending (*"asc"*) or descending (*"desc"*). The default value is *"asc"*.

5. ***border_cols()*** (Figure T.22): Decorator for vertical bordering (its logic is different from that used in the *border_rows()* decorator). This decorator has one parameter:

   a) *cols*: The name(s) of the column(s)/feature(s) that define(s) the location of the vertical border(s). The borders are positioned to the left of the columns/features.

6. ***magnify()*** (Figure T.23): Decorator to magnify individual cells or rows on mouse hover (it can be useful for presentations, for example). This decorator has two parameters:

   a) *font_size*: Font size (CSS *font-size* property) when hovering. The default value is *16* (in points).
   b) *mode*: Magnify only the hovered *"cell"* or the entire corresponding *"row"*. The default value is *"row"*.

7. ***highlight_rows()*** (Figure T.24): Decorator to highlight specific rows (the highlight is made through a black background and white text). This decorator has one parameters:

   a) *filter_criteria*: A pandas query string to evaluate (the matched rows will be highlighted).

8. ***highlight_cols()*** (Figure T.25): Decorator to highlight specific columns. The highlight is made through a background with a user-defined color and white or black text, according to the color that best contrasts with the chosen background color. This decorator has three parameters:

   a) *cols*: The name(s) of the column(s)/feature(s) to highlight.
   b) *color*: The background color. The default value is *"#2F2F2F"* (MevaL's black color).
   c) *highlight_col_heading*: Whether to highlight the cell with the name of each column in the DataFrame header. The default value is *True*.

9. ***sort()*** (Figure T.26): Decorator to sort, with the addition of Unicode arrows [323] in the column/feature names to identify the sorting criteria. This decorator has two parameters:

   a) *cols*: The name(s) of the column(s)/feature(s) by which to sort the pandas DataFrame.
   b) *order*: Sort ascending (*"asc"*) or descending (*"desc"*). The default value is *"asc"* (it is used in all specified columns).

10. ***sticky_header()*** (Appendix W): Decorator to make the pandas DataFrame header a fixed/sticky header. In this way, the height of the DataFrame is*fixed* to an arbitrary number of rows, allowing the data scientist to navigate (to scroll up/down) along all the rows without this implying that the DataFrame extends across the computational notebook. This decorator has one parameter:

    a) *height*: The height, in pixels, of the pandas DataFrame. The default value is *300*.

11. ***hide_index()*** (Figure T.27): A decorator to hide the index of a pandas DataFrame (the leftmost integer column). This decorator has no parameters.

12. ***hide_null_cols()***: A decorator to hide all columns made up only of null values. This decorator only considers the pandas DataFrame passed as an argument of the display function, be it a complete DataFrame or just a slice. This decorator has no parameters.

13. ***highlight_null_values()*** (Figure T.28): A decorator to highlight the null values present in a pandas DataFrame. The *highlight_rows()*, *highlight_cols()* and *shade_alternate_rows* decorators overlap (they have higher priority) in case of joint use in the relevant cells. This decorator has one parameter:

    a) *color*: The color to highlight cells with null values. The default value is *"#EE6C4D"* (MevaL's red color).

14. ***shade_alternate_rows()*** (Figure T.29): A decorator to apply one of two background colors to the rows of a pandas DataFrame alternately (excluding the header). The color is alternated if the next value is different, that is, the rows with the same contiguous value are colored with the same color. The *highlight_rows()* and *highlight_cols()* decorators overlap (they have higher priority) in case of joint use in the relevant cells. This decorator has four parameters:

    a) *col*: The name of the column/feature by which to guide color switching.
    b) *sort*: Whether to order the pandas DataFrame by the specified column before coloring the rows. The default value is *False*.
    c) *order*: Sort ascending (*"asc"*) or descending (*"desc"*). The default value is *"asc"*.
    d) *colors*: The colors for the rows (their order matters). The default value is *["#FFFFFF", "#F5F5F5"]* (MevaL's/pandas' white color and pandas' light gray color).

The idea of implementing style decorators for pandas DataFrames came from the PrettyPandas package [97], a small Python package that leverages the Styling API to provide chainable methods for formatting values and adding summary rows/columns via a custom pipeable class, and Tailwind CSS [309], a utility-first CSS framework (it provides HTML-ready classes like *text-center* and *text-lg*, for example, thus inspiring the basic idea for these declarative decorators). On the other hand, the idea of adopting or adapting a general-purpose graphical user interface (GUI) package for pandas, such as D-Tale [268], PandasGUI [253], bamboolib [146] (it is not an open-source package), and Qgrid [238], was also investigated (EDA report build packages, like Sweetviz [32] and pandas-profiling [43], were also checked, as well as Lux [163], a *semi-automatic* EDA package using Altair). However, this idea was dropped because of the extra dependencies, the amount of work needed to adapt one (or more) of the options properly, and/or because it is a more disruptive choice for the JupyterLab environment and for the current workflow followed by Feedzai's data scientists (and also due to the lack of support for some of the front-facing customizations implemented).

For the *highlight_cols()* and *highlight_rows()* decorators, the font color (pandas' black or pandas' white) is automatically selected based on the chosen background color, so that there is an appropriate contrast between both colors. To do so, the 6-digit hexadecimal (background) color is converted to an RGB color [133] and each channel is used individually in the following (normalized) formula [16, 86, 249]:

$$Brightness = \frac{0.299 \times Red + 0.587 \times Green + 0.114 \times Blue}{255} \tag{3.1}$$

This formula (based on the formula to convert from the RGB color space to the YIQ one, namely the part used to calculate the Y component representing luma/brightness [249, 337]) is used to calculate the color brightness, more precisely the perceived brightness for a color, and if this value is greater than 0.5 (the chosen threshold that represents the intermediate value between 0 and 1), the implemented function will return the black color (since the background color is *bright*); otherwise, the function will return the white color (since the background color is *dark*) [86, 249]. This formula was the only one tested and implemented since it worked empirically after manual testing of various background colors.

When using the *set_table_styles()* instance method of the *Styler* class, it is necessary to save the list of *table_styles* (instance variable) in a dummy variable, for example, as this list (if not empty) must be concatenated with the list of new styles to apply (otherwise, the new styles will simply *overwrite* the ones already applied previously).

3.6.8 Threshold Tuning

Threshold Tuning corresponds to the process or set of processes followed in order to choose the most appropriate classification threshold for the decisions of a ML model. It does not necessarily imply the choice of a single threshold, as several thresholds can maximize the results obtained, such as different thresholds for certain values in a breakdown field, for example. Other charts, such as those in the Visual Performance Analysis section, can also help with this choice.

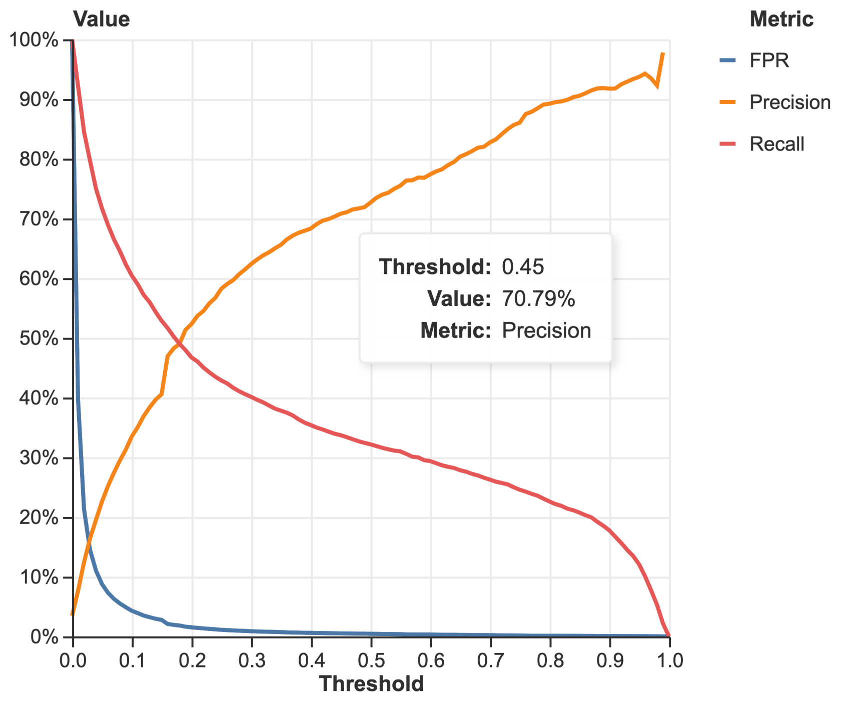


Figure 3.18: Overall classification threshold chart.

3.6.8.1 Classification threshold chart

The classification threshold chart (Figure 3.18) is a line chart of the (overall) value of one or more performance metrics against the classification threshold (throughout its range). However, if a breakdown field is specified, this chart is broken down into multiple charts (small multiple), one for each performance metric, with each line corresponding to a different breakdown value (Figure 3.19). This variation is particularly useful to help the data scientist to have a glimpse of the relationship between a subspace of classification thresholds and the respective values for different performance metrics at the subgroup level, since the choice of a single threshold may imply disparate results in these subgroups of interest. That said, the parameters of the respective class are:

1. ***data***: The performance metrics dataset (pandas DataFrame) to be plotted.
2. ***xvar***: The column name with the values for the X-axis (classification threshold column).
3. ***metrics***: The name(s) of the performance metric(s) to be plotted.
4. ***breakdown_field***: Instead of a single chart with one line for each specified performance metric, show different charts for each specified performance metric where each line concerns one of the unique values of a breakdown column, as well as the overall performance. The default value is *None* (single chart for all specified performance metrics).

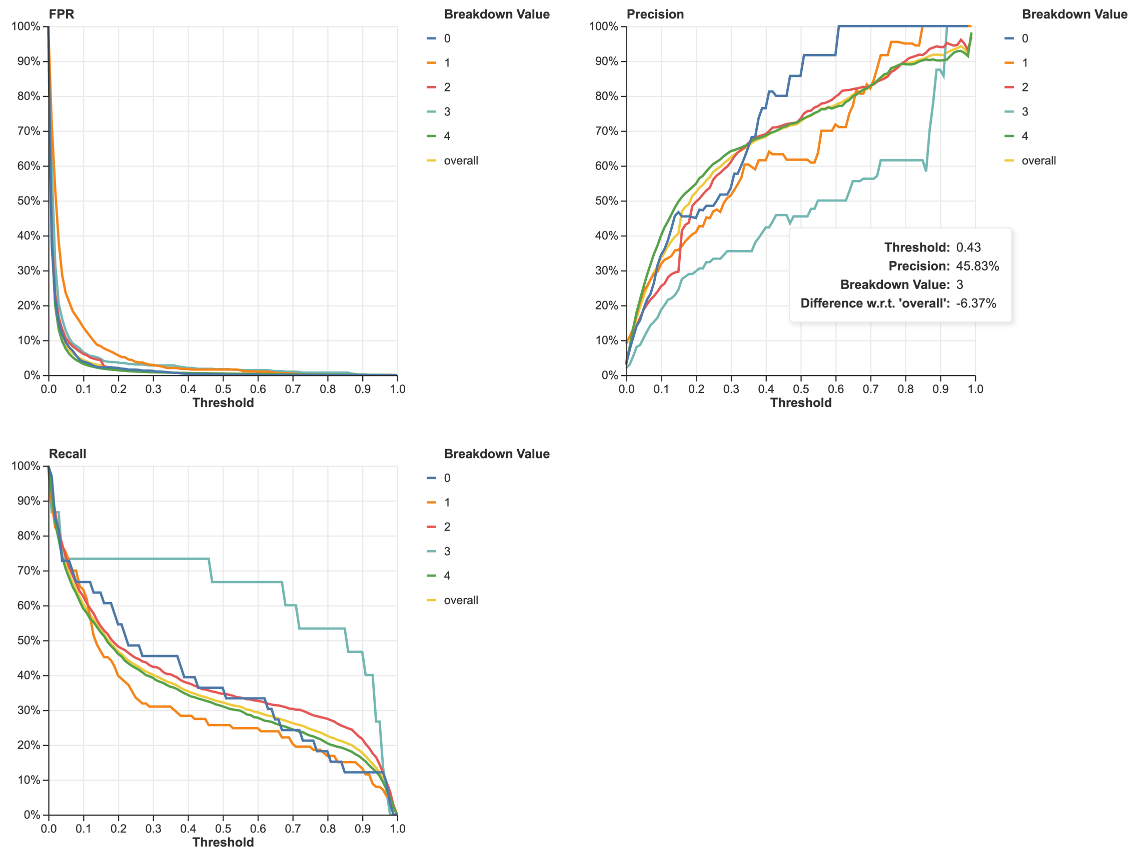


Figure 3.19: Classification threshold chart for the *card4* breakdown field. The tooltip provides extra information, such as the difference in value with respect to overall performance (considering a certain threshold).

3.6.9 Uncertainty Estimation

3.6.9.1 Binomial proportion confidence interval

In MevaL, in order to *extend* the point estimate to three of the supported (count-based) performance metrics (FPR, Precision, and Recall), there is the option to compute confidence intervals. When a model is evaluated in the test set, for example, the exact Recall of the model is not obtained, for example, but an estimate. Even when performance metrics are calculated from entire datasets, computing confidence intervals around the obtained values is a good informative practice, especially when comparing multiple models. In other scenarios, confidence intervals can also be useful to assess whether a given point estimate is very sensitive to a training/test split (thus having high variance) or not [245].

In this way, a possible approach for the calculation of confidence intervals is via the Normal approximation/Wald method [306, 325], thus generating binomial proportion confidence intervals [245, 269, 325]. To this end, it is assumed that the predictions (that is, the predicted labels according to the scores of a model and a certain classification threshold) follow a Normal distribution. Thus, the confidence intervals will be computed from the mean in a given dataset (a test set, for example, or, in practice, a subset according to the denominators of the supported performance metrics) under the Central Limit Theorem. That said, each prediction can be interpreted as a Bernoulli trial, with the number of correct predictions following a binomial distribution ($X \sim (n, p)$) with $n \in \mathbb{N}$ instances, k trials, and the probability of success $p \in [0, 1]$ (the probability of failure, an incorrect prediction, is $q = (1-p)$). This random variable has a variance of $\sigma^2 = np(1-p)$ and a standard deviation of $\sigma = \sqrt{np(1-p)}$ (a necessary element to compute the confidence intervals) [245, 306, 324, 325]. The (estimated) performance metric value ($metric$) here is like the proportion of successes, where the numerator is the number of possible successes (a specific confusion category) and the denominator is the number of trials (so, it can be used to estimate the probability of success p). Therefore, since the *average* number of successes is the value of interest, not the associated absolute value, the standard deviation formula can be adapted as follows:

$$\sigma = \sqrt{\frac{1}{n} metric(1 - metric)} \tag{3.2}$$

So, the Normal approximation/Wald interval can be computed as follows (in this work, only FPR, Precision, and Recall are considered possible values for $metric$, although these confidence intervals can also be computed for other performance metrics, such as Accuracy and False Negative Rate (FNR) [96]):

$$metric \pm z\sqrt{\frac{1}{n} metric(1 - metric)} \tag{3.3}$$

z is the $1-\alpha/2$ quantile of a standard Normal distribution and α is the error quantile. Moreover, for a 95% confidence level ($\alpha = 1-0.95 = 0.05$), $1-\alpha/2 = 0.975$ and $z = 1.96$ [245, 306, 325]. In addition, as expected, n (the denominator) is different depending on the performance metric in question [96]:

- For FPR, $n = FP + TN$ (# legitimate instances).
- For Precision, $n = TP + FP$ (# predicted positives).
- For Recall, $n = TP + FN$ (# fraudulent instances).

On the other hand, the assumption at stake is known not to apply well with very small datasets or when the success probability is close to 0 or 1 (in this case, the true values of the performance metrics) [306]. Also, given that the upper and lower values may overshoot, they are clipped to be in the [0, 1] interval [269]. Although there are more reliable, more or less computationally demanding alternatives (including other similar methods that try to overcome the problems identified), the great advantage of this approach (or similar ones) is that it does not require any resampling method, such as bootstrapping or CV, or to vary the seed used to train a particular model, for example. From a single dataset (pandas DataFrame), the confidence intervals are calculated with the values previously obtained for the confusion categories and for the performance metrics themselves directly. It was for this simplicity that this specific method was adopted in MevaL in order to kick off uncertainty estimation features (in addition, within the binomial proportion confidence interval landscape [306, 325], the simplest method, the one described in this section, was chosen to introduce this topic more easily).

3.6.9.2 Confidence interval chart

The confidence interval chart (Figure 3.20) is a dot plot with error bars that complements the point estimates for a set of performance metrics with binomial proportion confidence intervals via normal approximation. This chart complements the tabular analysis of concrete values for performance metrics and their confidence intervals with an overview for raising red flags or helping to choose a model (when plotting several of these charts), for example. That said, the parameters of the respective class are:

1. ***data***: The performance metrics dataset (pandas DataFrame) to be plotted.
2. ***metrics***: The name(s) of the performance metric(s) to be plotted. The supported values are FPR, Precision, and Recall.
3. ***threshold***: The classification threshold to consider for performance metrics.
4. ***shared_xscale***: Whether to consider a unique X-axis scale for all performance metrics, thus forming a single dot plot (otherwise, in practice, different charts will be generated and stacked on each other, forming a small multiple where each constituent chart has an X-axis scale adapted to its range of values). This parameter is particularly useful if the common range of values is significantly different between metrics (as can happen between low FPR values and relatively high Recall values, for example). The default value is *False*.

3.6.10 Visual Performance Analysis

3.6.10.1 Confusion category bar chart

The confusion category bar chart (Figure 3.21) is a dual bar chart for comparing the absolute values of a confusion category (TP, FP, TN, and FN) in general against the breakdown values coming from a breakdown field and for comparing the same breakdown values only between them. This chart contrasts with the other charts in this subpackage by working with absolute metrics instead of ratio-based ones (excluding the confusion matrix). That said, the parameters of the respective class are:

1. ***data***: The performance metrics dataset (pandas DataFrame) to be plotted.
2. ***xvar***: The breakdown field name with the (categorical) values for the X-axis. The bar for the overall value is included by default.
3. ***yvar***: The column name with the values for the Y-axis (category confusion column).

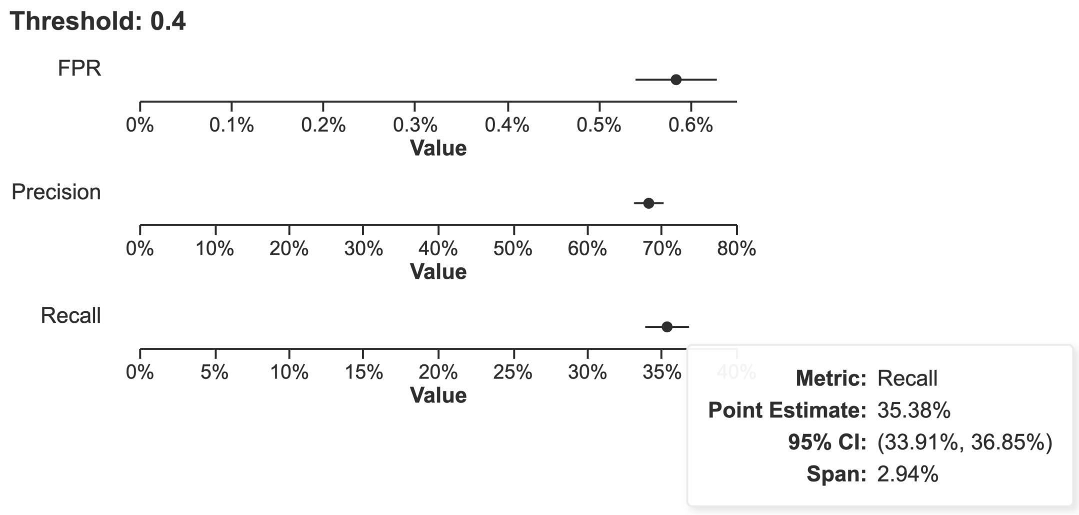


Figure 3.20: Classification threshold chart. Each horizontal axis has a title to point out that each of the scales is different, and it is not possible to make comparisons between charts directly.

4. ***threshold***: The classification threshold to consider for the confusion category.
5. ***show_bar_labels***: Whether to display the text labels above each bar with its absolute value encoded by the respective height (so as to include the precise value statically). The default value is *True*.

3.6.10.2 Confusion matrix

The confusion matrix (Figure 3.22) is a contingency table (or 2x2 matrix) that summarizes the values obtained for the four confusion categories (TP, FP, TN, and FN) in a single layout. That said, the parameters of the respective class are:

1. ***data***: The performance metrics dataset (pandas DataFrame) to be plotted.
2. ***at_value***: The classification threshold of interest (if the *reference* argument is *"threshold"*) or the reference value for the *reference* argument performance metric for which the classification threshold should be obtained. In the case of the second behavior, the classification threshold is obtained depending on the value of a given reference performance metric (if the exact value does not exist in the dataset, the closest smaller value is used). This option is important for the data scientist because, in many cases, they are guided by a certain reference value (or upper bound) for a given performance metric and do not directly use the classification threshold.
3. ***mode***: The chart mode, that is, the type of chart to display. The supported values are *"heatmap"* (the typical confusion matrix) and *"quadrant"* (a layout where the area of each square in each quadrant encodes the number of instances in each of the confusion categories [7]). The default value is *"heatmap"*.

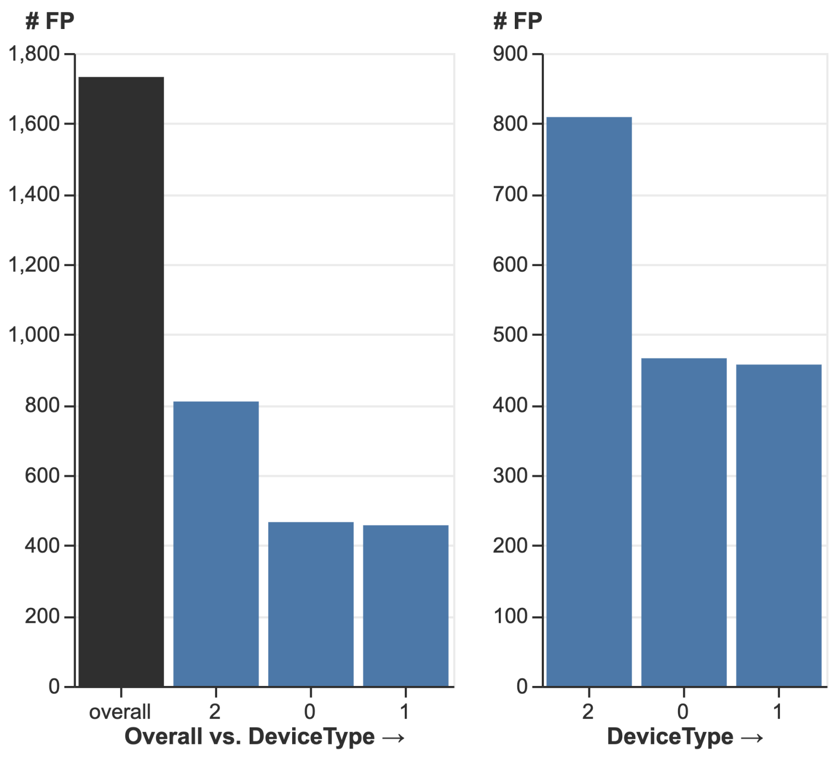


Figure 3.21: FP-based confusion category bar chart. In this example, the *DeviceType* breakdown field is analyzed and the classification threshold is 0.2.

4. ***breakdown_field***: The name of a breakdown field for which to plot a confusion matrix for each of its breakdown values. The default value is *"overall"* (single confusion matrix considering the general results).
5. ***count_mode***: If a breakdown field is specified, consider a single scale for all confusion matrices (*"shared"*) or different scales adapted to the domain of each of the generated charts (*"independent"*). The default value is *"shared"*.
6. ***reference***: The performance metric to be used as a reference to obtain the classification threshold for which to select the values to be plotted or simply *"threshold"* in order to specify a classification threshold using the *at_value* argument. The default value is *"threshold"*.
7. ***show_count_prefix***: Whether to format the absolute numbers (counts) of each quadrant so as to follow decimal notation with an SI prefix (without insignificant trailing zeros). The default value is *True*.

The Blues sequential single-hue color scheme (a common choice in other packages [203, 217]) is used in the heatmap version of the confusion matrix on a single scale for all confusion categories. In an imbalanced context like this one, the number of TNs will tend to be significantly higher than the rest, which, in practice, will lead to this quadrant having a dark blue as the background color and the remaining lighter blue tones. Although this

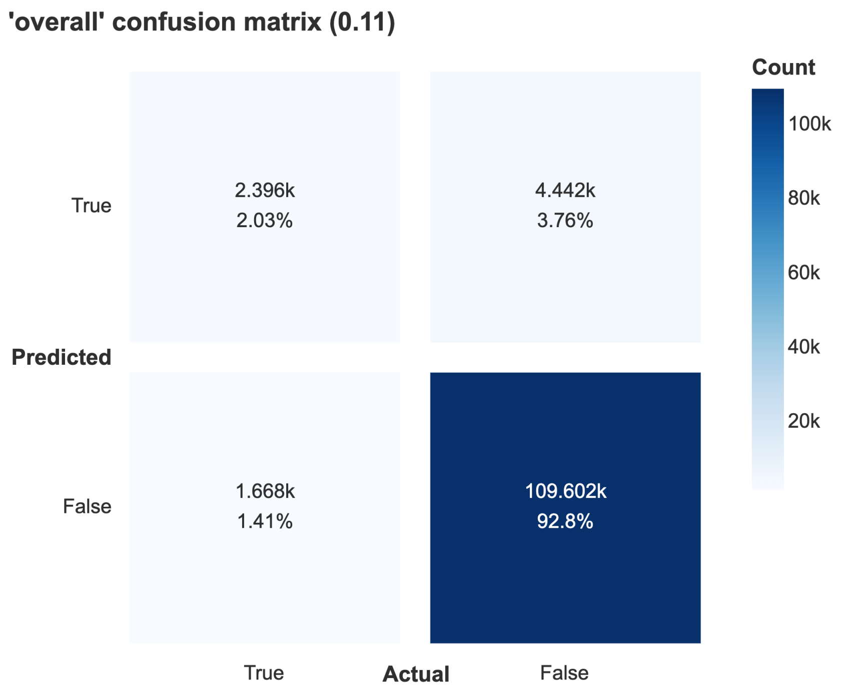


Figure 3.22: Overall confusion matrix. The selected classification threshold (0.11) was obtained for a Recall value of 60%.

color scheme is not optimized for this scenario, it is maintained to function as a kind of flag to signal strange behavior in the distribution of instances across the four confusion categories. Another option would be to have a color scheme only for the three quadrants other than the one regarding TNs.

3.6.10.3 ROC, PR, and Gain curve chart

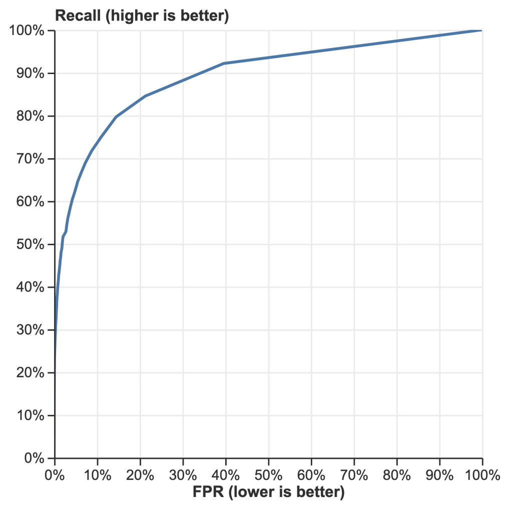
The ROC curve chart (Figure 3.23a) shows the trade-off between Recall and FPR, while the PR curve chart (Figure 3.23b) the trade-off between Precision and Recall [30]. On the other hand, the Gain curve chart shows the trade-off between Recall and AR (the main idea is to show the effectiveness of a model by comparing the percentage of TPs and the percentage of alerted instances, that is, instances classified as positive ones). In other words, for a set of classification thresholds, it is possible to see how two different performance metrics behave. Each of these charts also has a partial version (in this context, a partial ROC/PR/Gain curve chart implies that one or both axes contain only a portion of the [0, 1] interval). That said, the parameters of the respective classes are:

1. ***data***: The performance metrics dataset (pandas DataFrame) to be plotted.
2. ***xvar***: The column name with the values for the X-axis (FPR for the ROC curve, Recall for the PR curve, and AR for the Gain curve).
3. ***yvar***: The column name with the values for the Y-axis (Recall for the ROC curve, Precision for the PR curve, and Recall for the Gain curve).

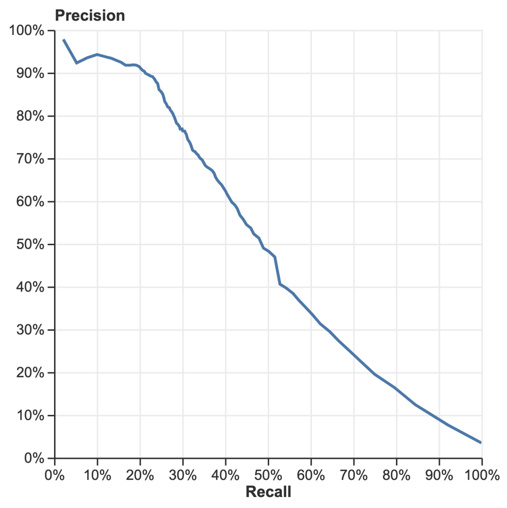
4. ***partial***: A dictionary whose keys identify the axes (X-axis and/or Y-axis) and the values identify the visible limits (minimum and maximum) for them (in order to obtain a partial ROC/PR/Gain curve). Values can be specified in the range [0, 1] or [0, 100]. This hard way of generating partial charts was implemented for two reasons: (1) in Altair 2.4.1, it is necessary to rerun the respective cell when zooming and panning to return to the initial state; (2) to allow defining static partial charts. An example of a possible argument (to limit the FPR to 5%) is as follows: *"X-axis": [0, 5], "Y-axis": [0, 100]*. The default value is *None* (full curve).
5. ***overall***: Whether to show the overall ROC/PR/Gain curve (considering the complete dataset). The default value is *True*.
6. ***breakdown_fields***: The name(s) of the breakdown column(s)/feature(s) to plot as individual ROC/PR/Gain curves. The default value is *None*.
7. ***partial_mode***: The way to trim the full ROC/PR/Gain curve according to the defined limits (in order to obtain the partial ROC/PR/Gain curve). There are three different modes available: *"clip"* (to cut the curve at the defined limits), *"clamp"* (to cut the curve at the defined limits and move points beyond the limit to the edge of the axis), and *"filter"* (to explicitly filter the dataset based on the defined limits). The default value is *"clip"*.
8. ***show_points***: Whether to show the points that make up the ROC/PR/Gain curve.
9. ***highlight_threshold***: Highlight the point that corresponds to the classification threshold specified in all curves present in a chart and with a different shape (a star) than the one added by *show_points*. This parameter can be useful to check the impact of choosing a certain classification threshold on the various breakdown values plotted. The default value is *None*.
10. ***show_axis_annotations***: Whether to add an annotation next to the titles of each axis to give a clue on how to interpret the respective values (*Recall (higher is better)* and *FPR (lower is better)*). This parameter works only for the ROC curve chart. The default value is *False*.
11. ***highlight_area***: The coordinates in the ROC/PR/Gain space to highlight a particular area for comparison with the curve(s). These coordinates must be specified in a sequence (list or tuple) in the following order: [y2, x2, y1, x1]. The default value is *None*.

3.6.10.4 Scatterplot

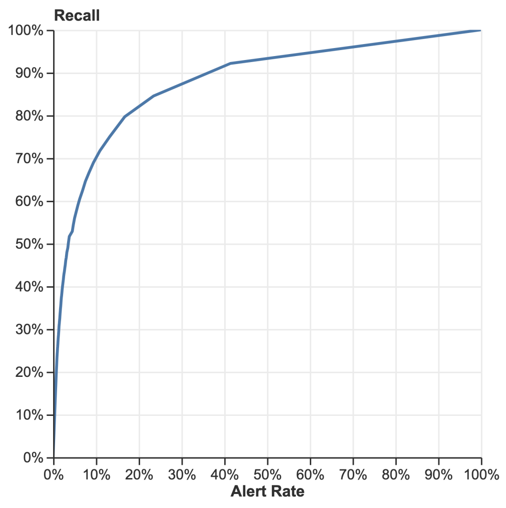
In MevaL, the main use case for the scatterplot (Figure 3.24) is to visualize two performance metrics considering a certain threshold and breakdown column/feature, that is, individual points are arranged for every single value of a breakdown column/feature, as



(a) ROC curve chart.



(b) PR curve chart.



(c) Gain curve chart.

Figure 3.23: Basic version of the curve charts available in MevaL. The ROC and Gain curves are similar in terms of shape, but the concrete values are different.

well as a point for the overall performance. That said, the parameters of the respective class are:

1. ***data***: The performance metrics dataset (pandas DataFrame) to be plotted.
2. ***xvar***: The column name with the values for the X-axis (a performance metric).
3. ***yvar***: The column name with the values for the Y-axis (a performance metric).
4. ***threshold***: The classification threshold to consider for performance metrics.
5. ***breakdown_field***: The name of the breakdown column/feature to plot as individual points.
6. ***xscale***: The scale type for the X-axis (*"linear"* or *"log"*). The default value is *"linear"*.

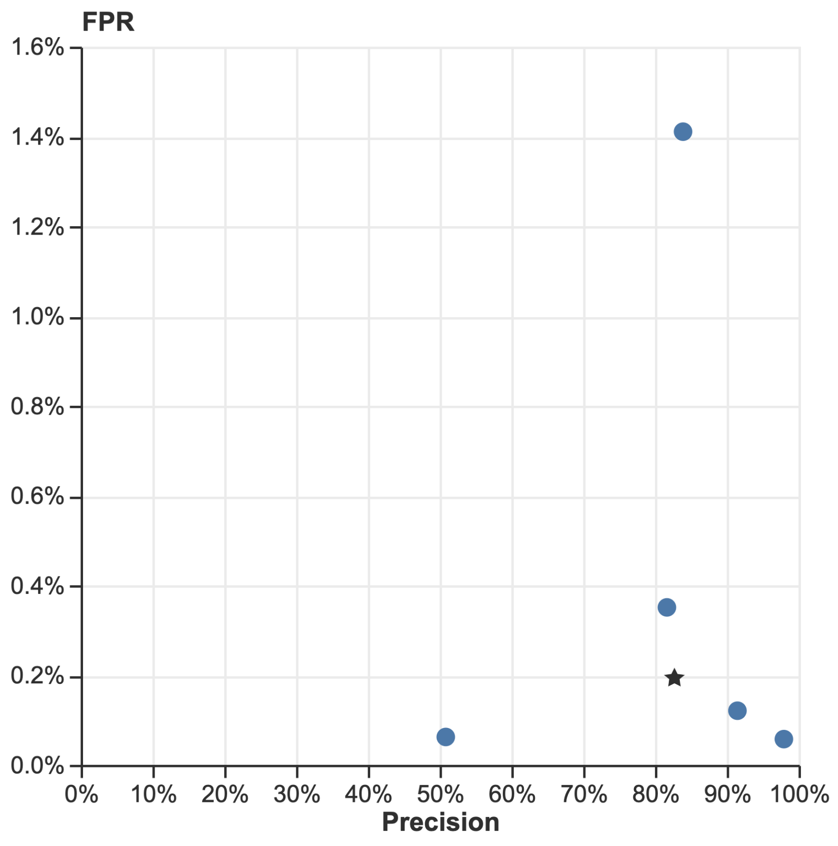


Figure 3.24: Scatterplot for the *ProductCD* breakdown field and for the classification threshold 0.7. The star (black) mark corresponds to the overall performance.

7. ***yscale***: The scale type for the Y-axis (*"linear"* or *"log"*). The default value is *"linear"*.

3.6.10.5 Breakdown strip chart

The breakdown strip chart (Figure 3.25) is a kind of one-dimensional (small multiple or not) chart or with a categorical vertical axis with a tick for overall performance (in one or more performance metrics of interest), as well as ticks for performance in different subgroups (considering a certain threshold and breakdown column/feature). The main objective of this chart is to show if the values of a breakdown are spread out, in comparison with the general performance, or if these values are close to the general one (and also to identify the subgroups whose performance is significantly inferior or superior). This chart was inspired by FairVis [5]. That said, the parameters of the respective class are:

1. ***data***: The performance metrics dataset (pandas DataFrame) to be plotted.
2. ***metrics***: The name(s) of the performance metric(s) to be plotted.
3. ***at_value***: The classification threshold of interest (if the *reference* argument is *"threshold"*) or the reference value for the *reference* argument performance metric for which

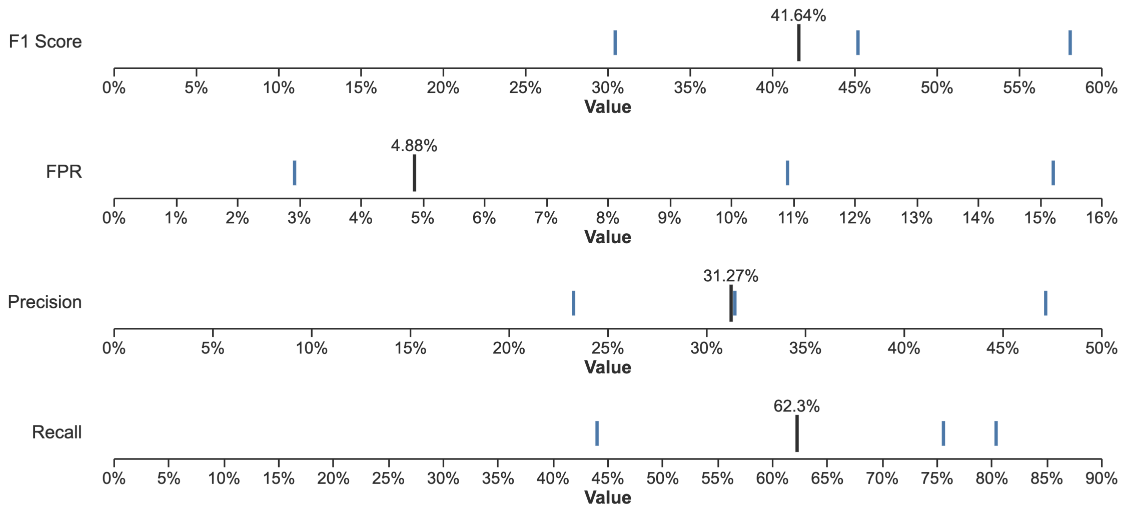


Figure 3.25: Breakdown strip chart for the *DeviceType* breakdown field. This example was generated from a reference value of 5% for FPR.

the classification threshold should be obtained. In the case of the second behavior, the classification threshold is obtained depending on the value of a given reference performance metric (if the exact value does not exist in the dataset, the closest smaller value is used). This option is important for the data scientist because, in many cases, they are guided by a certain reference value (or upper bound) for a given performance metric and do not directly use the classification threshold.

4. ***breakdown_field***: The name of the breakdown column/feature to plot as individual ticks.

5. ***mode***: How the X-axis scale should be defined. There are three modes available, the first two being absolute and the third one relative (and experimental): *"independent"* (each subchart, that is, each part referring to a performance metric, will have its own X-axis scale), *"shared"* (unique X-axis scale for all performance metrics), and *"relative"* (the X-axis scale reflects how many times each subgroup value is larger or smaller than the reference value, that is, the overall performance value for each metric of interest). The default value is *"independent"*.

6. ***reference***: The performance metric to be used as a reference to obtain the classification threshold for which to select the values to be plotted or simply *"threshold"* in order to specify a classification threshold using the *at_value* argument. The default value is *"threshold"*.

7. ***show_legend***: Whether to color each breakdown-based tick with a different color, in order to identify each breakdown value, and to add a legend. This flag parameter is particularly useful when this chart is used statically. The default value is *False*.

3.7 Deployment

In addition to the Minimum Viable Product (MVP)-style package developed, MevaL also integrates Feedzai's main DS Python package as one of its subpackages. So far, the structure, the base classes, the *entry point* class, and the data processing pipeline have been migrated/validated in a collective effort that also allowed to improve the previously developed code (and support most of it with proper docstrings and unit tests). The choice to migrate the general structure of MevaL and the data processing pipeline first was due to their connection with the API, the data scientist-facing part, and because the data processing pipeline is on the critical path for the adoption of this package, as a mechanism like this, doubly checked to see if there are no errors, is fundamental for the use of the remaining features of MevaL. As future work, the visual part will also be migrated.

Chapter 4

User Validation and Case Study

4.1 Dataset

In order to have a dataset accessible and relatively representative of Feedzai's domain, while being free of restrictions that prevent its sharing and manipulation, it was decided to use the IEEE-CIS Fraud Detection Kaggle competition [119] (labeled) training datasets for prediction score-based data gathering, demonstration and user-oriented validation purposes. The process overview and the corresponding script to prepare this dataset for use can be found in Appendix K.

In addition, Table 4.1 and Table 4.2 present some summary statistics of the Model Evaluation procedure and the datasets created, respectively, while Figure 4.1 shows the unbalanced proportion of the target classes (in particular for the raw test dataset separated from the original training datasets to effectively run the Model Evaluation procedure). As a complement, it is also possible to have a glimpse of each of these datasets (or pandas DataFrames) from Table 4.3 (performance metrics dataset), Table 4.4 (aggregate scores dataset), and Table 4.5 (FI dataset).

4.2 User Validation

In order to validate the first version of MevaL and to collect some qualitative feedback (some of the suggestions were subsequently implemented), five user interviews (this number was defined in a similar way to that presented in the User Interviews and Input Gathering section) were conducted with data scientists (the respective guidelines can be found in Appendix Q). The focus of these interviews was on the visual part of the package, thus ignoring the implemented data layer. Thus, the interviews were conducted online with the aid of a showcase (computational) notebook, allowing data scientists to

Table 4.1: Summary statistics derived from the arguments for the Model Evaluation conducted. The functional datasets used were derived from here. In the first two rows, the first values in parentheses refer to those before data processing and to non-time values, respectively.

Evaluation statistics	Value
Number of (#) breakdown fields	9 (7) + 1 ("overall")
# breakdown values	55 (11) + 1 ("overall")
# confidence intervals	3
# cost-based metrics	4
# count-based metrics	10
# decimal places for scores	2
# models	1
# threshold steps	101

Table 4.2: Summary statistics for each of the three functional datasets (pandas DataFrames) used. These numbers were obtained through the report generated by the pandas-profiling package [43]. The size statistics relate to the size in memory. The total size is obtained by adding the output of a pandas method that calculates the memory usage of each column (including the index) in bytes (the average size is obtained by dividing this value by the total number of rows). *Deep* size values are more accurate values, as they account for the system-level memory consumption by the *object* data types of each DataFrame. For comparative purposes, the raw test set from which these datasets were derived has a total (*deep*) size in memory of approximately 9.0 MiB (16.7 MiB).

Dataset statistics	Performance metrics	Aggregate scores	FI
# columns	32	6	3
# rows	8080	5625	365
Average record/row size	≈ 224.0 B	≈ 48.0 B	≈ 24.4 B
Average *deep* record/row size	≈ 415.4 B	≈ 243.1 B	≈ 77.1 B
Total size	≈ 1.7 MiB	≈ 263.8 KiB	≈ 8.7 KiB
Total *deep* size	≈ 3.2 MiB	≈ 1.3 MiB	≈ 27.5 KiB

see the charts and interact indirectly with them as the exchange of ideas took place. Of these five data scientists, three saw MevaL for the first time during these interviews, while the other two also participated in the initial ones.

For all participants, in general, the presented version of MevaL covers all the main features they expect to see in a package like this. In addition, in terms of design and aesthetics, they also approved the charts. Complementing this general perspective, some points were also raised that reinforce the developed solution, such as:

- For one of the data scientists, the classification threshold chart for a breakdown field is interesting and could have been useful for a previous project. The overall version was also highlighted by one of the data scientists.
- For one of the data scientists, a standardized package like MevaL can help all data

Table 4.3: Six sample rows from the performance metrics dataset. The first row corresponds to the general performance when choosing the value 0.45 as a classification threshold, while the remaining five rows refer to the performance for each of the subgroups of the *ProductCD* feature/breakdown field. Metric values are rounded to four decimal places.

model	breakdown_field	breakdown_value	threshold	TP	FP	TN
thesis	overall	overall	0.45	1369	565	113479
thesis	ProductCD	3	0.45	50	9	3309
thesis	ProductCD	1	0.45	70	30	3106
thesis	ProductCD	2	0.45	164	24	5215
thesis	ProductCD	4	0.45	157	177	91682
thesis	ProductCD	0	0.45	928	325	10167

FN	cost_TP	cost_FP	cost_TN	cost_FN	alert_rate
2695	151207.7188	64699.1094	1.5569e+07	458726.5938	0.0164
133	1585.0000	620.0000	2.5055e+05	6053.0000	0.0169
127	9960.0000	6524.0000	2.2744e+05	17700.0000	0.0300
93	45550.0000	7300.0000	7.6595e+05	21300.0000	0.0342
1653	43630.6484	39077.6016	1.3926e+07	382644.3438	0.0036
689	50482.0625	11177.5137	3.9879e+05	31029.2793	0.1035

precision_lower	precision	precision_upper	cost_precision	recall_lower
0.6876	0.7079	0.7281	0.7003	0.3223
0.7557	0.8475	0.9392	0.7188	0.2087
0.6102	0.7000	0.7898	0.6042	0.2885
0.8246	0.8723	0.9200	0.8619	0.5794
0.4165	0.4701	0.5236	0.5275	0.0738
0.7164	0.7406	0.7649	0.8187	0.5498

recall	recall_upper	cost_recall	f0point5_score	geometric_mean	f1_score
0.3369	0.3514	0.2479	0.5801	0.5790	0.4565
0.2732	0.3378	0.2075	0.5967	0.5220	0.4132
0.3553	0.4222	0.3601	0.5863	0.5932	0.4714
0.6381	0.6969	0.6814	0.8127	0.7970	0.7371
0.0867	0.0997	0.1024	0.2495	0.2942	0.1465
0.5739	0.5980	0.6193	0.7000	0.7457	0.6467

fpr_lower	fpr	fpr_upper	cost_fpr	f2_score	alerts	cost_alerts	mcc
0.0045	0.0050	0.0054	0.0041	0.3763	1934	215906.8125	0.4814
0.0009	0.0027	0.0045	0.0025	0.3161	59	2205.0000	0.4716
0.0062	0.0096	0.0130	0.0279	0.3941	100	16484.0000	0.4868
0.0028	0.0046	0.0064	0.0094	0.6743	188	52850.0000	0.7379
0.0016	0.0019	0.0022	0.0028	0.1036	334	82708.2500	0.2000
0.0277	0.0310	0.0343	0.0273	0.6010	1253	61659.5781	0.6211

Table 4.4: Five sample rows from the aggregate scores dataset. The first three rows are derived from all instances of the test dataset used, while the last two lines are obtained only from all fraudulent instances of this test dataset. The (non-multiplied) percentage values are rounded to four decimal places.

model	breakdown_field	breakdown_value	fraud_score	count	percentage
thesis	overall	overall	0.21	127	0.0011
thesis	overall	overall	0.75	34	0.0003
thesis	overall	overall	0.40	47	0.0004
thesis	isFraud	1	0.80	18	0.0044
thesis	isFraud	1	0.26	29	0.0071

Table 4.5: Nine sample rows from the FI dataset. The first six rows contain the three most important features for each of the FI results identified. The last three rows correspond to three examples of candidate features to be removed in a future model. In addition, although their position in the ranking changes significantly, in practice, there is no change in their value (although this example is hypothetical, supporting charts aimed at comparing FI results with filtering mechanisms either by ranking and/or by value can be important to find a subset of features interesting to analyze).

Feature	variable	value	fi	rank
card2	Absolute Importance	147	data/fi_data_kaggle.csv	1
DeviceInfo_device	Absolute Importance	139	data/fi_data_kaggle.csv	2
DeviceInfo_version	Absolute Importance	130	data/fi_data_kaggle.csv	3
DeviceInfo_version	Absolute Importance	147	data/fi2_data_kaggle.csv	1
P_emaildomain	Absolute Importance	139	data/fi2_data_kaggle.csv	2
DeviceInfo_device	Absolute Importance	130	data/fi2_data_kaggle.csv	3
V201	Absolute Importance	0	data/fi2_data_kaggle.csv	365
V202	Absolute Importance	0	data/fi2_data_kaggle.csv	366
V168	Absolute Importance	0	data/fi2_data_kaggle.csv	367

not_percentage	rank_diff	value_diff	box	absolute_change
True	-3	25	NaN	card2: 122 → 147 (+25)
True	-1	9	NaN	DeviceInfo_device: 130 → 139 (+9)
True	2	-17	NaN	DeviceInfo_version: 147 → 130 (-17)
True	-2	17	NaN	DeviceInfo_version: 130 → 147 (+17)
True	-2	17	NaN	P_emaildomain: 122 → 139 (+17)
True	1	-9	NaN	DeviceInfo_device: 139 → 130 (-9)
True	61	0	├	V201: 0 → 0 (+0)
True	61	0	├	V202: 0 → 0 (+0)
True	87	0	└	V168: 0 → 0 (+0)

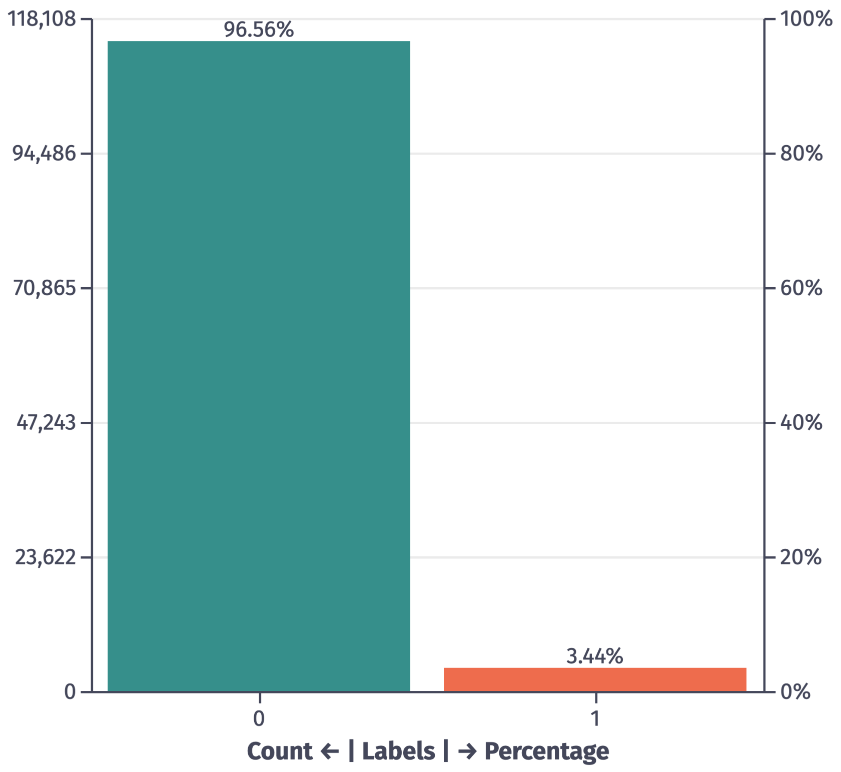


Figure 4.1: Label bar chart for the (test) dataset used. The label *1* corresponds to the fraudulent class. It is possible to quickly verify the significant imbalance between classes. The prevalence of fraud is less than 5%. The script to generate this chart can be found in Appendix O.

scientists create charts with the same style.

- For two of the data scientists, the threshold areas of the score distribution chart are useful for defining alert and automatic decline strategies (although they can be difficult to explain to clients), for example.
- For three of the data scientists, the FI comparison is interesting/useful and could be an alternative to the manual process performed using spreadsheets.
- For one of the data scientists, being able to generate score distribution charts with several bin widths at once is important.
- For one of the data scientists, the panel for selecting a zone of interest in the temporal chart is interesting, not only for the interactivity but also for the preview it shows.

Out of curiosity, it is interesting to note that the various data scientists point out and comment on different things (not just in these interviews, but in those before development as well), in more or less detail. This arguably shows the disparity of needs and interests that exist, possibly influenced by the variability of projects and backgrounds, even if the main tasks that they have to complete are, at the Model Evaluation level, very similar (at least within each use case). On the other hand, data scientists also shared a set of ideas full of potential that, due to the project timeline, could not be implemented and tested immediately, thus inspiring some of the directions presented in the Future Work and Conclusion section.

Besides this round of user interviews, the first version of MevaL was also presented at the weekly event that takes place at Feedzai dedicated to DS. In addition to the opportunity to show and publicize the package, the discussion allowed for the collection of extra ideas to keep in mind, although it focused mainly on data processing and its importance for the usability of a tool like MevaL.

4.3 Internal Projects

After the release of the first full working version of MevaL (in August 2020), teams from two internal projects at Feedzai started using it for two different purposes (in fact, there is a first version of each project already available). Both projects are producing reports for the evaluation and comparison of ML models, but with different objectives and designs. In addition to the value added by MevaL, as a kind of component library for automatic generation of reports for/with Model Evaluation (or similar), these collaborations also allowed to improve some aspects of the current version of the package, as well as to discuss possible new features for a more complete and robust version in the future.

The project descriptions below are based on two interviews conducted in order to ensure the accuracy of the information regarding different aspects of each of the projects (the guidelines can be found in Appendix R).

4.3.1 Automatic Model Retraining Report

The first project is called "Automatic Model Retraining Report" and, in general, it boils down to a two-way model comparison report. In simple words, this report, generated from a Jupyter notebook, allows comparing two models and was designed mainly to facilitate the comparison between a model in production and a new model (although it is possible to compare any two models). The new model can be a retrained model, that is, a model developed from scratch (in the sense of using new features, different hyperparameters, among other possible choices), or a refreshed model, i.e., a model equivalent to the one in production but trained with the latest data.

Prior to this project, this type of comparison was conducted only through one-time computational notebooks/scripts developed individually for each of the projects that required it (there was no centralized tooling for this specific purpose). So, the motivation for this project derives from the need and interest in saving development time and in unifying not only the look and feel of a front-facing actionable document, but also the design of the charts and the technologies used to make this possible. Simply put, the main goal of this report is to streamline the decision-making process regarding the replacement of a given model in production with a new model, in a quick and informed way.

In addition to the motivation shared above, this project was also pushed forward as it fills a gap previously identified for one of Feedzai's clients: Lloyds Banking Group, a major British financial institution.

As for the report itself, it consists of dynamic text/tables and charts (powered by MevaL) interspersed on a Jupyter notebook that is later converted to a shareable HTML file (main output). This report also has a data layer with a (fast even for large datasets) raw data processing pipeline, in order to facilitate the preparation of the necessary datasets for performance verification and model comparison. In the simplest case, a data scientist only needs to indicate the data sources and the appropriate parameters, as the report will be generated automatically from this input, without the need for a *one-time* custom computational notebook/script (the report is flexible and general enough for different DS projects).

Moreover, this report has two main target groups: data scientists (who can use both formats) and Feedzai's clients (who can freely check the HTML report). For clients, a report like this one is a good way to show some evidence that justifies or not an update of a model in production, for example. On the other hand, data scientists (from a developer and end user perspective), can customize the report, either in terms of cells/content, or in terms of parameters for the different functions and charts, for example, while they can use it for their daily work as well.

As for the design goals, it is intended that this report follows a nice and clean interface, where aesthetics and professionalism are visible throughout the provision of information considered relevant. Reusability, on the other hand, is a design principle of this report, both in terms of what it uses and what it provides. In this way, in terms of implementation, this project takes advantage of some of Feedzai's in-house packages to assemble the proposed solution.

As for MevaL and its visual part, the report uses the ROC/PR curve chart, the gain chart, the (colored) breakdown strip chart (which they found very informative), the confusion matrix, the temporal (Model Decay) chart, and the (faceted) score distribution chart. According to the project team, MevaL was chosen and *reused* for five reasons: (1) it provides the desired charts; (2) it supports breakdowns; (3) it is aesthetically appealing; (4) it supports interactivity (such as tooltips, for example); (5) it offers a series of options that allow customization of the ready-to-use charts.

4.3.2 Automatic Model Governance Report

The second internal project that uses MevaL is called "Automatic Model Governance Report" and, in a nutshell, consists of an end-to-end report for a given model, from the data used to its detailed results, and it is intended to meet compliance requirements. Therefore, this template-oriented project is broader than the previous one and MevaL only covers a specific part in terms of Model Evaluation. This report thus contains a section for a new model, focused on performance and where a comparison is made with the current model in production — MevaL joins here with its charts to facilitate the sharing of information about model performance.

Previously, specifically for one of Feedzai's clients, a Model Governance report was prepared manually using a word processor. However, for each written report, most of the time was spent in the proper formatting of the document and its components (like the style for the charts) for the sake of consistency. Thus, the idea arose to develop a template-like document that allows to create one of these reports quickly, freeing the data scientist's time to prepare, exclusively, the content and data, while the look and feel would be assured. Thus, the main success criterion for this work is to achieve a significant difference in the time needed to prepare a report compared to the previous support.

As for the report itself, it consists of a mixture of text, tables and charts mounted on a Jupyter notebook (enriched with custom CSS) that is later converted, using the nbconvert package [126], to PDF format. Regarding the target group, this (PDF) report is designed mainly for clients, although data scientists can also avail oneself of the computational notebook version. In the simplest case, to generate a Model Governance report, it is only necessary to specify a dataset with a predefined schema (and some variable-like parameters) and run the notebook.

In addition, in terms of implementation, this project takes advantage of some of Feedzai's in-house packages to assemble the proposed solution as well. MevaL was one of the tools chosen for four reasons in particular: (1) it yields the desired charts; (2) it provides consistently designed charts and a standard aesthetic; (3) it is easy to use; (4) it allows to (directly) embed the charts in an HTML file (although this is simply thanks to Altair). As for MevaL and its visual part, the report uses the ROC curve chart, the Gain chart, the classification threshold chart, the temporal (Model Decay) chart, and the (grouped) score distribution chart.

Finally, as a side note, one of the team members of this project highlighted the showcase notebook available in the MevaL repository as it helped to start using the package (having a support to show the charts, as a kind of visual documentation, is useful and makes it easy to pick up the charts).

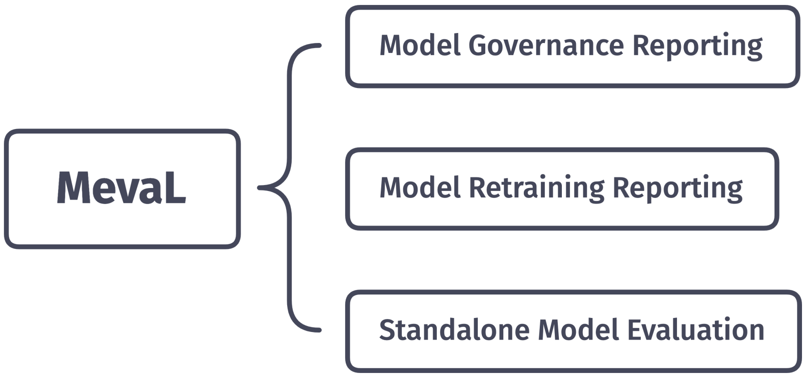

MevaL
Model Governance Reporting
Model Retraining Reporting
Standalone Model Evaluation

Figure 4.2: MevaL as a Python package serving different end goals.

Chapter 5

Future Work and Conclusion

MevaL is a Python package for visual ML Model Evaluation integrated into Feedzai's DS environment, developed with the help of data scientists and data visualization engineers, and already supporting two internal client-facing reporting projects. The first version is up and running, but there are a couple of directions already identified that could make MevaL a more powerful and robust tool in subsequent versions.

The first action point, with an action plan until January 2021, is to finish the transfer to production, as described in the Deployment section, and update the charts with the latest version from Altair (4.1.0). In addition to enabling any data scientist to use MevaL and its charts OOTB at Feedzai, this unification will allow for the collection of more feedback coming from its (expected) role in DS projects during Model Evaluation.

MevaL cannot be a static tool for ever-changing needs and scenarios. Thus, in addition to the well-defined previous point, there are, at this moment, other (research/experimentation) directions with the potential to derive new features and know-how for MevaL and for Model Evaluation as a fundamental procedure.

In terms of features, in addition to improvements to the current charts with new options and new custom themes (like a theme with a larger font size for presentation purposes), for example, the next step would be to natively support multiple models *in situ*. In other words, the main idea is to extend the plotting API for relevant charts, such as the curve chart, the temporal chart, the breakdown strip chart, and the classification threshold chart, so that, in the same chart and in the same space, the results of multiple models are visually represented in line with all the customization options already available. Another option would be to include statistical tests, like the McNemar's test and Cochran's Q test (both tests consider the same test set), to help select models. Besides that, a chart, like the one used for Skyline Queries [293], for AutoML model selection is an option to explore as well (AutoML generates many different models, so this chart could help the data scientist choose a smaller subsample based on two performance metrics, for example).

In terms of uncertainty-based features, it would be positive to integrate new estimation mechanisms, such as the calculation of confidence intervals using bootstrapping methods [245], and to add the option of including confidence bands in ROC curve charts (vertical bands in terms of Recall), for example.

Plus, there are totally new features for MevaL that are also quite promising to amplify the package. Taking the tasks described in the Model Evaluation Topology section, it would be interesting to investigate and test new charts for FP/FN Analysis, and new functions for Feature Reduction.

For the first task (FP/FN Analysis), in addition to a data layer that would allow the use of the original dataset(s) enriched with the results of one (or more) ML model(s), a possible idea would be the geospatial analysis of predicted labels through an interactive map (assuming that a given dataset would contain geospatial features). In this interactive map, the instances would be encoded as points, for example, and each point would be colored in order to distinguish the correctly classified instances from the incorrectly classified ones, the correctly classified instances from the FPs and FNs, or all confusion categories. Another option would be to support different color scales, as this map could be used to see the geographical distribution of the (continuous) scores. The main idea is to facilitate the inspection of the instances from a geospatial point of view and help the data scientist to find potentially interesting patterns in order to identify certain data-based problems or to create new features, for example (this idea is based on Manifold's Geo Feature View [168] and could be realized with the Folium package [84], for example). Another idea would be to include correctness validation in MevaL, a method composed of the following steps: (1) convert the predicted labels to 0 or 1 (for example) depending on whether the predicted labels are correct or not, respectively (label 1 may also consist of FPs or FNs only, while the remaining instances are labeled as 0); (2) train a new model (a RF, for example) using the new labels as "ground truth"; (3) compute the FI values/ranking; (4) analyze the most important features.

For the second task (Feature Reduction), using the existing charts, it would be interesting to *connect* MevaL to Pulse (Feedzai's e2e ML platform) and, with a function call, allow the data scientist to apply a Feature Reduction method and then inspect the results obtained with a subset of features. A simple method for this would be the one-shot feature elimination, a method consisting of the following steps (the main disadvantage is the computational cost for training new models): (1) compute the FI values/ranking; (2) select a number of the best features (or select a number of the less important features to remove); (3) train and evaluate a new model with this feature subset; (4) compare performance.

In terms of testing, both from a package and user perspective, there are two aspects to explore in the future. The first is visual regression testing, a testing method based on screenshots and their comparison at different times with reference ones, to complement unit testing. As a visual-dependent package, this dimension would be central to ensuring (or to be even closer to guaranteeing) that the charts, as an image, are really generated as expected (this could be implemented by adapting the pytest [149] and pytest-selenium [118] packages, for example). The second is the design and execution of a formal quantitative evaluation with data scientists (where a series of metrics, such as success rate, mouse metrics, and task duration, would be collected during the resolution of different tasks and then analyzed, for example). This evaluation was not carried out due to the lack of a benchmark for comparison (in this way, it could also serve to establish an in-house benchmark in the future), the package development process (an exploratory and iterative process based on qualitative feedback), and the commitment to deploy MevaL (considered a priority).

If possible in the future, there are some Jupyter-based technologies (which are not currently supported in Feedzai's DS environment) that can be used to implement certain mechanisms in sync with MevaL and Altair. An example is the ipywidgets package [125], a package with several interactive HTML-based widgets, such as sliders, to explore and exploit interaction even more (a slider could allow the data scientist to easily change the classification threshold and see the impact on charts parameterized by this threshold, for example). Alternatively, given that JupyterLab is at its core an extensible platform, it is also possible to consider (almost) everything from small changes to the interface to serve specific purposes (like the custom colors for the input and output collapsers to the left of the cells, depending on whether it is recommended or unsafe to rerun a given cell in terms of implicit dependencies, implemented in NBSafety [177], a Jupyter kernel with visual functionalities to assist in reasoning about out-of-order cell executions) to full extensions that help in Model Evaluation. On the other hand, mainly for a first-time Model Evaluation, MevaL may, in the future, have an automatic computational notebook (with some metadata in the header, such as the information provided by the watermark package [243], for example) in which the data scientist will only have to define some arguments (data sources and some global variables) and will have access to a notebook with a series of charts (and potentially interactive widgets) ready to use.

Nevertheless, although there are many directions to explore, the first version of MevaL (and this work) can be considered a success, not only for the Python package developed and made available internally, but also for the feedback collected and the organic way it was adopted in two internal projects at Feedzai.

That said, to conclude this work, it remains to look at the success criteria defined in the first chapter (Success Criteria section):

- The Python package enables the exploration of the created models from at least a new perspective that is not currently available OOTB in Pulse.
  - Comparing the Feedzai section and the Features section, it is possible to check that MevaL onboards other tasks that are not possible to accomplish using

Pulse, as is the case for Model Decay, and complements others with complementary information and new charts, such as the binomial proportion confidence intervals and the breakdown strip chart. Thus, this criterion was met.

- At least two action points of each of the defined and selected tasks/requirements are implemented and ready to use.
  - Scanning the Features section, it is clear that each of the tasks selected for the first version of MevaL has at least one chart with multiple options (derived from two or more action points), offering, in practice, an extended set of possible combinations to support a Model Evaluation procedure. In addition, supporting each of the tasks, there is an efficient data processing pipeline that allows the data scientist to have access to the ready-to-view datasets. Thus, this criterion was met.
- All small, low-effort action points related to improvements of what Pulse currently offers in terms of DV for Model Evaluation are implemented (if the charts in question are implemented in the Python package).
  - In addition to MevaL providing an improved version of each of the charts available on Pulse, it also accommodates a flexible API that allows to address some of the identified needs, such as finer granularity for the bin widths of the score distribution chart and full-fledged *breakdown* charts, as is the case for curve charts, for example. Thus, this criterion was met.
- The Python package receives approval from the Product and Customer Success departments.
  - Complementing the transfer to production (Deployment section), in collaboration with the Product department, and the two internal reporting projects (Internal Projects section), in collaboration with the Customer Success department, the overall feedback received from the various stakeholders is mostly positive. Thus, this criterion was met.
- The Python package is tested and validated by Feedzai's data scientists.
  - Although this aspect is supported only by a qualitative component, MevaL was developed iteratively with the support of data scientists (and data visualization engineers) and constantly challenged, culminating in a set of formal user interviews and as one of the tools used by the teams of two internal projects at Feedzai. Thus, this criterion was met (although there is room for improvement in the future, as described in this chapter).
- All information about this project is documented in a central repository at Feedzai.

  - All materials referring to MevaL and this project in a more general way are neatly hosted in a (Git) repository and in a folder on Feedzai's cloud storage service. These materials are accessible to everyone at Feedzai. Thus, this criterion was met.

- The Python package features a modular architecture so that it is possible to add new functionalities efficiently.

  - Based on the Package Architecture and Features sections, MevaL has a data layer separate from the plotting layer, and each component/chart can be used starting from the *EvaluationManager* class or individually, according to some considerations. It is quite easy and intuitive to add new charts (just create a new file/folder with a class that inherits one of the abstract classes, if necessary, and follows some guidelines, as well as the respective function to generate the chart using Altair) or new performance metrics (just add a function with the necessary calculations from the pandas DataFrame columns for each confusion category and add it to a central Python dictionary), for example, in addition to the fact that the various parts of the package are documented and have well-defined purposes. Thus, this criterion was met.

- The Python package is complemented with a set of tutorials and documentation.

  - The package contains docstrings and comments aimed at helping data scientists and future developers, as well as a showcase notebook to introduce MevaL and its features. In addition, several documents were prepared to describe all phases of this project, such as documents with guidelines for the interviews, documents with the summaries and conclusions of the interviews, documents with the description and justification of the content to be implemented, among others. Recordings of presentations regarding the project and MevaL are also available for everyone at Feedzai. Thus, this criterion was met.

- The Python package is promoted internally on at least two occasions.

  - Briefly, MevaL and this project were presented on four separate occasions at Feedzai, twice at the weekly event of the Research department and twice at the weekly event dedicated to DS. Thus, this criterion was met.

- The Python package is promoted externally on at least one occasion.

  - This thesis, written in as much detail as possible, is the main vehicle for external promotion (since it will be publicly available). Furthermore, in the near future, the writing of a blog post about MevaL is also planned. Thus, this criterion was met.

Bibliography

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng. "TensorFlow: A system for large-scale machine learning." In: *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 2016, pp. 265–283. URL: https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf.

[2] A. Abdallah, M. A. Maarof, and A. Zainal. "Fraud detection system: A survey." In: *Journal of Network and Computer Applications* 68 (2016), pp. 90–113. ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2016.04.007. URL: http://www.sciencedirect.com/science/article/pii/S1084804516300571.

[3] Y. Ahn and Y.-R. Lin. "FairSight: Visual Analytics for Fairness in Decision Making." In: *IEEE Transactions on Visualization and Computer Graphics* (2019), 1–1. ISSN: 2160-9306. DOI: 10.1109/tvcg.2019.2934262. URL: http://dx.doi.org/10.1109/TVCG.2019.2934262.

[4] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama. "Optuna: A Next-generation Hyperparameter Optimization Framework." In: *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2019.

[5] Ángel Alexander Cabrera, W. Epperson, F. Hohman, M. Kahng, J. Morgenstern, and D. H. Chau. *FairVis: Visual Analytics for Discovering Intersectional Bias in Machine Learning*. 2019. arXiv: 1904.05419 [cs.LG].

[6] M. Ali. *PyCaret: An open source, low-code machine learning library in Python*. PyCaret version 2.2. Apr. 2020. URL: https://www.pycaret.org.

[7] V. Allen. *Titanic - Guided by a confusion matrix*. Kaggle notebook. Version 16. [Online; accessed 30-November-2020]. 2016. URL: https://www.kaggle.com/vinceallenvince/titanic-guided-by-a-confusion-matrix.

[8] B. Alsallakh, A. Hanbury, H. Hauser, S. Miksch, and A. Rauber. "Visual Methods for Analyzing Probabilistic Classification Data." In: *IEEE Transactions on Visualization and Computer Graphics* 20.12 (Dec. 2014), pp. 1703–1712. ISSN: 2160-9306. DOI: 10.1109/TVCG.2014.2346660.

[9] Alteryx, Inc. *EvalML*. https://github.com/alteryx/evalml. 2019.

[10] Amazon.com, Inc. *Deep Java Library (DJL)*. Version 0.8.0. Apache Software Foundation License 2.0. Sept. 22, 2020. URL: https://javadoc.djl.ai/.

[11] amCharts. *DataViz Tip #13: Switch To Horizontal Bar Chart When Labels Don't Fit*. [Online; accessed 15-November-2020]. Jan. 2018. URL: https://www.amcharts.com/dataviz-tip-13-switch-horizontal-bar-chart-labels-dont-fit/.

[12] S. Amershi, M. Chickering, S. M. Drucker, B. Lee, P. Simard, and J. Suh. "ModelTracker: Redesigning Performance Analysis Tools for Machine Learning." In: *Proceedings of the Conference on Human Factors in Computing Systems (CHI 2015)*. ACM - Association for Computing Machinery, Apr. 2015. URL: https://www.microsoft.com/en-us/research/publication/modeltracker-redesigning-performance-analysis-tools-for-machine-learning/.

[13] Anaconda. *2020 State of Data Science*. Tech. rep. 2020. URL: https://www.anaconda.com/state-of-data-science-2020.

[14] B. Andersson and Media Pop. *datetime_truncate*. https://github.com/mediapop/datetime_truncate. 2013.

[15] S. André, I. Cordasco, and Python Code Quality Authority. *flake8-docstrings*. https://gitlab.com/pycqa/flake8-docstrings. 2013.

[16] Anonymous (https://meta.stackoverflow.com/users/61574/anonymous). *Formula to determine brightness of RGB color*. Stack Overflow. https://stackoverflow.com/q/596216 (version: 2009-02-27). eprint: https://stackoverflow.com/q/596216. URL: https://stackoverflow.com/a/596243.

[17] D. Anson. *markdownlint*. https://github.com/DavidAnson/vscode-markdownlint. 2015.

[18] Apache Software Foundation. *Apache Ignite*. Version 2.9.0. Oct. 21, 2020. URL: https://ignite.apache.org/.

[19] Apache Software Foundation. *Useful Developer Tools*. [Online; accessed 27-November-2020]. 2020. URL: https://spark.apache.org/developer-tools.html.

[20] Apple Inc. *MacBook Pro (13-inch, 2017, Two Thunderbolt 3 ports) - Technical Specifications*. [Online; accessed 9-November-2020]. Aug. 2020. URL: https://support.apple.com/kb/SP754.

[21] S. Arlot and A. Celisse. "A survey of cross-validation procedures for model selection." In: *Statistics Surveys* 4.0 (2010), 40–79. ISSN: 1935-7516. DOI: 10.1214/09-ss054. URL: http://dx.doi.org/10.1214/09-SS054.

[22] Association for Financial Professionals and J.P. Morgan. *2018 AFP® Payments Fraud and Control Survey: Key Highlights*. Tech. rep. 2018. URL: https://commercial.jpmorganchase.com/jpmpdf/1320745402134.pdf.

[23] Association for Financial Professionals and J.P. Morgan. *2020 AFP® Payments Fraud and Control Survey: Key Highlights*. Tech. rep. 2020. URL: https://www.afponline.org/docs/default-source/registered/2020paymentsfraudandcontrolreport-highlights-final.pdf.

[24] Association of Certified Fraud Examiners. *2020 Report to the Nations: Global Study on Occupational Fraud and Abuse*. Tech. rep. 2020. URL: https://www.acfe.com/report-to-the-nations/2020/.

[25] N. Babich. *The Art of the User Interview*. [Online; accessed 15-October-2020]. 2017. URL: https://medium.springboard.com/the-art-of-the-user-interview-cf40d1ca62e8.

[26] K. Basques. *Run Commands With The Chrome DevTools Command Menu*. [Online; accessed 30-October-2020]. 2020. URL: https://developers.google.com/web/tools/chrome-devtools/command-menu.

[27] K. Basques. *Simulate Mobile Devices with Device Mode in Chrome DevTools*. [Online; accessed 30-October-2020]. 2020. URL: https://developers.google.com/web/tools/chrome-devtools/device-mode.

[28] B. G. Becker. *Research Report: Visualizing Decision Table Classifiers*. 1998.

[29] P. Bell. *How Pew Research Center uses small multiple charts*. [Online; accessed 22-October-2020]. 2018. URL: https://medium.com/pew-research-center-decoded/how-pew-research-center-uses-small-multiple-charts-2531bfc06419.

[30] B. Bengfort and R. Bilbro. "Yellowbrick: Visualizing the Scikit-Learn Model Selection Process." In: *Journal of Open Source Software* 4.35 (2019), p. 1075. DOI: 10.21105/joss.01075. URL: https://doi.org/10.21105/joss.01075.

[31] M. R. Berthold, N. Cebron, F. Dill, T. R. Gabriel, T. Kötter, T. Meinl, P. Ohl, C. Sieb, K. Thiel, and B. Wiswedel. "KNIME: The Konstanz Information Miner." In: *Studies in Classification, Data Analysis, and Knowledge Organization (GfKL 2007)*. Springer, 2007. ISBN: 978-3-540-78239-1.

[32] F. Bertrand. *Sweetviz*. https://github.com/fbdesignpro/sweetviz. 2020.

[33] P. Biecek. *ceterisParibus: Ceteris Paribus Profiles*. R package version 0.3.1. 2019. URL: https://CRAN.R-project.org/package=ceterisParibus.

[34] B. Bischl, M. Lang, L. Kotthoff, J. Schiffner, J. Richter, E. Studerus, G. Casalicchio, and Z. M. Jones. "mlr: Machine Learning in R." In: *Journal of Machine Learning Research* 17.170 (2016), pp. 1–5. URL: https://jmlr.org/papers/v17/15-066.html.

[35] W. Bittremieux. "spectrum_utils: A Python Package for Mass Spectrometry Data Processing and Visualization." In: *Analytical Chemistry* 92.1 (2020). PMID: 31809021, pp. 659–661. DOI: 10.1021/acs.analchem.9b04884. eprint: https://doi.org/10.1021/acs.analchem.9b04884. URL: https://doi.org/10.1021/acs.analchem.9b04884.

[36] B. Boehmke and B. M. Greenwell. *Hands-On Machine Learning with R*. First edition. Chapman and Hall/CRC, 2019. ISBN: 978-1138495685.

[37] B. Boehmke. *Cleveland Dot Plots (UC Business Analytics R Programming Guide)*. [Online; accessed 28-November-2020]. 2016. URL: http://uc-r.github.io/cleveland-dot-plots.

[38] M. Bostock, V. Ogievetsky, and J. Heer. "D3: Data-Driven Documents." In: *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis)* (2011). URL: http://idl.cs.washington.edu/papers/d3.

[39] B. Branco, P. Abreu, A. S. Gomes, M. S. C. Almeida, J. T. Ascensão, and P. Bizarro. "Interleaved Sequence RNNs for Fraud Detection." In: *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (Aug. 2020). DOI: 10.1145/3394486.3403361. URL: http://dx.doi.org/10.1145/3394486.3403361.

[40] D. Brickley and L. Miller. *FOAF Vocabulary Specification 0.99*. [Online; accessed 24-November-2020]. 2014. URL: http://xmlns.com/foaf/spec/.

[41] M. Brondbjerg and City Intelligence. *City Intelligence Data Design Guidelines*. [Online; accessed 1-November-2020]. 2019. URL: https://data.london.gov.uk/dataset/city-intelligence-data-design-guidelines.

[42] J. Brownlee. *How to Calculate Feature Importance With Python*. [Online; accessed 8-November-2020]. 2020. URL: https://machinelearningmastery.com/calculate-feature-importance-with-python/.

[43] S. Brugman. *pandas-profiling: Exploratory Data Analysis for Python*. https://github.com/pandas-profiling/pandas-profiling. Version: 2.9.0, Accessed: 18-November-2020. 2019.

[44] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux. "API design for machine learning software: experiences from the scikit-learn project." In: *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*. 2013, pp. 108–122.

[45] A. Burkov. *The Hundred-Page Machine Learning Book*. First edition. Andriy Burkov, 2019. ISBN: 978-1999579500.

[46] L. Cappelletti. *sanitize_ml_labels*. https://github.com/LucaCappelletti94/sanitize_ml_labels. 2019.

[47] F. Carcillo, Y.-A. Le Borgne, O. Caelen, and G. Bontempi. "Streaming active learning strategies for real-life credit card fraud detection: assessment and visualization." In: *International Journal of Data Science and Analytics* 5.4 (Apr. 2018), 285–300. ISSN: 2364-4168. DOI: 10.1007/s41060-018-0116-z. URL: http://dx.doi.org/10.1007/s41060-018-0116-z.

[48] T. D. V. Catalogue. *Pie Charts*. [Online; accessed 1-October-2020]. URL: https://datavizcatalogue.com/methods/pie_chart.html.

[49] T. Chen and C. Guestrin. "XGBoost." In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '16* (2016). DOI: 10.1145/2939672.2939785. URL: http://dx.doi.org/10.1145/2939672.2939785.

[50] D. Chicco and G. Jurman. "The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation." In: *BMC Genomics* 21.1 (Jan. 2020), p. 6. ISSN: 1471-2164. DOI: 10.1186/s12864-019-6413-7. URL: https://doi.org/10.1186/s12864-019-6413-7.

[51] M. Conlen. *The Value of Visualization*. [Online; accessed 29-September-2020]. 2020. URL: https://courses.cs.washington.edu/courses/cse442/20wi/lectures/CSE442-ValueOfVisualization.pdf.

[52] T. igraph Core Team. *igraph*. Version 0.8.3. Oct. 2020. DOI: 10.5281/zenodo.4063562. URL: https://doi.org/10.5281/zenodo.4063562.

[53] A. Cotgreave. *Stephen Few's wrapped bars: thoughts*. [Online; accessed 26-November-2020]. 2013. URL: https://gravyanecdote.com/uncategorized/wrappedbarsthoughts/.

[54] C. Coyier. *When to Use SVG vs. When to Use Canvas*. [Online; accessed 1-November-2020]. 2019. URL: https://css-tricks.com/when-to-use-svg-vs-when-to-use-canvas/.

[55] T. Crosley and Python Code Quality Authority. *isort*. https://github.com/PyCQA/isort. 2013.

[56] J. Czakon. *24 Evaluation Metrics for Binary Classification (And When to Use Them)*. [Online; accessed 28-October-2020]. 2019. URL: https://neptune.ai/blog/evaluation-metrics-binary-classification.

[57] J. Czakon. *F1 Score vs ROC AUC vs Accuracy vs PR AUC: Which Evaluation Metric Should You Choose?* [Online; accessed 30-November-2020]. 2019. URL: https://neptune.ai/blog/f1-score-accuracy-roc-auc-pr-auc.

[58] d3r3kx14o, Lcorvle, C. Chen, and TangLin12. *BOOSTVis2*. https://github.com/TangLin12/BOOSTVis2. 2017.

[59] A. Dal Pozzolo, G. Boracchi, O. Caelen, C. Alippi, and G. Bontempi. “Credit card fraud detection and concept-drift adaptation with delayed supervised information.” In: *2015 International Joint Conference on Neural Networks (IJCNN)*. 2015, pp. 1–8. DOI: 10.1109/IJCNN.2015.7280527.

[60] J. Damiba, N. Kruchten, and Plotly Technologies Inc. *Network Graphs Comparison in Python/v3*. [Online; accessed 24-November-2020]. 2020. URL: https://plotly.com/python/v3/igraph-networkx-comparison/.

[61] Dataiku. *Dataiku Data Science Studio (Community Edition)*. Version 8.0.2. Sept. 23, 2020. URL: https://www.dataiku.com/.

[62] Datawrapper. *How to create a range plot*. [Online; accessed 28-November-2020]. 2020. URL: https://academy.datawrapper.de/article/111-how-to-create-a-range-plot.

[63] Datawrapper. *How to create an arrow plot*. [Online; accessed 28-November-2020]. 2020. URL: https://academy.datawrapper.de/article/123-how-to-create-an-arrow-plot.

[64] J. Demšar. “Statistical Comparisons of Classifiers over Multiple Data Sets.” In: *J. Mach. Learn. Res.* 7 (Dec. 2006), 1–30. ISSN: 1532-4435.

[65] J. Demšar, T. Curk, A. Erjavec, Črt Gorup, T. Hočevar, M. Milutinovič, M. Možina, M. Polajnar, M. Toplak, A. Starič, M. Štajdohar, L. Umek, L. Žagar, J. Žbontar, M. Žitnik, and B. Zupan. “Orange: Data Mining Toolbox in Python.” In: *Journal of Machine Learning Research* 14 (2013), pp. 2349–2353. URL: http://jmlr.org/papers/v14/demsar13a.html.

[66] C. Deotte. *Feature Engineering Techniques*. [Online; accessed 30-October-2020]. 2019. URL: https://www.kaggle.com/c/ieee-fraud-detection/discussion/108575.

[67] scikit-learn developers. *3.1. Cross-validation: evaluating estimator performance*. [Online; accessed 28-September-2020]. URL: https://scikit-learn.org/stable/modules/cross_validation.html.

[68] scikit-learn developers. *3.3. Metrics and scoring: quantifying the quality of predictions*. [Online; accessed 28-September-2020]. URL: https://scikit-learn.org/stable/modules/model_evaluation.html.

[69] S. Doody. *Starter Questions for User Research*. [Online; accessed 15-October-2020]. 2016. URL: http://projects.iq.harvard.edu/files/harvarduxgroup/files/ux-research-guide-sample-questions-for-user-interviews.pdf.

[70] DrivenData. *Cookiecutter Data Science*. https://github.com/drivendata/cookiecutter-data-science. 2015.

[71] DrivenData. *deon*. https://github.com/drivendataorg/deon. 2018.

[72] Eclipse Deeplearning4j Development Team. *Deeplearning4j: Open-source distributed deep learning for the JVM*. Version 1.0.0-beta7. Apache Software Foundation License 2.0. May 13, 2020. URL: https://deeplearning4j.org/.

[73] D. Eugénio. *What Is Machine Learning Builder and What Problems Does It Solve?* [Online; accessed 28-November-2020]. 2020. URL: https://www.outsystems.com/blog/posts/what-is-machine-learning-builder/.

[74] Europol. *Economic Crime*. [Online; accessed 24-September-2020]. URL: https://www.europol.europa.eu/crime-areas-and-trends/crime-areas/economic-crime.

[75] T. Fawcett. "An Introduction to ROC Analysis." In: *Pattern Recogn. Lett.* 27.8 (June 2006), 861–874. ISSN: 0167-8655. DOI: 10.1016/j.patrec.2005.10.010. URL: https://doi.org/10.1016/j.patrec.2005.10.010.

[76] Feedzai. *About Us*. [Online; accessed 29-September-2020]. URL: https://feedzai.com/about-us/.

[77] Feedzai. *One Platform to Manage Financial Crime*. [Online; accessed 29-September-2020]. URL: https://feedzai.com/feedzai-platform/.

[78] Feedzai. *The Feedzai Code Book*. Tech. rep. URL: https://feedzai.com/feedzai-code-book/.

[79] Feedzai. *The Partner Program*. [Online; accessed 29-September-2020]. URL: https://feedzai.com/partners/.

[80] Feedzai. *Fighting Account Opening and Application Fraud with Machine Learning: A Blueprint for Retail Banks*. Tech. rep. 2018. URL: https://feedzai.com/resources/wp/fighting-account-opening-fraud-with-machine-learning-digital.

[81] ferdio. *Lollipop Chart*. [Online; accessed 26-November-2020]. 2017. URL: https://datavizproject.com/data-type/lollipop-chart/.

[82] ferdio. *Scaled-up Number*. [Online; accessed 26-November-2020]. 2017. URL: https://datavizproject.com/data-type/number/.

[83] FICO. *Fraud Transaction Monitoring*. [Online; accessed 25-September-2020]. URL: https://www.fico.com/en/solutions/fraud-transaction-monitoring.

[84] Filipe, M. Journois, Frank, R. Story, J. Gardiner, H. Rump, A. Bird, A. Lima, J. Cano, dbf, J. Leonel, T. Sampson, J. Baker, B. Welsh, J. Reades, O. Komarov, Q. Kong, odovad, R. Dumas, G. Harris, A. Crosby, L. Furtado, kenmatsu4, T. P. Nogueira, N. Wilson, penguindustin, D. Kato, R. Signell, J. Duke, and A. Patil. *python-visualization/folium: v0.11.0*. Version v0.11.0. May 2020. DOI: 10.5281/zenodo.3806268. URL: https://doi.org/10.5281/zenodo.3806268.

[85] J. H. Friedman. “Greedy Function Approximation: A Gradient Boosting Machine.” In: *The Annals of Statistics* 29.5 (2001), pp. 1189–1232. ISSN: 905364. URL: http://www.jstor.org/stable/2699986.

[86] Gacek (https://meta.stackoverflow.com/users/177167/gacek). *Determine font color based on background color*. Stack Overflow. https://stackoverflow.com/q/1855884 (version: 2012-01-09). eprint: https://stackoverflow.com/q/1855884. URL: https://stackoverflow.com/a/1855903.

[87] L. Gatto. *An Introduction to Machine Learning with R*. [Online; accessed 28-September-2020]. 2019. URL: https://lgatto.github.io/IntroMachineLearningWithR/index.html.

[88] D. Godoy. *HandySpark*. https://github.com/dvgodoy/handyspark. 2018.

[89] O. Gomez, S. Holter, J. Yuan, and E. Bertini. *ViCE: Visual Counterfactual Explanations for Machine Learning Models*. 2020. arXiv: 2003.02428 [cs.HC].

[90] Google. *Data visualization*. Material. [Online; accessed 1-November-2020]. Nov. 2019. URL: https://material.io/design/communication/data-visualization.html.

[91] M. Gorelli and G. Pasupathy. *nbQA*. https://github.com/nbQA-dev/nbQA. 2020.

[92] J. Grau, I. Grosse, and J. Keilwagen. “PRROC: computing and visualizing precision-recall and receiver operating characteristic curves in R.” In: *Bioinformatics* 31.15 (Mar. 2015), pp. 2595–2597. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btv153. eprint: https://academic.oup.com/bioinformatics/article-pdf/31/15/2595/5026174/btv153.pdf. URL: https://doi.org/10.1093/bioinformatics/btv153.

[93] H2O.ai. *h2o: Python Interface for H2O*. Python package version 3.30.0.6. 2020. URL: https://github.com/h2oai/h2o-3.

[94] H2O.ai. *H2O: Scalable Machine Learning Platform*. version 3.30.0.6. 2020. URL: https://github.com/h2oai/h2o-3.

[95] A. A. Hagberg, D. A. Schult, and P. J. Swart. “Exploring Network Structure, Dynamics, and Function using NetworkX.” In: *Proceedings of the 7th Python in Science Conference*. Ed. by G. Varoquaux, T. Vaught, and J. Millman. Pasadena, CA USA, 2008, pp. 11–15.

[96] S. Haghighi, M. Jasemi, S. Hessabi, and A. Zolanvari. “PyCM: Multiclass confusion matrix library in Python.” In: *Journal of Open Source Software* 3.25 (May 2018), p. 729. DOI: 10.21105/joss.00729. URL: https://doi.org/10.21105/joss.00729.

[97] H. Hammond. *PrettyPandas*. https://github.com/HHammond/PrettyPandas. 2016.

[98] B. Hanczar, J. Hua, C. Sima, J. Weinstein, M. Bittner, and E. R. Dougherty. "Small-sample precision of ROC-related estimates." In: *Bioinformatics* 26.6 (Feb. 2010), pp. 822–830. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btq037. eprint: https://academic.oup.com/bioinformatics/article-pdf/26/6/822/676553/btq037.pdf. URL: https://doi.org/10.1093/bioinformatics/btq037.

[99] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant. "Array programming with NumPy." In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[100] L. Haviland and DataRobot, Inc. *Comparing Models Overview*. [Online; accessed 29-November-2020]. 2020. URL: https://community.datarobot.com/t5/resources/comparing-models-overview/ta-p/1708.

[101] L. Haviland and E. W. DataRobot, Inc. *Describing and Evaluating Models*. [Online; accessed 29-November-2020]. 2020. URL: https://community.datarobot.com/t5/resources/describing-and-evaluating-models/ta-p/1491.

[102] C. Hayashi. "What is Data Science? Fundamental Concepts and a Heuristic Example." In: *Data Science, Classification, and Related Methods*. Ed. by C. Hayashi, K. Yajima, H.-H. Bock, N. Ohsumi, Y. Tanaka, and Y. Baba. Tokyo: Springer Japan, 1998, pp. 40–51. ISBN: 978-4-431-65950-1.

[103] D. Haziza, J. Rapin, and G. Synnaeve. *HiPlot: High-dimensional interactive plots made easy*. [Online; accessed 12-February-2020]. 2020. URL: https://ai.facebook.com/blog/hiplot-high-dimensional-interactive-plots-made-easy/.

[104] K. Healy. *Data Visualization: A Practical Introduction*. First edition. Princeton University Press, 2018. ISBN: 978-0691181622.

[105] J. Heer. *Research Publications*. [Online; accessed 29-October-2020]. 2017. URL: https://vega.github.io/vega/about/research/.

[106] J. Heer. *Label Transform*. [Online; accessed 29-October-2020]. 2020. URL: https://vega.github.io/vega/docs/transforms/label/.

[107] J. Hermann. "Michelangelo - Machine Learning @Uber." QCon San Francisco 2018. 2018. URL: https://www.youtube.com/watch?v=iCpp5mqTeXE.

[108] J. Hermann and M. D. Balso. *Meet Michelangelo: Uber's Machine Learning Platform*. [Online; accessed 26-November-2020]. 2017. URL: https://eng.uber.com/michelangelo-machine-learning-platform/.

[109] J. Hermann and M. D. Balso. *Scaling Machine Learning at Uber with Michelangelo*. [Online; accessed 26-November-2020]. 2018. URL: https://eng.uber.com/scaling-michelangelo/.

[110] Highsoft AS. *Highcharts JS*. https://github.com/highcharts/highcharts. 2011.

[111] F. Hohman, M. Kahng, R. Pienta, and D. H. Chau. "Visual Analytics in Deep Learning: An Interrogative Survey for the Next Frontiers." In: *CoRR* abs/1801.06889 (2018). arXiv: 1801.06889. URL: http://arxiv.org/abs/1801.06889.

[112] F. Hohman, A. Srinivasan, and S. M. Drucker. "TeleGam: Combining Visualization and Verbalization for Interpretable Machine Learning." In: *IEEE Visualization Conference (VIS)* (2019). URL: https://poloclub.github.io/telegam/.

[113] E. Holder. *Settling the Debate: Bars vs. Lollipops (vs. Dot Plots)*. [Online; accessed 26-November-2020]. 2020. URL: https://medium.com/nightingale/bar-graphs-vs-lollipop-charts-vs-dot-plots-experiment-ba0bd8aad5d6.

[114] Y. Holtz and C. Healy. *LOLLIPOP CHART*. [Online; accessed 26-November-2020]. 2018. URL: https://www.data-to-viz.com/graph/lollipop.html.

[115] A. K. Hopkins, M. Correll, and A. Satyanarayan. "VisuaLint: Sketchy In Situ Annotations of Chart Construction Errors." In: *Computer Graphics Forum* 39.3 (2020), pp. 219–228. DOI: https://doi.org/10.1111/cgf.13975. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.13975. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.13975.

[116] jonnyjandles (https://meta.stackoverflow.com/users/2080074/jonnyjandles). *Multiprocessing causes Python to crash and gives an error may have been in progress in another thread when fork() was called*. Stack Overflow. https://stackoverflow.com/q/50168647 (version: 2018-05-04). eprint: https://stackoverflow.com/q/50168647. URL: https://stackoverflow.com/a/52230415.

[117] koffein (https://meta.stackoverflow.com/users/2964777/koffein). *Controlling distance of shuffling*. Stack Overflow. https://stackoverflow.com/q/30747690 (version: 2015-06-17). eprint: https://stackoverflow.com/q/30747690. URL: https://stackoverflow.com/a/30784808.

[118] D. Hunt and Contributors. *pytest-selenium*. https://github.com/pytest-dev/pytest-selenium. 2015.

[119] IEEE Computational Intelligence Society and Vesta Corporation. *IEEE-CIS Fraud Detection*. [Online; accessed 30-October-2020]. July 2019. URL: https://www.kaggle.com/c/ieee-fraud-detection.

[120] P. T. Inc. *Collaborative data science*. 2015. URL: https://plot.ly.

[121] N. Japkowicz and M. Shah. *Evaluating Learning Algorithms: A Classification Perspective*. Cambridge University Press, 2011. DOI: 10.1017/CBO9780511921803.

[122] I. Johnson. *Machine Learning for Visualization*. [Online; accessed 29-September-2020]. 2018. URL: https://medium.com/@enjalot/machine-learning-for-visualization-927a9dff1cab.

[123] K. Jolly. *Machine Learning with scikit-learn Quick Start Guide*. Packt Publishing, 2018. ISBN: 9781789343700.

[124] G. Jones. *Microsoft LightGBM (~0.795)*. Kaggle notebook. Version 7. [Online; accessed 30-October-2020]. 2017. URL: https://www.kaggle.com/garethjns/microsoft-lightgbm-0-795.

[125] Jupyter Development Team. *ipywidgets*. https://github.com/jupyter-widgets/ipywidgets. 2015.

[126] Jupyter Development Team. *nbconvert*. https://github.com/jupyter/nbconvert. 2015.

[127] Kaggle. *Titanic: Machine Learning from Disaster*. [Online; accessed 30-October-2020]. Sept. 2012. URL: https://www.kaggle.com/c/titanic.

[128] Kaggle. *Kaggle's State of Data Science and Machine Learning 2019*. Tech. rep. 2019. URL: https://www.kaggle.com/kaggle-survey-2019.

[129] M. Kahng, P. Y. Andrews, A. Kalro, and D. H. Chau. "ActiVis: Visual Exploration of Industry-Scale Deep Neural Network Models." In: *CoRR* abs/1704.01942 (2017). arXiv: 1704.01942. URL: http://arxiv.org/abs/1704.01942.

[130] A. Kallner. "Formulas." In: *Laboratory Statistics (Second Edition)*. Ed. by A. Kallner. Second Edition. Elsevier, 2018, pp. 1–140. ISBN: 978-0-12-814348-3. DOI: https://doi.org/10.1016/B978-0-12-814348-3.00001-0. URL: http://www.sciencedirect.com/science/article/pii/B9780128143483000010.

[131] A. Karpathy, [@karpathy]. *When you sort your dataset descending by loss you are guaranteed to find something unexpected, strange and helpful.* [Online; accessed 8-October-2020]. Twitter. Oct. 2, 2020. URL: https://twitter.com/karpathy/status/1311884485676294151.

[132] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu. "LightGBM: A Highly Efficient Gradient Boosting Decision Tree." In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Curran Associates, Inc., 2017, pp. 3146–3154. URL: http://papers.nips.cc/paper/6907-lightgbm-a-highly-efficient-gradient-boosting-decision-tree.pdf.

[133] H. van Kemenade, wiredfool, A. Murray, A. Clark, A. Karpinsky, nulano, C. Gohlke, J. Dufresne, B. Crowell, D. Schmidt, A. Houghton, K. Kopachev, S. Mani, S. Landey, vashek, J. Ware, Jason, D. Caro, S. Kossouho, R. Lahd, A. Lee, E. W. Brown, O. Tonnhofer, M. Bonfill, P. R. (변기호), F. Al-Saidi, M. Górny, M. Korobov, M. Kurczewski, and B. Yang. *python-pillow/Pillow: 7.2.0*. Version 7.2.0. June 2020.

DOI: 10.5281/zenodo.3923759. URL: https://doi.org/10.5281/zenodo.3923759.

[134] M. Khalusova. *ML-JVM*. https://github.com/MKhalusova/ML-JVM. 2020.

[135] M. R. A. Khan. *ROCit: An R Package for Performance Assessment of Binary Classifier with Visualization*. [Online; accessed 1-October-2020]. 2020. URL: https://cran.r-project.org/web/packages/ROCit/vignettes/my-vignette.html.

[136] R. Khan. *Binary classifier evaluation metrics: error rate, KS statistic, AUROC, lift, gains table*. [Online; accessed 28-October-2020]. 2017. URL: http://rstudio-pubs-static.s3.amazonaws.com/303414_fb0a43efb0d7433983fdc9adcf87317f.html.

[137] Y. Kim and J. Heer. "Gemini: A Grammar and Recommender System for Animated Transitions in Statistical Graphics." In: *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis)* (2021). URL: http://idl.cs.washington.edu/papers/gemini.

[138] A. Kirk. *SLOPE GRAPH*. [Online; accessed 29-November-2020]. 2015. URL: http://seeingdata.org/taketime/inside-the-chart-slope-graph/.

[139] C. Kittivorawang, D. Moritz, K. Wongsuphasawat, and J. Heer. "Fast and Flexible Overlap Detection for Chart Labeling with Occupancy Bitmap." In: *IEEE VIS Short Papers*. 2020. URL: http://idl.cs.washington.edu/papers/fast-labels.

[140] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla, and C. Willing. "Jupyter Notebooks – a publishing format for reproducible computational workflows." In: *Positioning and Power in Academic Publishing: Players, Agents and Agendas*. 2016, pp. 87–90. DOI: 10.3233/978-1-61499-649-1-87. URL: http://ebooks.iospress.nl/publication/42900.

[141] C. N. Knaflic. *exploratory vs explanatory analysis*. [Online; accessed 29-September-2020]. 2014. URL: http://www.storytellingwithdata.com/blog/2014/04/exploratory-vs-explanatory-analysis.

[142] KNIME AG. *Binary Classification Inspector*. Version 4.2.1. [Online; accessed 29-November-2020]. 2020. URL: https://kni.me/n/3-JGPq9anCe8LGG6.

[143] KNIME AG. *Scorer*. Version 4.2.3. [Online; accessed 29-November-2020]. 2020. URL: https://kni.me/n/5tsiYT6vDKM7YChN.

[144] J. Koponen and J. Hildén. *Data Visualization Handbook*. First edition. Aalto University, 2019. ISBN: 978-9526074498.

[145] R. Kosara. *Parallel Coordinates*. [Online; accessed 12-February-2020]. 2010. URL: https://eagereyes.org/techniques/parallel-coordinates.

[146] T. Krabel, F. Wetschoreck, and 8080 Labs. *bamboolib*. https://github.com/tkrabel/bamboolib. 2019.

[147] J. Krause, A. Dasgupta, J. Swartz, Y. Aphinyanaphongs, and E. Bertini. *A Workflow for Visual Diagnostics of Binary Classifiers using Instance-Level Explanations*. 2017. arXiv: 1705.01968 [stat.ML].

[148] J. Krause, A. Perer, and K. Ng. "Interacting with Predictions: Visual Inspection of Black-box Machine Learning Models." In: *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. CHI '16. Santa Clara, California, USA: ACM, 2016, pp. 5686–5697. ISBN: 978-1-4503-3362-7. DOI: 10.1145/2858036.2858529. URL: http://doi.acm.org/10.1145/2858036.2858529.

[149] H. Krekel, B. Oliveira, R. Pfannschmidt, F. Bruynooghe, B. Laugher, and F. Bruhin. *pytest*. 2004. URL: https://github.com/pytest-dev/pytest.

[150] P. Krensky, P. den Hamer, E. Brethenoux, J. Hare, C. Idione, A. Linden, S. Sicular, and F. Choudhary. *2020 Magic Quadrant for Data Science and Machine Learning Platforms*. Tech. rep. https://www.alteryx.com/third-party-content/gartner-2020-mq-data-science-machine-learning and https://www.gartner.com/en/documents/3980855. 2020.

[151] N. Kruchten, E. Gouillart, A. Siddiqui, J. Damiba, C. Fox, and Plotly Technologies Inc. *Version 4 Migration Guide in Python*. [Online; accessed 24-November-2020]. 2019. URL: https://plotly.com/python/v4-migration/.

[152] T. Kulesza, M. Burnett, W.-K. Wong, and S. Stumpf. "Principles of Explanatory Debugging to Personalize Interactive Machine Learning." In: vol. 2015. Mar. 2015. DOI: 10.1145/2678025.2701399.

[153] O. Lacan. *Keep a Changelog*. [Online; accessed 31-October-2020]. 2014. URL: https://keepachangelog.com/.

[154] M. Lacchia and Contributors. *Radon*. https://github.com/rubik/radon. 2012.

[155] H. Lai. *Why Ruby app servers break on macOS High Sierra and what can be done about it*. [Online; accessed 27-November-2020]. 2017. URL: https://blog.phusion.nl/2017/10/13/why-ruby-app-servers-break-on-macos-high-sierra-and-what-can-be-done-about-it/.

[156] N. Lally. *The false promise of Chargeback Guarantee Models in fraud detection*. [Online; accessed 4-November-2020]. URL: https://www.ravelin.com/blog/the-false-realities-of-chargeback-guarantee-models-in-fraud-detection.

[157] T. K. Landauer and D. W. Nachbar. "Selection from Alphabetic and Numeric Menu Trees Using a Touch Screen: Breadth, Depth, and Width." In: *SIGCHI Bull.* 16.4 (Apr. 1985), 73–78. ISSN: 0736-6906. DOI: 10.1145/1165385.317470. URL: https://doi.org/10.1145/1165385.317470.

[158] M. Lang and M. Binder. *mlr3measures*. https://github.com/mlr-org/mlr3measures. 2019.

[159] M. Lang, P. Schratz, R. Sonabend, and D. Pulatov. *mlr3viz*. https://github.com/mlr-org/mlr3viz. 2020.

[160] Łukasz Langa. *flake8-bugbear*. https://github.com/PyCQA/flake8-bugbear. 2016.

[161] Łukasz Langa and Python Software Foundation. *Black*. https://github.com/psf/black. 2018.

[162] E. LeDell, M. Petersen, and M. van der Laan. "Computationally efficient confidence intervals for cross-validated area under the ROC curve estimates." In: *Electron. J. Statist.* 9.1 (2015), pp. 1583–1607. DOI: 10.1214/15-EJS1035. URL: https://doi.org/10.1214/15-EJS1035.

[163] D. J.-L. Lee. *Lux*. https://github.com/lux-org/lux. 2020.

[164] E. Lees. *Understanding The Altair Stack*. [Online; accessed 31-October-2020]. 2020. URL: https://eitanlees.github.io/altair-stack/.

[165] J. Lehtosalo and mypy contributors. *mypy*. https://github.com/python/mypy. 2014.

[166] G. Lemaître, F. Nogueira, and C. K. Aridas. "Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning." In: *Journal of Machine Learning Research* 18.17 (2017), pp. 1–5. URL: http://jmlr.org/papers/v18/16-365.html.

[167] H. Li. *Smile*. https://haifengl.github.io. 2014.

[168] L. Li. *Open Sourcing Manifold, a Visual Debugging Tool for Machine Learning*. [Online; accessed 1-October-2020]. 2020. URL: https://eng.uber.com/manifold-open-source/.

[169] R. Linacre. *Why I'm backing Vega-Lite as our default tool for data visualisation*. [Online; accessed 29-October-2020]. 2018. URL: https://robinlinacre.medium.com/why-im-backing-vega-lite-as-our-default-tool-for-data-visualisation-51c20970df39.

[170] LinkedIn Corporation. *Dagli*. Version 15.0.0-beta5. Nov. 15, 2020. URL: https://github.com/linkedin/dagli.

[171] F. T. Liu, K. M. Ting, and Z. Zhou. "Isolation Forest." In: *2008 Eighth IEEE International Conference on Data Mining*. Dec. 2008, pp. 413–422. DOI: 10.1109/ICDM.2008.17.

[172] S. Liu, J. Xiao, J. Liu, X. Wang, J. Wu, and J. Zhu. "Visual Diagnosis of Tree Boosting Methods." In: *IEEE Transactions on Visualization and Computer Graphics* 24.1 (Jan. 2018), pp. 163–173. ISSN: 2160-9306. DOI: 10.1109/TVCG.2017.2744378.

[173] A. Lukyanenko. *EDA and models*. Kaggle notebook. Version 37. [Online; accessed 30-October-2020]. 2019. URL: https://www.kaggle.com/artgor/eda-and-models.

[174] Lynn@Vesta. *Data Description (Details and Discussion)*. [Online; accessed 4-November-2020]. 2019. URL: https://www.kaggle.com/c/ieee-fraud-detection/discussion/101203#589276.

[175] L. v. d. Maaten and G. Hinton. "Visualizing High-Dimensional Data Using t-SNE." In: *Journal of Machine Learning Research* 9.Nov (2008), pp. 2579–2605.

[176] G. M. Machado, M. M. Oliveira, and L. A. F. Fernandes. "A Physiologically-based Model for Simulation of Color Vision Deficiency." In: *IEEE Transactions on Visualization and Computer Graphics* 15.6 (Nov. 2009), pp. 1291–1298.

[177] S. Macke, H. Gong, D. J.-L. Lee, A. Head, D. Xin, and A. Parameswaran. *Fine-Grained Lineage for Safer Notebook Interactions*. 2020. URL: https://smacke.net/papers/nbsafety.pdf.

[178] Mailchimp. *Typography*. [Online; accessed 31-October-2020]. 2013. URL: https://templates.mailchimp.com/design/typography/.

[179] C. Maçãs, E. Polisciuc, and P. Machado. "VaBank: Visual Analytics for Banking Transactions." In: *2020 24th International Conference Information Visualisation (IV)*. 2020.

[180] Wes McKinney. "Data Structures for Statistical Computing in Python." In: *Proceedings of the 9th Python in Science Conference*. Ed. by Stéfan van der Walt and Jarrod Millman. 2010, pp. 56–61. DOI: 10.25080/Majora-92bf1922-00a.

[181] E. Meeks. *Data Visualization, Fast and Slow*. [Online; accessed 29-September-2020]. 2018. URL: https://medium.com/nightingale/data-visualization-fast-and-slow-d2653d4850b0.

[182] E. Meeks. *WHAT CHARTS DO*. [Online; accessed 29-September-2020]. 2018. URL: https://medium.com/nightingale/what-charts-do-48ed96f70a74.

[183] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen, D. Xin, R. Xin, M. J. Franklin, R. Zadeh, M. Zaharia, and A. Talwalkar. *MLlib: Machine Learning in Apache Spark*. 2015. arXiv: 1505.06807 [cs.LG].

[184] Microsoft. *The Team Data Science Process lifecycle*. [Online; accessed 27-September-2020]. 2017. URL: https://docs.microsoft.com/en-us/azure/machine-learning/team-data-science-process/lifecycle.

[185] Microsoft Corporation. *Visual Studio Code*. https://github.com/microsoft/vscode. 2015.

[186] I. Mierswa, M. Wurst, R. Klinkenberg, M. Scholz, and T. Euler. "YALE: Rapid Prototyping for Complex Data Mining Tasks." In: *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '06. Philadelphia, PA, USA: Association for Computing Machinery, 2006, 935–940. ISBN: 1595933395. DOI: 10.1145/1150402.1150531. URL: https://doi.org/10.1145/1150402.1150531.

[187] Y. Ming, H. Qu, and E. Bertini. *RuleMatrix: Visualizing and Understanding Classifiers with Rules*. 2018. arXiv: 1807.06228 [cs.LG].

[188] M. Mitchell, S. Wu, A. Zaldivar, P. Barnes, L. Vasserman, B. Hutchinson, E. Spitzer, I. D. Raji, and T. Gebru. "Model Cards for Model Reporting." In: *Proceedings of the Conference on Fairness, Accountability, and Transparency - FAT* '19* (2019). DOI: 10.1145/3287560.3287596. URL: http://dx.doi.org/10.1145/3287560.3287596.

[189] MLJAR, Inc. *mljar-supervised*. https://github.com/mljar/mljar-supervised. 2019.

[190] P. Molino, Y. Dudin, and S. S. Miryala. *Ludwig*. https://github.com/uber/ludwig. 2019.

[191] P. Molino, Y. Dudin, and S. S. Miryala. *Ludwig: a type-based declarative deep learning toolbox*. 2019. eprint: arXiv:1909.07930.

[192] C. Molnar. *Interpretable Machine Learning. A Guide for Making Black Box Models Explainable*. https://christophm.github.io/interpretable-ml-book/. 2019.

[193] I. Montani. "Designing Practical NLP Solutions." L3-AI. 2020. URL: https://www.youtube.com/watch?v=JpkzK58lkmA.

[194] D. Moritz. *Incorporate Vega-Tooltip*. GitHub Issue. [Online; accessed 1-November-2020]. 2018. URL: https://github.com/altair-viz/altair/issues/240#issuecomment-374432432.

[195] D. Moritz. *Message posted in the #altair channel of the Vega workspace*. Slack. [Online; accessed 31-October-2020]. 2020. URL: https://vega-js.slack.com/archives/C7K00SK98/p1603333839014700.

[196] D. Moritz, B. E. Granger, and J. VanderPlas. *IPython Vega*. https://github.com/vega/ipyvega. 2015.

[197] D. Moritz, S. Horradarn, Z. Qu, and UW Interactive Data Lab. *Vega Tooltip*. https://github.com/vega/vega-tooltip. 2016.

[198] D. Moritz and UW Interactive Data Lab. *Vega-Embed*. https://github.com/vega/vega-embed. 2015.

[199] D. Moritz, C. Wang, G. Nelson, H. Lin, A. M. Smith, B. Howe, and J. Heer. "Formalizing Visualization Design Knowledge as Constraints: Actionable and Extensible Models in Draco." In: *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis)* (2019). URL: http://idl.cs.washington.edu/papers/draco.

[200] D. Moritz, K. Wongsuphasawat, and Vega-Lite contributors. *Scale*. [Online; accessed 31-October-2020]. 2020. URL: https://vega.github.io/vega-lite/docs/scale.html.

[201] B. Morrow, T. Manz, A. E. Chung, N. Gehlenborg, and D. Gotz. *Periphery Plots for Contextualizing Heterogeneous Time-Based Charts*. 2019. arXiv: 1906.07637 [cs.HC].

[202] K. P. Murphy. *Machine Learning: A Probabilistic Perspective*. First edition. The MIT Press, 2012. ISBN: 978-0262018029.

[203] R. Nakano, C. Wells, wikke, lugq, R. J. Liwag, M. Emery, L. Miranda, J. Engelman, F. Herfert, E. Can, D. Friedman, and C. Ren. *reiinakano/scikit-plot: v0.3.5*. Version v0.3.5. May 2018. DOI: 10.5281/zenodo.1245853. URL: https://doi.org/10.5281/zenodo.1245853.

[204] S. S. Nazrul. *UX Design Guide for Data Scientists and AI Products*. [Online; accessed 15-October-2020]. 2018. URL: https://towardsdatascience.com/ux-design-guide-for-data-scientists-and-ai-products-465d32d939b0.

[205] B. Neville. *Viz Variety Show: When to use a lollipop chart and how to build one*. [Online; accessed 26-November-2020]. 2017. URL: https://www.tableau.com/about/blog/2017/1/viz-whiz-when-use-lollipop-chart-and-how-build-one-64267.

[206] E. Ngai, Y. Hu, Y. Wong, Y. Chen, and X. Sun. "The application of data mining techniques in financial fraud detection: A classification framework and an academic review of literature." In: *Decision Support Systems* 50.3 (2011). On quantitative methods for detection of financial fraud, pp. 559–569. ISSN: 0167-9236. DOI: https://doi.org/10.1016/j.dss.2010.08.006. URL: http://www.sciencedirect.com/science/article/pii/S0167923610001302.

[207] A. Niculescu-Mizil and R. Caruana. "Predicting good probabilities with supervised learning." In: Jan. 2005, pp. 625–632. DOI: 10.1145/1102351.1102430.

[208] J. Nielsen. *How Many Test Users in a Usability Study?* [Online; accessed 15-October-2020]. 2012. URL: https://www.nngroup.com/articles/how-many-test-users/.

[209] B. Nunnally and D. Farkas. *UX Research: Practical Techniques for Designing Better Products*. First edition. O'Reilly Media, 2016. ISBN: 978-1491951293.

[210] R. Okamoto, S. Verma, S. Kaur, and E. Moorhouse. *hueniversitypy*. https://github.com/UBC-MDS/hueniversitypy. 2020.

[211] Oracle. *Tribuo*. Version 4.0.2. Nov. 5, 2020. URL: https://tribuo.org/.

[212] J. Palmeiro. *coauthorship-graph*. Plotly Technologies Inc., https://plotly.com/~joaopalmeiro/79/ (Last accessed: November 24, 2020).

[213] J. Palmeiro. *Plotting the first point of the Feedzai Charting Library*. [Online; accessed 30-November-2020]. 2019. URL: https://medium.com/feedzaitech/plotting-the-first-point-of-the-feedzai-charting-library-50f21b4a5e01.

[214] T. Parr, T. Lapusan, and P. Grover. *dtreeviz*. https://github.com/parrt/dtreeviz. 2018.

[215] K. Patel, S. M. Drucker, J. Fogarty, A. Kapoor, and D. Tan. "Using Multiple Models to Understand Data." In: *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI 2011)*. AAAI Press, July 2011, pp. 1723–1728. ISBN: 978-1-57735-514-4. URL: https://www.microsoft.com/en-us/research/publication/using-multiple-models-understand-data/.

[216] M. Pathak. *Introduction to t-SNE*. [Online; accessed 12-October-2020]. 2018. URL: https://www.datacamp.com/community/tutorials/introduction-t-sne.

[217] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. "Scikit-learn: Machine Learning in Python." In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[218] F. Pedregosa, P. Gervais, and memory_profiler contributors. *memory_profiler*. https://github.com/pythonprofilers/memory_profiler. 2012.

[219] E. Peirson and ASU Digital Innovation Group (DigInG). *Tethne*. https://github.com/diging/tethne. 2014.

[220] F. Pérez and B. E. Granger. "IPython: a System for Interactive Scientific Computing." In: *Computing in Science and Engineering* 9.3 (May 2007), pp. 21–29. ISSN: 1521-9615. DOI: 10.1109/MCSE.2007.53. URL: https://ipython.org.

[221] C. Phua, V. C. S. Lee, K. Smith-Miles, and R. W. Gayler. "A Comprehensive Survey of Data Mining-based Fraud Detection Research." In: *CoRR* abs/1009.6119 (2010). arXiv: 1009.6119. URL: http://arxiv.org/abs/1009.6119.

[222] A. Piccolboni. *altair_recipes*. https://github.com/piccolbo/altair_recipes. 2018.

[223] F. Pinto, M. O. P. Sampaio, and P. Bizarro. *Automatic Model Monitoring for Data Streams*. 2019. arXiv: 1908.04240 [cs.LG].

[224] Pipenv maintainer team. *Pipenv*. https://github.com/pypa/pipenv. 2017.

[225] Plotly. *What is a SPLOM chart? Making scatterplot matrices in Python*. [Online; accessed 29-October-2020]. 2018. URL: https://medium.com/plotly/what-is-a-splom-chart-make-scatterplot-matrices-in-python-8dc4998921c3.

[226] D. Powers. "Evaluation: From Precision, Recall and F-Factor to ROC, Informedness, Markedness & Correlation." In: *Mach. Learn. Technol.* 2 (Jan. 2008).

[227] T. Preston-Werner. *Semantic Versioning 2.0.0*. [Online; accessed 31-October-2020]. 2013. URL: https://semver.org/.

[228] C. Priest. *How to Understand a DataRobot Model: Drilling Down into Model Accuracy [Part 3]*. [Online; accessed 29-November-2020]. 2018. URL: https://www.datarobot.com/blog/how-to-understand-a-datarobot-model-drilling-down-into-model-accuracy-part-3/.

[229] P. Probst, B. Bischl, and A.-L. Boulesteix. *Tunability: Importance of Hyperparameters of Machine Learning Algorithms*. 2018. arXiv: 1802.09596 [stat.ML].

[230] PwC. *Global Economic Crime and Fraud Survey 2018 - Perspetiva sobre Portugal*. Tech. rep. 2018. URL: https://www.pwc.pt/pt/advisory/forensic-services/gecs/pwc-gecs2018.pdf.

[231] PwC. *Global Economic Crime and Fraud Survey 2020*. Tech. rep. 2020. URL: https://www.pwc.com/gx/en/forensics/gecs-2020/pdf/global-economic-crime-and-fraud-survey-2020.pdf.

[232] Python Code Quality Authority. *Bandit*. https://github.com/PyCQA/bandit. 2015.

[233] Python Packaging Authority. *Setuptools*. https://github.com/pypa/setuptools. 2006.

[234] Python Packaging Authority. *Python Packaging User Guide*. [Online; accessed 31-October-2020]. 2013. URL: https://packaging.python.org/.

[235] Python Packaging Authority. *Python Packaging Authority*. [Online; accessed 31-October-2020]. 2014. URL: https://www.pypa.io/en/latest/.

[236] Python Software Foundation. *6.1.* string *— Common string operations*. [Online; accessed 16-November-2020]. 2016. URL: https://docs.python.org/3.6/library/string.html.

[237] Python Software Foundation and JetBrains. *Python Developers Survey 2019 Results*. Tech. rep. 2019. URL: https://www.jetbrains.com/lp/python-developers-survey-2019/.

[238] Quantopian Inc. *Qgrid*. https://github.com/quantopian/qgrid. 2015.

[239] RapidMiner Inc. *ANOVA*. Version 9.8.0. [Online; accessed 29-November-2020]. 2020. URL: https://docs.rapidminer.com/latest/studio/operators/validation/performance/significance_tests/anova.html.

[240] RapidMiner Inc. *Auto Model*. Version 9.8.0. [Online; accessed 29-November-2020]. 2020. URL: https://docs.rapidminer.com/latest/studio/guided/auto-model/.

[241] RapidMiner Inc. *T-Test*. Version 9.8.0. [Online; accessed 29-November-2020]. 2020. URL: https://docs.rapidminer.com/latest/studio/operators/validation/performance/significance_tests/t_test.html.

[242] RapidMiner Inc. *Visualize Model by SOM*. Version 9.8.0. [Online; accessed 29-November-2020]. 2020. URL: https://docs.rapidminer.com/latest/studio/operators/validation/visual/visualize_model_by_som.html.

[243] S. Raschka. *watermark*. https://github.com/rasbt/watermark. 2015.

[244] S. Raschka. "MLxtend: Providing machine learning and data science utilities and extensions to Python's scientific computing stack." In: *The Journal of Open Source Software* 3.24 (Apr. 2018). DOI: 10.21105/joss.00638. URL: http://joss.theoj.org/papers/10.21105/joss.00638.

[245] S. Raschka. *Model Evaluation, Model Selection, and Algorithm Selection in Machine Learning*. 2018. arXiv: 1811.12808 [cs.LG].

[246] S. Raschka, J. Patterson, and C. Nolet. *Machine Learning in Python: Main developments and technology trends in data science, machine learning, and artificial intelligence*. 2020. arXiv: 2002.04803 [cs.LG].

[247] L. Regebro. *Pyroma*. https://github.com/regebro/pyroma. 2011.

[248] D. Ren, S. Amershi, B. Lee, J. Suh, and J. D. Williams. "Squares: Supporting Interactive Performance Analysis for Multiclass Classifiers." In: *IEEE Transactions on Visualization and Computer Graphics* 23.1 (Jan. 2017), pp. 61–70. ISSN: 2160-9306. DOI: 10.1109/TVCG.2016.2598828.

[249] C. Ridpath and W. Chisholm. *Techniques For Accessibility Evaluation And Repair Tools*. WD not longer in development. https://www.w3.org/TR/2000/WD-AERT-20000426. W3C, Apr. 2000.

[250] X. Robin, N. Turck, A. Hainard, N. Tiberti, F. Lisacek, J.-C. Sanchez, and M. Müller. "pROC: an open-source package for R and S+ to analyze and compare ROC curves." In: *BMC Bioinformatics* 12.1 (2011), p. 77. ISSN: 1471-2105. DOI: 10.1186/1471-2105-12-77. URL: https://doi.org/10.1186/1471-2105-12-77.

[251] S. Robinson and J. Wexler. *Introducing the What-If Tool for Cloud AI Platform models*. [Online; accessed 1-October-2020]. 2019. URL: https://cloud.google.com/blog/products/ai-machine-learning/introducing-the-what-if-tool-for-cloud-ai-platform-models.

[252] A. Ronacher and Pallets. *Jinja (Jinja2)*. https://github.com/pallets/jinja. 2008.

[253] A. Rose. *PandasGUI*. https://github.com/adamerose/pandasgui. 2019.

[254] G. van Rossum, J. Lehtosalo, and Łukasz Langa. *Type Hints*. PEP 484. 2014. URL: https://www.python.org/dev/peps/pep-0484/.

[255] L. C. Rost. *How to pick more beautiful colors for your data visualizations*. [Online; accessed 22-October-2020]. 2020. URL: https://blog.datawrapper.de/beautifulcolors/.

[256] L. C. Rost. *How your colorblind and colorweak readers see your colors.* [Online; accessed 22-October-2020]. 2020. URL: https://blog.datawrapper.de/colorblindness-part1/.

[257] M. Sachs. "plotROC: A Tool for Plotting ROC Curves." In: *Journal of Statistical Software, Code Snippets* 79.2 (2017), pp. 1–19. ISSN: 1548-7660. DOI: 10.18637/jss.v079.c02. URL: https://www.jstatsoft.org/v079/c02.

[258] T. Saito and M. Rehmsmeier. "Precrec: fast and accurate precision-recall and ROC curve calculations in R." In: *Bioinformatics* 33 (1) (2017), pp. 145–147. DOI: 10.1093/bioinformatics/btw570.

[259] P. Saleiro, B. Kuester, L. Hinkson, J. London, A. Stevens, A. Anisfeld, K. T. Rodolfa, and R. Ghani. *Aequitas: A Bias and Fairness Audit Toolkit.* 2018. arXiv: 1811.05577 [cs.LG].

[260] S. Samuel and B. König-Ries. "ProvBook: Provenance-based Semantic Enrichment of Interactive Notebooks for Reproducibility." In: *Proceedings of the ISWC 2018 Posters & Demonstrations, Industry and Blue Sky Ideas Tracks co-located with 17th International Semantic Web Conference (ISWC 2018), Monterey, USA, October 8th - to - 12th, 2018.* 2018. URL: http://ceur-ws.org/Vol-2180/paper-57.pdf.

[261] S. Samuel and B. König-Ries. *ReproduceMeGit: A Visualization Tool for Analyzing Reproducibility of Jupyter Notebooks.* 2020. arXiv: 2006.12110 [cs.CY].

[262] A. Sanya. "Feature Engineering @Uber with Michelangelo." Phoenix Data Conference: Virtual Summit 2020. 2020. URL: https://vimeo.com/477753622.

[263] A. Satyanarayan and J. Heer. "Lyra: An Interactive Visualization Design Environment." In: *Computer Graphics Forum (Proc. EuroVis)* (2014). URL: http://idl.cs.washington.edu/papers/lyra.

[264] A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer. "Vega-Lite: A Grammar of Interactive Graphics." In: *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis)* (2017). URL: http://idl.cs.washington.edu/papers/vega-lite.

[265] A. Satyanarayan, R. Russell, J. Hoffswell, and J. Heer. "Reactive Vega: A Streaming Dataflow Architecture for Declarative Interactive Visualization." In: *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis)* (2016). URL: http://idl.cs.washington.edu/papers/reactive-vega-architecture.

[266] A. Satyanarayan, K. Wongsuphasawat, and J. Heer. "Declarative Interaction Design for Data Visualization." In: *ACM User Interface Software & Technology (UIST).* 2014. URL: http://idl.cs.washington.edu/papers/reactive-vega.

[267] E. Schofield. *Starborn.* https://github.com/PythonCharmers/starborn. 2018.

[268] A. Schonfeld and MAN Alpha Technology. *D-Tale.* https://github.com/man-group/dtale. 2019.

[269] S. Seabold and J. Perktold. “statsmodels: Econometric and statistical modeling with python.” In: *9th Python in Science Conference*. 2010.

[270] M. Seaman. *The Right Number of User Interviews*. [Online; accessed 15-October-2020]. 2015. URL: https://medium.com/@mitchelseaman/the-right-number-of-user-interviews-de11c7815d9.

[271] T. Segal. *Chargeback*. [Online; accessed 4-November-2020]. 2020. URL: https://www.investopedia.com/terms/c/chargeback.asp.

[272] S. Shah, R. Fernandez, and S. M. Drucker. “A system for real-time interactive analysis of deep learning training.” In: *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems, EICS 2019, Valencia, Spain, June 18-21, 2019*. 2019, 16:1–16:6. DOI: 10.1145/3319499.3328231. URL: https://arxiv.org/abs/2001.01215.

[273] V. Shapiro. *UX for AI: Trust as a Design Challenge*. [Online; accessed 15-October-2020]. 2018. URL: https://medium.com/sap-design/ux-for-ai-trust-as-a-design-challenge-62044e22c4ec.

[274] Shopify. *Data visualizations*. Polaris style guide. [Online; accessed 2-November-2020]. Aug. 2018. URL: https://polaris.shopify.com/design/data-visualizations.

[275] D. Skau and R. Kosara. “Arcs, Angles, or Areas: Individual Data Encodings in Pie and Donut Charts.” In: *Computer Graphics Forum* 35.3 (2016), pp. 121–130. DOI: https://doi.org/10.1111/cgf.12888. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.12888. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.12888.

[276] S. Smith. *Online Transaction Fraud to More Than Double to $25bn by 2020, finds Juniper Research*. [Online; accessed 24-September-2020]. 2016. URL: https://www.juniperresearch.com/press/press-releases/online-transaction-fraud-to-more-than-double.

[277] Sociedade Portuguesa de Estatística and Associação Brasileira de Estatística. *Glossário Inglês-Português de Estatística*. [Online; accessed 17-November-2020]. 2011. URL: https://www.spestatistica.pt/glossario.

[278] E. Socolofsky, D. Moritz, S. Horradarn, Z. Qu, R. Millette, and M. Chang. *Customizing Your Tooltip*. [Online; accessed 1-November-2020]. 2019. URL: https://github.com/vega/vega-tooltip/blob/master/docs/customizing_your_tooltip.md.

[279] J. Sousa. *JupyterLab: Next Generation Data Exploration*. [Online; accessed 31-October-2020]. 2019. URL: https://medium.com/feedzaitech/jupyterlab-next-generation-data-exploration-3d7fc6f2316c.

[280] T. Spinner, U. Schlegel, H. Schafer, and M. El-Assady. "explAIner: A Visual Analytics Framework for Interactive and Explainable Machine Learning." In: *IEEE Transactions on Visualization and Computer Graphics* (2019), 1–1. ISSN: 2160-9306. DOI: 10.1109/tvcg.2019.2934629. URL: http://dx.doi.org/10.1109/TVCG.2019.2934629.

[281] Stack Overflow. *2020 Annual Developer Survey*. Tech. rep. 2020. URL: https://insights.stackoverflow.com/survey/2020.

[282] E. W. Steyerberg, B. V. Calster, and M. J. Pencina. "Performance Measures for Prediction Models and Markers: Evaluation of Predictions and Classifications." In: *Revista Española de Cardiología (English Edition)* 64.9 (2011), pp. 788–794. ISSN: 18855857. DOI: 10.1016/j.rec.2011.05.004. URL: https://www.revespcardiol.org/en-performance-measures-for-prediction-models-articulo-S1885585711003604.

[283] M. Stone. *How we designed the new color palettes in Tableau 10*. [Online; accessed 22-October-2020]. 2016. URL: https://www.tableau.com/about/blog/2016/7/colors-upgrade-tableau-10-56782.

[284] H. Strobelt, S. Gehrmann, M. Behrisch, A. Perer, H. Pfister, and A. M. Rush. "Seq2Seq-Vis: A Visual Debugging Tool for Sequence-to-Sequence Models." In: *CoRR* abs/1804.09299 (2018). arXiv: 1804.09299. URL: http://arxiv.org/abs/1804.09299.

[285] K. Stumpf, S. Bedratiuk, and O. Cirit. *Michelangelo PyML: Introducing Uber's Platform for Rapid Python ML Model Development*. [Online; accessed 26-November-2020]. 2018. URL: https://eng.uber.com/michelangelo-pyml/.

[286] N. Stylianou, C. Guibourg, and BBC. *BBC Visual and Data Journalism cookbook for R graphics*. [Online; accessed 1-November-2020]. 2019. URL: https://bbc.github.io/rcookbook/.

[287] K. Sullivan. *Anti–Money Laundering in a Nutshell: Awareness and Compliance for Financial Personnel and Business*. First edition. Apress, Berkeley, CA, 2015. ISBN: 978-1430261605.

[288] S. Sánchez. *Consistently Beautiful Visualizations with Altair Themes*. [Online; accessed 1-November-2020]. 2018. URL: https://towardsdatascience.com/consistently-beautiful-visualizations-with-altair-themes-c7f9f889602.

[289] J. Talbot, B. Lee, A. Kapoor, and D. Tan. "EnsembleMatrix: Interactive Visualization to Support Machine Learning with Multiple Classifiers." In: *CHI '09 Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM Press, Apr. 2009, pp. 1283–1292. ISBN: 978-1-60558-246-7. URL: https://www.microsoft.com/en-us/research/publication/ensemblematrix-interactive-visualization-support-machine-learning-multiple-classifiers/.

[290] M. Tanaka, Ændrew Rininsland, and Y. Hinosawa. *C3*. https://github.com/c3js/c3. 2014.

[291] The PyPI Admins. *trove-classifiers*. https://github.com/pypa/trove-classifiers. 2020.

[292] ThetaRay. *Anti-Money Laundering: AI solution to detect previously unknown money laundering with industry lowest False Positive Rate*. Tech. rep. 2018. URL: https://thetaray.com/resources/aml-solutions/.

[293] E. Tiakas, A. N. Papadopoulos, and Y. Manolopoulos. "Skyline queries: An introduction." In: *2015 6th International Conference on Information, Intelligence, Systems and Applications (IISA)*. 2015, pp. 1–6. DOI: 10.1109/IISA.2015.7388053.

[294] S. Torborg. *How To Package Your Python Code*. [Online; accessed 31-October-2020]. 2012. URL: https://python-packaging.readthedocs.io/en/latest/.

[295] Urban Institute. *URBAN INSTITUTE DATA VISUALIZATION STYLE GUIDE*. [Online; accessed 1-November-2020]. 2015. URL: http://urbaninstitute.github.io/graphics-styleguide/.

[296] S. van den Elzen and J. J. van Wijk. "BaobabView: Interactive construction and analysis of decision trees." In: *2011 IEEE Conference on Visual Analytics Science and Technology (VAST)*. Oct. 2011, pp. 151–160. DOI: 10.1109/VAST.2011.6102453.

[297] J. VanderPlas, B. E. Granger, J. Heer, D. Moritz, K. Wongsuphasawat, A. Satyanarayan, E. Lees, I. Timofeev, B. Welsh, and S. Sievert. "Altair: Interactive Statistical Visualizations for Python." In: *Journal of Open Source Software* 3.32 (2018), p. 1057. DOI: 10.21105/joss.01057. URL: https://doi.org/10.21105/joss.01057.

[298] J. VanderPlas. *Python Data Science Handbook: Essential Tools for Working with Data*. 1st. O'Reilly Media, Inc., 2016. ISBN: 1491912057.

[299] J. VanderPlas. *altair_saver*. https://github.com/altair-viz/altair_saver. 2019.

[300] J. VanderPlas. *How can I style the tooltip when creating a custom theme?* GitHub Issue. [Online; accessed 1-November-2020]. 2020. URL: https://github.com/altair-viz/altair/issues/1970#issuecomment-586545913.

[301] J. VanderPlas and Altair Developers. *Frequently Asked Questions*. Altair 4.1.0. [Online; accessed 31-October-2020]. 2018. URL: https://altair-viz.github.io/user_guide/faq.html.

[302] J. VanderPlas and Altair Developers. *Top-Level Chart Configuration*. Altair 4.1.0. [Online; accessed 1-November-2020]. 2019. URL: https://altair-viz.github.io/user_guide/configuration.html.

[303] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python." In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

[304] P. Virtanen and numpydoc maintainers. *numpydoc – Numpy's Sphinx extensions*. Version 1.0.0. [Online; accessed 30-November-2020]. 2020. URL: https://numpydoc.readthedocs.io/.

[305] S. Wachter, B. Mittelstadt, and C. Russell. *Counterfactual Explanations without Opening the Black Box: Automated Decisions and the GDPR*. 2017. arXiv: 1711.00399 [cs.AI].

[306] S. Wallis. "Binomial Confidence Intervals and Contingency Tests: Mathematical Fundamentals and the Evaluation of Alternative Methods." In: *Journal of Quantitative Linguistics* 20.3 (2013), pp. 178–208. DOI: 10.1080/09296174.2013.799918. eprint: https://doi.org/10.1080/09296174.2013.799918. URL: https://doi.org/10.1080/09296174.2013.799918.

[307] J. Wang, B. Yu, and L. Gasser. *Classification Visualization with Shaded Similarity Matrix*. Jan. 2002.

[308] M. Waskom and the seaborn development team. *mwaskom/seaborn*. Version latest. Sept. 2020. DOI: 10.5281/zenodo.592845. URL: https://doi.org/10.5281/zenodo.592845.

[309] A. Wathan, J. Reinink, D. Hemphill, S. Schoger, and Tailwind Labs. *Tailwind CSS*. https://github.com/tailwindlabs/tailwindcss. 2017.

[310] B. Welsh. *Pez charts*. Observable notebook. [Online; accessed 26-November-2020]. 2020. URL: https://observablehq.com/@datadesk/pez-charts.

[311] B. Welsh, C. L. Keller, M. Stiles, and The Los Angeles Times Data Desk. *altair-latimes*. https://github.com/datadesk/altair-latimes. 2019.

[312] J. Wexler. *Frequently Asked Questions*. [Online; accessed 8-October-2020]. 2020. URL: https://pair-code.github.io/what-if-tool/faqs/.

[313] J. Wexler. *What-If Tool releases*. [Online; accessed 1-October-2020]. 2020. URL: https://github.com/PAIR-code/what-if-tool/blob/master/RELEASE.md.

[314] J. Wexler, M. Pushkarna, T. Bolukbasi, M. Wattenberg, F. Viegas, and J. Wilson. "The What-If Tool: Interactive Probing of Machine Learning Models." In: *IEEE Transactions on Visualization and Computer Graphics* (2019), 1–1. ISSN: 2160-9306. DOI: 10.1109/tvcg.2019.2934619. URL: http://dx.doi.org/10.1109/TVCG.2019.2934619.

[315] H. Wickham. "A layered grammar of graphics." In: *Journal of Computational and Graphical Statistics* 19.1 (2010), 3–28. DOI: 10.1198/jcgs.2009.07098.

[316] H. Wickham. "Tidy Data." In: *Journal of Statistical Software, Articles* 59.10 (2014), pp. 1–23. ISSN: 1548-7660. DOI: 10.18637/jss.v059.i10. URL: https://www.jstatsoft.org/v059/i10.

[317] H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016. ISBN: 978-3-319-24277-4. URL: https://ggplot2.tidyverse.org.

[318] H. Wickham and G. Grolemund. *R for Data Science: Import, Tidy, Transform, Visualize, and Model Data*. 1st. O'Reilly Media, Inc., 2017. ISBN: 1491910399.

[319] M. Widmann. *Cohen's Kappa: what it is, when to use it, how to avoid pitfalls*. [Online; accessed 29-November-2020]. 2020. URL: https://www.knime.com/blog/cohens-kappa-an-overview.

[320] Wikipedia contributors. *Information retrieval — Wikipedia, The Free Encyclopedia*. [Online; accessed 30-November-2020]. 2017. URL: https://en.wikipedia.org/w/index.php?title=Information_retrieval&oldid=793358396.

[321] Wikipedia contributors. *Acquiring bank — Wikipedia, The Free Encyclopedia*. [Online; accessed 25-September-2020]. 2020. URL: https://en.wikipedia.org/w/index.php?title=Acquiring_bank&oldid=961443364.

[322] Wikipedia contributors. *Aguardiente — Wikipedia, The Free Encyclopedia*. [Online; accessed 23-November-2020]. 2020. URL: https://en.wikipedia.org/w/index.php?title=Aguardiente&oldid=987568385.

[323] Wikipedia contributors. *Arrows (Unicode block) — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=Arrows_(Unicode_block)&oldid=965504090. [Online; accessed 16-November-2020]. 2020.

[324] Wikipedia contributors. *Binomial distribution — Wikipedia, The Free Encyclopedia*. [Online; accessed 17-November-2020]. 2020. URL: https://en.wikipedia.org/w/index.php?title=Binomial_distribution&oldid=988089845.

[325] Wikipedia contributors. *Binomial proportion confidence interval — Wikipedia, The Free Encyclopedia*. [Online; accessed 17-November-2020]. 2020. URL: https://en.wikipedia.org/w/index.php?title=Binomial_proportion_confidence_interval&oldid=985324095.

[326] Wikipedia contributors. *Box-drawing character — Wikipedia, The Free Encyclopedia*. [Online; accessed 11-November-2020]. 2020. URL: https://en.wikipedia.org/w/index.php?title=Box-drawing_character&oldid=985813662.

[327] Wikipedia contributors. *Color blindness — Wikipedia, The Free Encyclopedia*. [Online; accessed 1-November-2020]. 2020. URL: https://en.wikipedia.org/w/index.php?title=Color_blindness&oldid=985474090.

[328] Wikipedia contributors. *Confusion matrix — Wikipedia, The Free Encyclopedia*. [Online; accessed 30-November-2020]. 2020. URL: https://en.wikipedia.org/w/index.php?title=Confusion_matrix&oldid=980584181.

[329] Wikipedia contributors. *Data science — Wikipedia, The Free Encyclopedia*. [Online; accessed 25-September-2020]. 2020. URL: https://en.wikipedia.org/w/index.php?title=Data_science&oldid=978881938.

[330] Wikipedia contributors. *Enclosed Alphanumerics — Wikipedia, The Free Encyclopedia*. [Online; accessed 30-November-2020]. 2020. URL: https://en.wikipedia.org/w/index.php?title=Enclosed_Alphanumerics&oldid=984532470.

[331] Wikipedia contributors. *Enclosed CJK Letters and Months — Wikipedia, The Free Encyclopedia*. [Online; accessed 30-November-2020]. 2020. URL: https://en.wikipedia.org/w/index.php?title=Enclosed_CJK_Letters_and_Months&oldid=981665751.

[332] Wikipedia contributors. *F-score — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=F-score&oldid=987334080. [Online; accessed 6-November-2020]. 2020.

[333] Wikipedia contributors. *Financial crime — Wikipedia, The Free Encyclopedia*. [Online; accessed 24-September-2020]. 2020. URL: https://en.wikipedia.org/w/index.php?title=Financial_crime&oldid=975787698.

[334] Wikipedia contributors. *Five Ws — Wikipedia, The Free Encyclopedia*. [Online; accessed 1-October-2020]. 2020. URL: https://en.wikipedia.org/w/index.php?title=Five_Ws&oldid=973365638.

[335] Wikipedia contributors. *Halstead complexity measures — Wikipedia, The Free Encyclopedia*. [Online; accessed 24-November-2020]. 2020. URL: https://en.wikipedia.org/w/index.php?title=Halstead_complexity_measures&oldid=960821107.

[336] Wikipedia contributors. *Ranking — Wikipedia, The Free Encyclopedia*. [Online; accessed 11-November-2020]. 2020. URL: https://en.wikipedia.org/w/index.php?title=Ranking&oldid=987805648.

[337] Wikipedia contributors. *YIQ — Wikipedia, The Free Encyclopedia*. [Online; accessed 26-November-2020]. 2020. URL: https://en.wikipedia.org/w/index.php?title=YIQ&oldid=968721834.

[338] C. O. Wilke. *Fundamentals of Data Visualization: A Primer on Making Informative and Compelling Figures*. First edition. O'Reilly Media, 2019. ISBN: 978-1492031086.

[339] L. Wilkinson. *The Grammar of Graphics*. Berlin, Heidelberg: Springer-Verlag, 1999. ISBN: 0387987746.

[340] H. Xu, T. Hunner, D. Ushakov, P. Palaga, A. Zerr, J. Mao, M. Laarman, J. Peek, B. Lopez, S. Sorhus, C.-H. Chang, J. Sundström, 10sr, I. Stepanyan, K. Bell, B. Neugebauer, M. Koppen, J. Yamamoto, R. Brackett, W. Swanson, nulltoken, A. Shapkin, C. Dias, M. Sero, K. On, and A. Keishima. *EditorConfig*. 2012. URL: https://editorconfig.org/.

[341] J. Yablonski. *Aesthetic Usability Effect*. [Online; accessed 29-September-2020]. 2019. URL: https://lawsofux.com/aesthetic-usability-effect.html.

[342] J. Yablonski. *Hick's Law*. [Online; accessed 29-September-2020]. 2019. URL: https://lawsofux.com/hicks-law.html.

[343] K. Yakovlev. *IEEE - FE for Local test*. Kaggle notebook. Version 1. [Online; accessed 30-October-2020]. 2019. URL: https://www.kaggle.com/kyakovlev/ieee-fe-for-local-test.

[344] J. S. Yi, Y. a. Kang, J. Stasko, and J. A. Jacko. "Toward a Deeper Understanding of the Role of Interaction in Information Visualization." In: *IEEE Transactions on Visualization and Computer Graphics* 13.6 (2007), pp. 1224–1231. DOI: 10.1109/TVCG.2007.70515.

[345] M. Yusuf. *Delorean*. https://github.com/myusuf3/delorean. 2013.

[346] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. "Spark: Cluster Computing with Working Sets." In: *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing*. HotCloud'10. Boston, MA: USENIX Association, 2010, p. 10.

[347] ZakJost. *Why ROC AUC for fraud detection?* [Online; accessed 28-September-2020]. 2019. URL: https://www.kaggle.com/c/ieee-fraud-detection/discussion/99982.

[348] A. Zeileis, J. C. Fisher, K. Hornik, R. Ihaka, C. D. McWhite, P. Murrell, R. Stauffer, and C. O. Wilke. *colorspace: A Toolbox for Manipulating and Assessing Colors and Palettes*. arXiv 1903.06490. arXiv.org E-Print Archive, Mar. 2019. URL: http://arxiv.org/abs/1903.06490.

[349] J. Zhang, Y. Wang, P. Molino, L. Li, and D. S. Ebert. "Manifold: A Model-Agnostic Framework for Interpretation and Diagnosis of Machine Learning Models." In: *IEEE Transactions on Visualization and Computer Graphics* 25.1 (Jan. 2019), 364–373. ISSN: 2160-9306. DOI: 10.1109/tvcg.2018.2864499. URL: http://dx.doi.org/10.1109/TVCG.2018.2864499.

[350] J. Zhao, M. Karimzadeh, A. Masjedi, T. Wang, X. Zhang, M. M. Crawford, and D. S. Ebert. *FeatureExplorer: Interactive Feature Selection and Exploration of Regression Models for Hyperspectral Images*. 2019. arXiv: 1908.00671 [cs.HC].

[351] K. Zhao, M. Ward, E. Rundensteiner, and H. Higgins. "LoVis: Local Pattern Visualization for Model Refinement." In: *Computer Graphics Forum* 33 (June 2014). DOI: 10.1111/cgf.12389.

[352] X. Zhao, Y. Wu, D. L. Lee, and W. Cui. "iForest: Interpreting Random Forests via Visual Analytics." In: *IEEE Transactions on Visualization and Computer Graphics* 25.1 (Jan. 2019), pp. 407–416. ISSN: 2160-9306. DOI: 10.1109/TVCG.2018.2864475.

[353] T. Ziadé, I. S. Cordasco, and Python Code Quality Authority. *Flake8*. https://gitlab.com/pycqa/flake8. 2010.

Appendix A

Literature Review summary

Table A.1: Summary of granularities of visual tools for ML Model Evaluation.

Paper	Year	Granularity
[28]	1998	Instance, Feature, Outcome
[307]	2002	Outcome, Class, Overall
[289]	2009	Outcome, Model, Class, Overall
[296]	2011	Instance, Feature, Outcome, Model, Class, Overall
[215]	2011	Instance, Outcome, Overall
[8]	2014	Instance, Feature, Outcome, Class, Overall
[351]	2014	Feature, Subgroup, Outcome, Overall
[12]	2015	Instance, Outcome, Overall
[152]	2015	Instance, Feature, Outcome, Overall
[148]	2016	Instance, Feature, Subgroup, Outcome
[147]	2017	Instance, Feature, Subgroup, Outcome, Overall
[248]	2017	Instance, Feature, Subgroup, Outcome, Class
[172]	2018	Instance, Feature, Outcome, Model, Class
[352]	2018	Instance, Feature, Outcome, Model
[187]	2018	Instance, Feature, Subgroup, Outcome
[314]	2019	Instance, Feature, Subgroup, Outcome, Overall
[168, 349]	2019	Feature, Subgroup, Outcome, Class
[350]	2019	Feature, Outcome, Overall
[5]	2019	Feature, Subgroup, Outcome, Overall
[3]	2019	Instance, Feature, Subgroup, Outcome, Overall
[280]	2019	Instance, Feature, Outcome, Model, Overall
[112]	2019	Instance, Feature, Outcome
[89]	2020	Instance, Feature, Outcome

The tools are matched to the following granularities:

- Instance: Individual data instance-level granularity.

- Feature: Feature-level granularity.
- Subgroup: Data subgroup-level granularity.
- Outcome: Outcome-level granularity. *Outcome* concerns the outputs of the models in terms of probabilities and labels, for example, and their manipulation and comparative evaluation translated into different metrics.
- Model: Internal model-level granularity.
- Class: Class-level granularity (mainly related to MCC problems).
- Overall: Overall performance-level granularity.

Table A.2: Summary of visual tools for ML Model Evaluation.

Paper	Year	OOTL	BC	MCC	Other Tasks	uFeatures	Multiple Models	Tracking of Changes	OOTB	Context	Target Group	Model	OSS
[28]	1998	●	●*	●*	○	○	○	○	○	Agnostic	DEV	●/○	○
[307]	2002	●	●	● (+/- 7)	○	○	●	○	○	Agnostic	DEV	●/○	○
[289]	2009	●/○	○	● (+/- 101)	○	○	○	○	○	Agnostic	DEV	○	○
[296]	2011	●/○	●	● (+/- 7)	○	○	○	○	○	Agnostic	DEV, DE	●/○	○
[215]	2011	●/○	●	● (+/- 20)	○	○	●	○	○	Agnostic	DEV	●	○
[8]	2014	●	●*	● (+/- 20)	○	●/○	●	○	○	Agnostic	DEV	●/○	○
[351]	2014	●/○	○	○	●	●/○	●	○	○	Agnostic	DEV	●	○
[12]	2015	●/○	●	●	●	○	○	○	○	Agnostic	DEV	●/○	○
[152]	2015	●/○	●	●*	○	●	○	●	○	Agnostic	DEV	●	○
[148]	2016	●	●	○	○	●	●	○	○	Healthcare	DEV	●	○
[147]	2017	●	●	○	○	●	○	○	○	Healthcare	DEV, DE	●	●
[248]	2017	●	○	● (3-20)	○	○	●	○	○	Agnostic	DEV	○	○
[172]	2018	●	●*	● (+/- 9)	○	●/○	○	○	○	Agnostic	DEV	○	●/○
[352]	2018	●	●	○	○	●	○	○	○	Agnostic	DEV	○	○
[187]	2018	●	●	● (+/- 4)	○	●	○	○	●	Agnostic	DEV, DE	○	●
[314]	2019	●/○	●	●/○ (+/- 3)	●	●/○	● (2)	○	●/○	Agnostic	DEV	●/○	●
[168, 349]	2019	●	●	● (+/- 3)	●	●	●	○	●/○	Agnostic	DEV	○	●
[350]	2019	●/○	○	○	●	●	●	○	○	Biomass	DEV, DE	●	○
[5]	2019	●	●	○	○	●	○	○	○	Agnostic	DEV	○	●
[3]	2019	●/○	○	○	●	●	○	○	○	Agnostic	DEV	●	●
[280]	2019	●/○	●	● (+/- 10)	●*	●	○	●*	○	Agnostic	DEV	●	○
[112]	2019	●	●	●	●	●/○	○	○	○	Agnostic	DEV	○	●
[89]	2020	●	●	○	○	●/○	○	○	●/○	Agnostic	DE	○	●

* Theoretically ● Property available ○ Property unavailable

Appendix B

Binary temporal confusion matrix

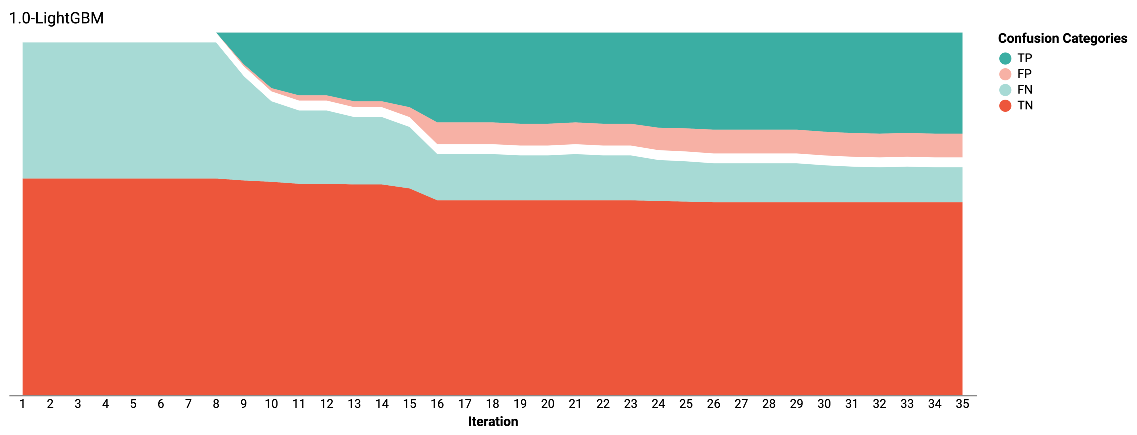


Figure B.1: Prototype of a binary temporal confusion matrix [172] for Gradient Boosting Machines like LightGBM [132] (inspired by a stacked area chart). The white *line* inside the chart serves only to separate the positives and the negatives (according to the predicted classes).

Listing B.1: Script to prepare the Titanic dataset [127], train a LightGBM model [132], assemble the dataset for visualization, and generate a binary temporal confusion matrix chart. The features engineering and model training parts are adapted from a Kaggle notebook created by Gareth Jones [124]. The time-series segmentation (*get_segment_cost()* and *time_series_segmentation()*) functions are adapted from an open-source implementation [58].

```
import re
import sys

import altair as alt
import lightgbm as lgb
```

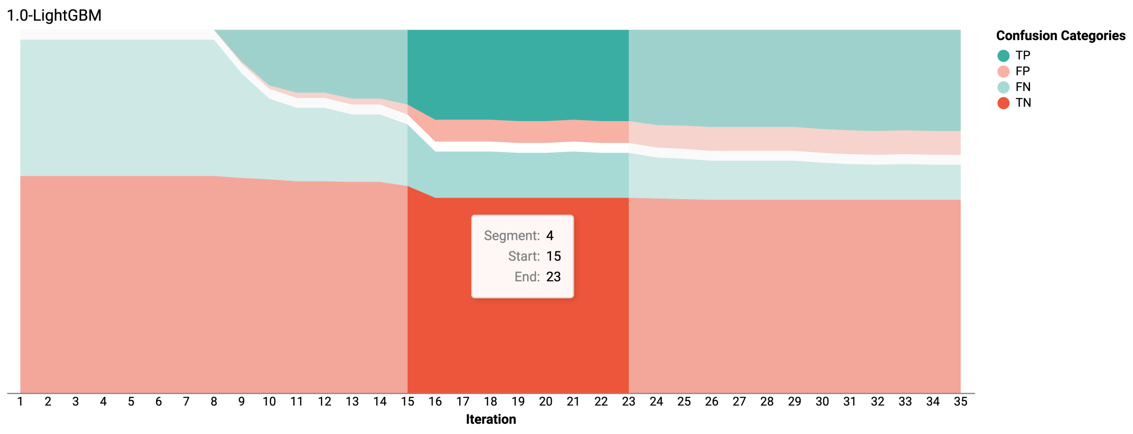


Figure B.2: Prototype of a binary temporal confusion matrix [172] when hovered. The opacity of the colors, as well as the tooltip, highlight one of the segments (whose number is specified by the user) obtained from a time-series segmentation algorithm that aims to minimize the intra-segment variances. The purpose of this algorithm and this visual layer is to help the user to quickly identify the iterations of interest as a proxy for considerable changes in confusion matrices.

```python
import numpy as np
import pandas as pd
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split

SEED = 0

TITLES = {
    "Capt": "Officer",
    "Col": "Officer",
    "Major": "Officer",
    "Jonkheer": "Sir",
    "Don": "Sir",
    "Sir": "Sir",
    "Dr": "Dr",
    "Rev": "Rev",
    "theCountess": "Lady",
    "Dona": "Lady",
    "Mme": "Mrs",
    "Mlle": "Miss",
    "Ms": "Mrs",
    "Mr": "Mr",
    "Mrs": "Mrs",
    "Miss": "Miss",
    "Master": "Master",
    "Lady": "Lady",
}

MODEL_PARAMS = {
    "boosting_type": "gbdt",
    "max_depth": -1,
    "objective": "binary",
    "nthread": 5,
    "num_leaves": 64,
    "learning_rate": 0.05,
    "max_bin": 512,
    "subsample_for_bin": 200,
    "subsample": 1,
    "subsample_freq": 1,
    "colsample_bytree": 0.8,
    "reg_alpha": 5,
    "reg_lambda": 10,
    "min_split_gain": 0.5,
    "min_child_weight": 1,
    "min_child_samples": 5,
    "scale_pos_weight": 1,
    "num_class": 1,
    "metric": "binary_error",
    "seed": SEED,
}


def load_datasets(*paths):
    """Load an arbitrary number of datasets in CSV format."""
    return [pd.read_csv(path) for path in paths]


def split_letter_number(cabin):
    """Extract cabin letter and number from the `Cabin` feature."""
    match = re.match(r"([a-z]+)([0-9]+)", cabin, re.I)

    try:
        letter = match.group(1)
    except AttributeError:
        letter = ""

    try:
        number = match.group(2)
    except AttributeError:
        number = 9999

    return letter, number


def split_cabin(cabin):
    """Extract cabin letter, number, and number of cabins from the `Cabin` feature."""
    if isinstance(cabin, (int, float)):
        letter = ""
        number = ""
```

```python
        n_cabins = 9999
    else:
        cabins = cabin.split()
        n_cabins = len(cabins)
        first_cabin = cabins[0]

        letter, number = split_letter_number(first_cabin)

    return letter, number, n_cabins


def split_name(name):
    """Extract the surname and the title from the `Name` feature (and standardize it)."""
    name = name.replace(".", "")
    name_parts = name.split("␣")

    surname = name_parts[0]

    name_title = [title for key, title in TITLES.items() if key in name_parts]

    if name_title == []:
        name_title = "Other"
    else:
        name_title = name_title[0]

    return surname.strip(","), name_title


def prep_dataset(data, class_col="", id_col="", ft_drop=()):
    """Prepare the dataset for use with LightGBM."""
    if class_col != "":
        labels = data[class_col]
        ft_drop = list(ft_drop) + [class_col]
    else:
        labels = []

    if id_col != "":
        ids = data[id_col]
    else:
        ids = []

    if ft_drop != []:
        data = data.drop(ft_drop, axis=1)

    l_data = lgb.Dataset(
        data,
        label=labels,
        free_raw_data=False,
        feature_name=list(data.columns),
        categorical_feature="auto",
    )

    return l_data, labels, ids, data


def get_predicted_classes(preds, threshold=0.5):
    """Obtain the predicted classes according to a threshold."""
    return (preds > threshold).astype(np.int)


def get_segment_cost(vectors, L, R):
    """Compute the cost function to measure the internal variance of a segment."""
    avg_vec = np.mean(vectors[L:R, :], axis=0)

    cost = 0
    for i in range(L, R + 1):
        delta = vectors[i, :] - avg_vec
        cost += np.sum(np.square((delta)))

    return cost


def time_series_segmentation(confusion_matrices, n_segment):
    """Divide a time series-like object (the confusion matrices over the various
        ↪ iterations) into a specified number of segments."""
    n_iteration = len(confusion_matrices)
    vectors = np.reshape(confusion_matrices, newshape=(n_iteration, -1))

    f = np.zeros(shape=(n_iteration, n_segment), dtype=np.float32)
    g = np.zeros(shape=(n_iteration, n_segment), dtype=np.int32)
    cost = np.zeros(shape=(n_iteration, n_iteration), dtype=np.float32)

    for i in range(n_iteration):
        cost[i][i] = 0
        for j in range(i + 1, n_iteration):
            cost[i][j] = get_segment_cost(vectors, i, j)

    for i in range(n_iteration):
        f[i][0] = cost[0][i]
        g[i][0] = -1

    for j in range(1, n_segment):
        for i in range(j - 1, n_iteration):
            f[i][j] = sys.maxsize
            for k in range(j - 2, i):
                if f[k][j - 1] + cost[k + 1][i] < f[i][j]:
                    f[i][j] = f[k][j - 1] + cost[k + 1][i]
                    g[i][j] = k

    endpoints = []
    i = np.int32(n_iteration - 1)
    j = n_segment - 1

    while i != -1:
        endpoints.append(i.item() + 1) # Add 1 because the X-axis starts at 1
        i = g[i][j]
        j = j - 1

    endpoints = endpoints[::-1]

    return endpoints


def temporal_confusion_matrix(
    df, segments, title, hover="outwards", width=800, height=300, **features
):
    """Generate a binary temporal confusion matrix (TCM) chart."""
    confusion_categories = ["TP", "FP", "FN", "TN"]

    color_scale = alt.Scale(
        domain=confusion_categories, range=["#3CAEA3", "#F6B1A5", "#A6DAD5", "#ED553B"],
    )

    legend_selection = alt.selection_multi(fields=["Confusion_Category"], bind="legend")

    if hover == "outwards":
        hover_selection = alt.selection_single(on="mouseover", nearest=False)
        color_selection = alt.condition(
            hover_selection, alt.value("transparent"), alt.value("#F4F4F4")
        )
        opacity_selection = alt.condition(
            hover_selection, alt.value(0.0), alt.value(0.5)
        )

    elif hover == "inwards":
        hover_selection = alt.selection_single(
            on="mouseover", nearest=False, empty="none"
        )
        color_selection = alt.condition(
            hover_selection, alt.value("#F4F4F4"), alt.value("transparent")
        )
        opacity_selection = alt.condition(
            hover_selection, alt.value(0.5), alt.value(0.0)
        )

    main_chart = (
        alt.Chart(df)
        .mark_area()
        .encode(
            alt.X(
                features["Iteration"] + ":O",
                axis=alt.Axis(
                    orient="bottom", labelAngle=0, ticks=False, title="Iteration"
                ),
            ),
            alt.Y(features["Count"] + ":Q", stack="zero", axis=None),
            alt.Color(
                features["Confusion_Category"] + ":N",
                title="Confusion␣Categories",
                scale=color_scale,
                legend=alt.Legend(values=confusion_categories),
            ),
            alt.Order(features["Class"], sort="ascending"),
            opacity=alt.condition(legend_selection, alt.value(1), alt.value(0.2)),
        )
        .properties(width=width, height=height)
        .add_selection(legend_selection)
    )

    hover_chart = (
        alt.Chart(segments)
        .mark_rect()
        .encode(
            alt.X(features["Seg_Start"] + ":O", axis=None),
            x2=features["Seg_End"] + ":O",
            color=color_selection,
            opacity=opacity_selection,
            tooltip=[
                alt.Tooltip(features["Seg_Number"], title="Segment"),
                alt.Tooltip(features["Seg_Start"], title="Start"),
                alt.Tooltip(features["Seg_End"], title="End"),
            ],
        )
        .add_selection(hover_selection)
    )

    tcm_chart = (
        alt.layer(main_chart, hover_chart, title=title)
        .configure_axis(grid=False)
        .configure_view(strokeWidth=0)
        .configure_title(anchor="start", fontWeight="normal")
    )

    return tcm_chart


if __name__ == "__main__":
    train, test = load_datasets("data/train.csv", "data/test.csv")
    full = pd.concat([train, test], axis=0, sort=True)

    # Metadata
    n_train = train.shape[0]
    ft_drop = ["Ticket", "Cabin", "Name"]
    class_col = "Survived"
    id_col = "PassengerId"
    conf_cat = ["TN", "FP", "FN", "TP"]
    class_types = [0, 2, 0, 2]
    n_segments = 5
    features = {
        "Iteration": "Iteration",
        "Count": "Count",
        "Confusion_Category": "Confusion_Category",
        "Class": "Class",
        "Seg_Start": "x_min",
        "Seg_End": "x_max",
        "Seg_Number": "number",
    }

    # `Cabin`-based features
    cabin_ft = full["Cabin"].apply(split_cabin).apply(pd.Series)
    cabin_ft.columns = ["CL", "CN", "nC"]

    full = pd.concat([full, cabin_ft], axis=1)

    # Family-based features
    full["fSize"] = full["SibSp"] + full["Parch"] + 1
    full["fRatio"] = (full["Parch"] + 1) / (full["SibSp"] + 1)
    full["Adult"] = full["Age"] > 18

    # `Name`-based features
    name_ft = full["Name"].apply(split_name).apply(pd.Series)
    name_ft.columns = ["Surname", "Title"]

    full = pd.concat([full, name_ft], axis=1)

    # Label encoding
    categorical_cols = ["Sex", "Embarked", "CL", "CN", "Surname", "Title"]

    for categorical in categorical_cols:
        full[categorical] = pd.Categorical(full[categorical])
        full[categorical] = full[categorical].cat.codes
        full[categorical] = pd.Categorical(full[categorical])

    # Age imputation
    full.loc[full["Age"].isnull(), "Age"] = np.median(
        full["Age"].loc[full["Age"].notnull()]
    )

    # Train-Test split
    train = full.iloc[0:n_train, :]
    test = full.iloc[n_train::, :]

    # Train-Validation split
    train_data, val_data = train_test_split(train, test_size=0.4, random_state=SEED)

    # LightGBM-ready datasets
    train_data_l, train_labels, train_ids, train_data = prep_dataset(
        train_data, class_col=class_col, id_col=id_col, ft_drop=ft_drop
    )

    val_data_l, val_labels, val_ids, valid_data = prep_dataset(
        val_data, class_col=class_col, id_col=id_col, ft_drop=ft_drop
    )

    test_data_l, _, _, test_data = prep_dataset(
        test, class_col=class_col, id_col=id_col, ft_drop=ft_drop
    )

    # Training
    gbm = lgb.train(
        params=MODEL_PARAMS,
        train_set=train_data_l,
        num_boost_round=100000,
        valid_sets=[train_data_l, val_data_l],
        early_stopping_rounds=50,
        verbose_eval=1,
    )

    # Generate confusion matrices for each iteration
    confusion_matrices_train = []

    for tree in range(0, gbm.num_trees()):
        train_pred = gbm.predict(train_data, num_iteration=tree + 1)

        train_decision = get_predicted_classes(train_pred)

        confusion_matrices_train.append(confusion_matrix(train_labels, train_decision))

    # Get the datasets for visualization
    # 1. Confusion categories dataset
    working_dict = dict()

    working_dict["Count"] = np.concatenate(confusion_matrices_train).ravel().tolist()

    working_dict["Iteration"] = [
        item + 1 for item in range(0, gbm.num_trees()) for i in range(4)
    ]
    working_dict["Class"] = class_types * gbm.num_trees()
    working_dict["Confusion_Category"] = conf_cat * gbm.num_trees()

    df_to_plot = pd.DataFrame.from_dict(working_dict)

    # 2. Positive/negative separation dataset
    sep_dict = dict()

    sep_dict["Iteration"] = [
        item + 1 for item in range(0, gbm.num_trees()) for i in range(1)
    ]
    sep_dict["Class"] = [1] * gbm.num_trees()
    sep_dict["Confusion_Category"] = ["Blank"] * gbm.num_trees()
    sep_dict["Count"] = [15] * gbm.num_trees()

    # 3. Plotting dataset
    df_to_plot = df_to_plot.append(pd.DataFrame.from_dict(sep_dict))

    # Get the indices that group the iterations into `n_segments` that minimize the intra-
        ↪ segment variances
    seg_index = time_series_segmentation(confusion_matrices_train, n_segments)

    segmentation = pd.DataFrame(
        {
            "x_min": [1] + seg_index[: n_segments - 1],
            "x_max": seg_index,
            "number": [i for i in range(1, n_segments + 1)],
        }
    )

    # Plotting
    tcm_chart = temporal_confusion_matrix(
        df=df_to_plot,
        segments=segmentation,
        title="1.0-LightGBM",
        hover="outwards",
        **features
    )

    tcm_chart.save("tcm-chart.png", scale_factor=6.0)
```

Appendix C

Predevelopment interview guidelines

Considerations:

- These interviews will be more oriented to the personal experience of each one than to try to perceive generalizations.
- These interviews target interviewees that have experience in the "traditional" DS workflow. Otherwise, through the impromptu checklist gathered at the beginning, the questions are framed according to the interviewee's experience. In addition, a more abstract agenda for Research data scientists was also defined. This agenda does not imply that no questions will be asked about Pulse, for example — it will depend on the participant's experience.
- Don't forget to ask for examples (probing questions).
- The interviews will be conducted in English or Portuguese, according to the most appropriate language for the moment. In addition, the interviews will be in person or online, depending on the availability and location of the participants.

Agenda (Customer Success edition):

1. Hello + Context.
2. Tell me about your previous experience and your current role at Feedzai.

   Get an impromptu checklist (a list to keep in mind or point out in a notebook with key points to have a basis for the rest of the interview).
3. DS workflow.
4. Model Evaluation workflow.

5. What are your goals when evaluating models? What actions do you usually take to evaluate models?

   Exemplify begins here.

6. Describe your (past and) current experience with Pulse.
7. Describe your (past and) current experience with Pulse's Model Evaluation features.
8. Pain points/Difficulties.
9. Optional: Which of Pulse's features do you usually use (this part was supported by a single slide with the list of features)?
10. Pros and cons (Pulse).
11. Was there anything missing that you expected? What are you currently doing to overcome your difficulties?

    What other options do you use?

    Why these options?

    What tasks and plots do you usually do?

    What are the drawbacks?

12. Average time split (between different options and Pulse).

    Do you think using notebooks/scripts is feasible, or would you rather have everything available in Pulse?

13. What could be done to improve Model Evaluation at Feedzai?

    Optional: If you could evaluate a model in the best way for you, what would that Model Evaluation environment look like?

14. Wrap-up summary + What haven't I asked you today that you think would be valuable for us to know? + Thanks.

Agenda (Research edition):

1. Hello + Context.
2. Tell me about your previous experience and your current role at Feedzai.

   Get an impromptu checklist.

3. DS workflow (per case, if necessary).
4. Model Evaluation workflow (per case, if necessary).

   To answer the "what" part mainly.

5. What are, in general, your goals when evaluating models?

   Exemplify begins here.

6. How do you evaluate your models (per case, if necessary)?

    What options (platforms, tools, packages, etc.) do you use?

    Why these options?

    What specific tasks and plots do you usually do? Why these options?

    What are the drawbacks?

7. Pain points/Difficulties.

8. What could be done to improve Model Evaluation at Feedzai?

    Optional: If you could evaluate a model in the best way for you, what would that Model Evaluation environment look like?

9. Wrap-up summary + What haven't I asked you today that you think would be valuable for us to know? + Thanks.

Appendix D

Logo

Figure D.1: MevaL logo. The name comes from the combination of **M**odel, **eval**uation, and **ML**. The logomark shows one of the possible nets for a cube (the typical three-dimensional geometric shape of a "box"), symbolizing an "open" ML model (ready to be checked and evaluated).

APPENDIX E

PyPI/Trove classifiers for MevaL

The list of PyPI/Trove classifiers [291] (added as package metadata) that categorize MevaL are:

- Development Status :: 4 - Beta
- Intended Audience :: Developers
- Intended Audience :: Science/Research
- License :: Other/Proprietary License
- Natural Language :: English
- Operating System :: OS Independent
- Programming Language :: Python
- Programming Language :: Python :: 3
- Programming Language :: Python :: 3 :: Only
- Programming Language :: Python :: 3.6
- Programming Language :: Python :: Implementation :: CPython
- Topic :: Scientific/Engineering :: Artificial Intelligence
- Topic :: Scientific/Engineering :: Information Analysis
- Topic :: Scientific/Engineering :: Visualization
- Topic :: Software Development :: Libraries
- Topic :: Software Development :: Libraries :: Python Modules
- Topic :: Utilities
- Typing :: Typed

Appendix F

R script to compare two colors after simulating color vision deficiency

Listing F.1: Script to simulate color vision deficiency using the *colorspace* package [348].

```r
library(colorspace)

rg <- c("#EE6C4D", "#368F8B")
rg_low_contrast <- c("#EE6C4D", "#659B5E")

png("rg.png", width=2000, height=2000, res=300)
  par(mfrow = c(2, 2), mar=c(2, 1, 4, 1))
  demoplot(rg, "bar")
  title(main="Original")
  demoplot(deutan(rg), "bar")
  title(main="Deuteranope␣(Green)")
  demoplot(protan(rg), "bar")
  title(main="Protanope␣(Red)")
  demoplot(tritan(rg), "bar")
  title(main="Tritanope␣(Blue)")
dev.off()

png("rg_low_contrast.png", width=2000, height=2000, res=300)
  par(mfrow = c(2, 2), mar=c(2, 1, 4, 1))
  demoplot(rg_low_contrast, "bar")
  title(main="Original")
  demoplot(deutan(rg_low_contrast), "bar")
  title(main="Deuteranope␣(Green)")
  demoplot(protan(rg_low_contrast), "bar")
  title(main="Protanope␣(Red)")
  demoplot(tritan(rg_low_contrast), "bar")
  title(main="Tritanope␣(Blue)")
```

```
dev.off()
```

Appendix G

JavaScript cells to save Altair charts via Vega-Embed in JupyterLab

These snippets were prepared to be possible, in Feedzai's DS environment, to save charts from a computational notebook programmatically. Since, in Altair 2.4.1, the Python package with the bindings for Selenium is required, as well as a browser (Google Chrome or Mozilla Firefox) and the respective WebDriver (ChromeDriver or geckodriver), it is not possible to use the *save* (class) instance method to get charts as images programmatically — Selenium is not supported in the DS environment and, depending on the case, the browser aspect may also be a deterrent. In this way, all charts whose cells were executed will be saved as images. Note that each snippet must be in a separate cell.

Listing G.1: First JavaScript snippet to be executed to save Altair charts via Vega-Embed in JupyterLab.

```
%%javascript

const fileType = "png"; // "png" or "svg"
let vegaActions = document.getElementsByClassName("vega-actions");

[...vegaActions].forEach((element) =>
  [...element.childNodes]
    .filter((child) => child.download.endsWith(`.${fileType}`))
    .map((download) => download.dispatchEvent(new MouseEvent("mousedown")))
);
```

Listing G.2: Second JavaScript snippet to be executed to save Altair charts via Vega-Embed in JupyterLab.

```
%%javascript

```

```
const fileType = "png"; // "png" or "svg"
let vegaActions = document.getElementsByClassName("vega-actions");

[...vegaActions].forEach((element) =>
  [...element.childNodes]
    .filter((child) => child.download.endsWith(`.${fileType}`))
    .map((download) => download.click())
);
```

Appendix H

Custom theme-based helper functions and union types for Altair

Listing H.1: Custom theme-based helper functions, and union types for all types of Altair charts and compound-only charts.

```
from json import dumps
from typing import Union

import altair as alt

UAltairChart = Union[
    alt.Chart,
    alt.LayerChart,
    alt.HConcatChart,
    alt.VConcatChart,
    alt.RepeatChart,
    alt.FacetChart,
]

UAltairCompoundChart = Union[
    alt.LayerChart, alt.HConcatChart, alt.VConcatChart, alt.RepeatChart, alt.FacetChart,
]


def get_alt_aesthetic():
    print(dumps(alt.themes.get()(), indent=4))


def set_default_alt_aesthetic():
    alt.themes.enable("default")

```

```

def get_alt_themes(verbose=True):
    if verbose:
        print(f"Current␣theme:␣{repr(alt.themes.active)}")
    return alt.themes.names()


def enable_alt_aesthetic(theme, **options):
    if theme in alt.themes.names():
        alt.themes.enable(theme, **options)
    else:
        raise ValueError(f"The␣{repr(theme)}␣theme␣is␣not␣available␣in␣MevaL.")


def update_alt_aesthetic(width=300, height=300):
    current_theme = alt.themes.active.replace("_updated", "")
    new_theme = f"{current_theme}_updated"

    alt.themes.register(new_theme, THEMES.get(current_theme))
    enable_alt_aesthetic(new_theme, width=width, height=height)
```

Appendix I

Generate high-resolution images from static or interacted Altair charts

Although it is possible to generate PNG images (and scalable SVG images) with variable size/resolution (controlled through the *scale_factor* argument) using the *save* (class) instance method available in Altair (or directly via altair_saver [299] nowadays), these images correspond to static representations of the charts previously created (this method takes a *screenshot* to the initial state of the rendered chart so to speak). In this way, a possible manual and agnostic procedure to generate high-resolution PNG images (using Google Chrome), with constant margins and that capture a representation after user interaction (it is possible to capture, for example, a tooltip and the appearance of the chart when hovered), is as follows:

1. Save the Altair chart to an HTML file (which will contain the Vega-Lite counterpart). This can be done through the following snippet (the SVG renderer is used to ensure better image quality): *chart.save("chart.html", embed_options={"renderer": "svg"})*.
2. Open the HTML file in Google Chrome.
3. Open Chrome DevTools (**Control+Shift+C** or **Command+Option+C** on Mac).
4. Click on the **Toggle Device Toolbar** button, then on the **Device** dropdown menu, and finally select **Edit**. This is the first of three steps to create a custom device with the desired resolution in order to generate high-resolution screenshots [27]. Thus, this step and the following two steps only need to be performed the first time.
5. In the **Settings** panel, click on the **Add custom device** button and then enter a name, width, and height for the new device (*Screenshot*, 1920 and 1080, respectively, for example). In addition, also fill in the device pixel ratio field (with the value 6, for example).

6. In the **Device** dropdown menu, select the newly created device. Close the **Settings** panel as well.
7. In the **Elements** panel, select the DOM node that contains only the chart or the parent node that also contains the Vega-Embed button (both are <*div*> elements). This step is important to ensure that the screenshot contains only the chart of interest with a small margin around it.
8. If necessary, to capture the chart state after an interaction, trigger the desired event. An example is hovering the chart to show a tooltip.
9. Open the Command Menu using the associated shortcut (**Control+Shift+P** or **Command+Shift+P** on Mac) [26] and search for *Capture node screenshot*. Select the option with this name to save a screenshot as a PNG image. This screenshot will correspond to the desired chart image.

This procedure was tested on version 86.0.4240.111 of Google Chrome.

APPENDIX J

PANDAS OPTIONS RECONFIGURED BY MEVAL

Listing J.1: Function responsible for setting certain pandas options for MevaL when imported.

```python
import pandas as pd


def set_pd_options():
    options = {
        "display": {
            "max_columns": None, # No limit on the number of columns so that the DataFrame
                ↪ is not horizontally truncated
            "max_colwidth": 30, # Smaller maximum width (in characters) for the columns in
                ↪ order to balance the total width with the above option
            "show_dimensions": False, # Don't show the dimensions (secondary information)
                ↪ of the DataFrame at the bottom
        },
        "mode": {
            "chained_assignment": "raise" # Raise an exception if the user tries to use
                ↪ chained assignment
        },
    }

    for category, option in options.items():
        for op, value in option.items():
            pd.set_option(f"{category}.{op}", value)
```

Appendix K

Script to prepare the demo dataset and train a baseline model

Listing K.1: Python script to prepare the demo dataset, based on the (labeled) training datasets of the IEEE-CIS Fraud Detection Kaggle competition [119], to apply minimal (and oversimplified compared to the procedures followed at Feedzai) feature engineering, and to train/*test* (in a time-aware fashion) a baseline LightGBM model [132] with the default hyperparameters [66, 173, 343]. In addition to saving a dataset with a subset of features that function as breakdown fields (fields that divide the dataset into several subsets with individual interest), with the target field (*isFraud*) and each score predicted for 118,108 transactions (*fraud_score*), it also stores the split-based (according to the number of times each feature is used in the model) [132] absolute and relative/percentage FI values. The applied feature engineering/preprocessing consists of removing columns with only one value (excluding null values), columns with more than 90% of their values as null values, and columns where one of their values appears more than 90% of the time (basically, columns with variability close to zero are removed) [173]. The three columns for devices (*DeviceInfo* and *id_30*) and the browser (*id_31*) where online transactions are made are split so that there are two distinct columns, one for the device model and one for the version, approximately (in the case of the browser column, it is only kept the browser name) [66, 343]. Although this data set is not 100% representative of those used at Feedzai, it works as a proxy given that it is made up of online (card-not-present) transactions binarily labeled in an imbalanced way, covers a period of six months (the time component is present), has semantic fields, such as the transaction amount field, categorical (or breakdown) fields, such as the device type field, more than 300 original features, and also a significant size manageable on a single machine. Since the *TransactionDT* field is a *timedelta* field (a duration or the difference between two dates) from a given reference date, this field was converted to a *datetime* field (*Timestamp*) using an arbitrary start date, in order to simulate a *timestamp* field. This script also contains a helper function (*save_split_value_histogram()*) to generate the histogram in Figure 2.9 and another one (*distance_constrained_shuffle()*) [117] to shuffle the FI dataset in a distance-constrained fashion (another option could be training another model with a different seed, for example) in order to obtain another dataset for comparison purposes (this particular method was applied to avoid huge jumps between the values of each feature, i.e., a completely random shuffle without considering the distribution of the original FI values).

```python
import os
import random
from datetime import datetime

import lightgbm as lgb
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from tqdm import tqdm

```

```python
RAW_DATA_PATH = "./raw_data/"
DATA_PATH = "./data/"
SEED = 0

MODEL_PARAMS = {
    "objective": "binary",
    "boosting_type": "gbdt",
    "verbose": 1,
    "seed": SEED,
}


def seedify(seed):
    random.seed(seed)
    os.environ["PYTHONHASHSEED"] = str(seed)
    np.random.seed(seed)


def get_columns_to_drop(df, target_column="isFraud"):
    one_value_cols = [
        col for col in df.columns if df[col].nunique() <= 1
    ] and col != target_column

    many_null_cols = [
        col
        for col in df.columns
        if df[col].isnull().sum() / df.shape[0] > 0.9 and col != target_column
    ]

    big_top_value_cols = [
        col
        for col in df.columns
        if df[col].value_counts(dropna=False, normalize=True).values[0] > 0.9
        and col != target_column
    ]

    cols_to_drop = list(set(one_value_cols + many_null_cols + big_top_value_cols))

    return cols_to_drop


def timedelta2datetime(
    df, td_col="TransactionDT", dt_col="Timestamp", start_date=datetime(2020, 1, 1),
):
    df[dt_col] = pd.Timestamp(start_date)
    df["Diff"] = df[td_col] - df[td_col].iloc[0]
    df[dt_col] = df[dt_col] + pd.to_timedelta(df["Diff"], unit="s")

    return df.drop("Diff", axis=1)

```

```python

def split_devices(df, version_cols, no_version_cols):
    for col in version_cols:
        df[col] = df[col].fillna("unknown_device").str.lower()
        df[f"{col}_device"] = df[col].apply(
            lambda x: "".join([i for i in x if i.isalpha()])
        )
        df[f"{col}_version"] = df[col].apply(
            lambda x: "".join([i for i in x if i.isnumeric()])
        )

    for col in no_version_cols:
        df[col] = df[col].fillna("unknown_device").str.lower()
        df[f"{col}_device"] = df[col].apply(
            lambda x: "".join([i for i in x if i.isalpha()])
        )

    return df.drop(version_cols + no_version_cols, axis=1)


def save_split_value_histogram(model, feature, filename="split_value_hist"):
    fig, ax = plt.subplots(1, 1, figsize=(10, 5), dpi=300)

    ax = lgb.plot_split_value_histogram(
        model, feature=feature, bins="auto", ax=ax, grid=False
    )

    fig.savefig(f"{filename}.png", bbox_inches="tight")


def distance_constrained_shuffle(df, distance, cols_to_shuffle, sort_by):
    df = df.copy().sort_values(sort_by, ascending=False).reset_index(drop=True)
    df_to_shuffle = df[cols_to_shuffle]

    # `random.random()` -> [0.0, 1.0)
    index_shuffled = [
        df_idx
        for idx, df_idx in sorted(
            enumerate(df_to_shuffle.index),
            key=lambda enum_tpl: enum_tpl[0] + (distance + 1) * random.random(),
        )
    ]

    df_to_shuffle = df_to_shuffle.reindex(index=index_shuffled).reset_index(drop=True)

    df[cols_to_shuffle] = df_to_shuffle[cols_to_shuffle]

    return df


```

```python
def make_test_predictions(train_df, test_size, target, timestamp, id_, lgb_params):
    for col in tqdm(
        train_df.select_dtypes(include="object").columns, desc="Ordinal␣encoding"
    ):
        train_df[col] = train_df[col].fillna("unseen_before_label")
        train_df[col] = train_df[col].astype(str)

        le = LabelEncoder()
        train_df[col] = le.fit_transform(train_df[col])

        train_df[col] = train_df[col].astype("category")

    train_df = timedelta2datetime(train_df)

    # For time-aware split
    X = train_df.sort_values(timestamp).drop([target, id_], axis=1)
    y = train_df.sort_values(timestamp)[target]

    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=test_size, shuffle=False
    )

    clf = lgb.LGBMClassifier(**lgb_params) # Scikit-learn API
    clf.fit(X_train.drop([timestamp, "Timestamp"], axis=1), y_train)
    test_scores = clf.predict_proba(X_test.drop([timestamp, "Timestamp"], axis=1))[:, 1]

    scores_df = X_test.copy()

    scores_df["isFraud"] = y_test
    scores_df["fraud_score"] = test_scores

    return clf, scores_df


if __name__ == "__main__":
    seedify(SEED)

    train_identity = pd.read_csv(f"{RAW_DATA_PATH}train_identity.csv")
    train_transaction = pd.read_csv(f"{RAW_DATA_PATH}train_transaction.csv")

    train = pd.merge(train_transaction, train_identity, on="TransactionID", how="left")

    cols_to_drop = get_columns_to_drop(train)
    train = train.drop(cols_to_drop, axis=1)
    train = split_devices(
        train, ["DeviceInfo", "id_30"], ["id_31"]
    ) # Device and browser

    clf, scores_df = make_test_predictions(
        train, 0.20, "isFraud", "TransactionDT", "TransactionID", MODEL_PARAMS
)

save_split_value_histogram(clf, "TransactionAmt")

scores_df[
    [
        "TransactionDT",
        "Timestamp",
        "TransactionAmt",
        "ProductCD",
        "card4",
        "card6",
        "DeviceType",
        "id_30_device",
        "isFraud",
        "fraud_score",
    ]
].to_csv(f"{DATA_PATH}evaluation_kaggle.csv", index=False)

fi_df = pd.DataFrame(
    {
        "Feature": scores_df.drop(
            ["TransactionDT", "Timestamp", "isFraud", "fraud_score"], axis=1
        ).columns,
        "Absolute␣Importance": clf.feature_importances_,
        "Relative␣Importance": clf.feature_importances_
        / sum(clf.feature_importances_),
    }
)

fi_df.to_csv(f"{DATA_PATH}fi_data_kaggle.csv", index=False)

distance_constrained_shuffle(
    fi_df, 10, ["Absolute␣Importance", "Relative␣Importance"], "Absolute␣Importance"
).to_csv(f"{DATA_PATH}fi2_data_kaggle.csv", index=False)
```

Appendix L

Performance and memory consumption analysis of MevaL's data processing pipeline

This analysis was conducted only for the part dedicated to processing the main dataset in a local JupyterLab environment on a laptop whose technical specifications can be found in Appendix M. Processing one or two FI raw datasets, on top of pandas, is negligible, given its small size, up to 1,000 rows and 10 columns as expected, and the simple operations (to generate new columns or to optionally reshape the dataset) involved. The *SparkSession*, the *entry point* for using PySpark, was initialized with the default configuration.

Listing L.1: Script to plot the resulting execution times (Figure 3.10), and assemble Table L.1 and Table L.2 based on the output (persisted as JSON files) of the corresponding IPython [220]/memory_profiler magic cell commands (*%%timeit* and *%%memit*, respectively).

```
import json
import math

import altair as alt
import pandas as pd

COLORS = {
    "black": "#44475A",
    "lgray": "#EBEBEB",
    "gray": "#858585",
    "white": "#FFFFFF",
    "green": "#368F8B",
    "red": "#EE6C4D",
}

NUM_FORMATTER = "{:,}".format
```

Table L.1: Mean execution times (and respective standard deviations) considering 20 (independent) repeats (and one execution per repeat) of the function (Python expression) responsible for (fully) processing the main dataset (the training/test dataset with the scores returned by a model for each instance). The dataset used is the one described in the Dataset section and Appendix K, corresponding to the first line of this table. Subsequent datasets were obtained through sampling with replacement in order to simulate larger datasets and see the (linear) trend in execution time (visible in the chart in Figure 3.10). On average, it takes less than 10 minutes to run the data processing pipeline locally on a relatively modern laptop (Appendix M) for a dataset with over a million rows, a totally tolerable run time. These performance values (in seconds) were collected with the following IPython built-in magic cell command [220, 298]: *%%timeit -o -r 20*.

Relative Size	# Rows	Mean (s)	Standard Deviation (s)
1x	118,108	65.95925090704114	1.0088734232719023
2x	236,216	116.96639213609743	2.5616991301031438
3x	354,324	219.22396667789434	4.040200567677255
4x	472,432	266.14367742915056	3.2281893261380463
5x	590,540	313.1284446614096	2.6885876901656593
6x	708,648	363.56116795226114	1.803603312141961
7x	826,756	412.5953572864528	1.106972649273104
8x	944,864	462.86697524811024	2.165084251789841
9x	1,062,972	516.18207561351	4.019599758861626
10x	1,181,080	572.9328743875667	6.74338383796817

Table L.2: Estimated memory usage (reported by the *Increment (MiB)* column) when running (only once) the (full) data processing pipeline for the main dataset (the training/test dataset with the scores returned by a model for each instance). The dataset used is the one described in the Dataset section and Appendix K, corresponding to the first line of this table. Subsequent datasets were obtained through sampling with replacement in order to simulate larger datasets and its impact on memory usage. The memory usage values (which include the memory usage of any spawned child process) vary approximately between 1,256 mebibytes (1,317.011 MB) for a dataset with about 120,000 lines and 2,068 mebibytes (2168.455 MB) for a similar dataset with 10 times more rows (these values are fine for local environments/laptops). In addition, they were computed using the memory_profiler package [218, 298], namely the following command: *%%memit -c -o*.

Relative Size	# Rows	Peak Memory (MiB)	Increment (MiB)
1x	118,108	1,532.69921875	1,256.0
2x	236,216	2,068.2265625	1,790.6328125
3x	354,324	2,153.32421875	1,875.234375
4x	472,432	2,175.453125	1,897.3359375
5x	590,540	2,299.7109375	2,021.578125
6x	708,648	2,136.17578125	1,858.83984375
7x	826,756	2,195.92578125	1,917.21484375
8x	944,864	2,307.515625	2,028.4921875
9x	1,062,972	2,358.7890625	2,079.7421875
10x	1,181,080	2,347.6484375	2,068.56640625

```python
N_ROWS = pd.read_csv("raw_thesis.csv").shape[0]


def persist_itresult(obj, filename):
    """Save the output of the %%memit/%%timeit IPython magic command as a JSON file."""
    with open(f"{filename}.json", "w") as outfile:
        json.dump(
            obj, outfile, default=lambda x: x.__dict__, ensure_ascii=False, indent=4
        )


def theme(font="Fira␣Sans"):
    return {
        "config": {
            "title": {"font": font, "color": COLORS["black"]},
            "axisX": {"labelAngle": 0,},
            "axisY": {"titleAngle": 0, "titleAlign": "left", "titleY": -5},
            "axis": {
                "labelFont": font,
                "titleFont": font,
                "gridColor": COLORS["lgray"],
                "labelColor": COLORS["black"],
                "tickColor": COLORS["black"],
                "titleColor": COLORS["black"],
                "domainColor": COLORS["black"],
            },
            "title": {
                "subtitleFont": font,
                "subtitleColor": COLORS["black"],
                "color": COLORS["black"],
                "font": font,
            },
            "view": {
                "fill": COLORS["white"],
                "stroke": COLORS["lgray"],
                "strokeWidth": 1,
            },
            "rule": {"color": COLORS["black"]},
            "line": {"color": COLORS["lgray"]},
            "text": {"font": font, "color": COLORS["black"], "fontSize": 10},
        },
    }


def time_perf_chart(data, xvar, yvar_mean, yvar_lb, yvar_ub, title, subtitle):
    title_params = {"text": title, "subtitle": subtitle, "anchor": "start"}
    y_values = list(range(0, 600 + 1, 60))

    base = alt.Chart(data, width=300, height=300, title=title_params)

    points = base.mark_point(
        filled=True,
        opacity=1,
        size=20,
        stroke=COLORS["white"],
        strokeWidth=0.25,
        color=COLORS["green"],
    ).encode(
        x=alt.X(
            f"{xvar}:O",
            axis=alt.Axis(
                title="Relative␣size␣(#␣rows)",
                labelExpr="datum.value␣==␣1␣?␣'Base'␣:␣datum.value␣+␣'x'",
            ),
        ),
        y=alt.Y(
            f"{yvar_mean}:Q",
            axis=alt.Axis(
                title=None,
                tickCount=len(y_values),
                values=y_values,
                labelExpr="(datum.value␣/␣60)␣+␣'␣min.'",
            ),
        ),
    )

    error_bars = base.mark_rule().encode(
        x=alt.X(f"{xvar}:O"), y=alt.Y(f"{yvar_lb}:Q",), y2=alt.Y2(f"{yvar_ub}:Q"),
    )

    line = base.mark_line(size=1).encode(
        x=alt.X(f"{xvar}:O"), y=alt.Y(f"{yvar_mean}:Q")
    )

    return line + error_bars + points


def average(obj, key="timings"):
    """Based on the IPython implementation."""
    return math.fsum(obj[key]) / len(obj[key])


def stdev(obj, mean, key="timings"):
    """Based on the IPython implementation."""
    return (math.fsum([(x - mean) ** 2 for x in obj[key]]) / len(obj[key])) ** 0.5


def get_peak_mem_and_inc(obj, mem_usage="mem_usage", baseline="baseline"):
    """Based on the memory_profiler implementation."""
    peak_mem = max(obj[mem_usage])
    inc = peak_mem - obj[baseline]

    return peak_mem, inc


if __name__ == "__main__":
    alt.themes.register(theme, theme)
    alt.themes.enable(theme)
    pd.options.display.float_format = NUM_FORMATTER

    time_stats = []
    mem_stats = []

    for i in range(1, 11):
        with open(f"time_20Runs_{i}xDataset.json") as ifile:
            data = json.load(ifile)

            mean = average(data)
            std = stdev(data, mean)

            lbound = mean - std
            ubound = mean + std

            time_stats.append([i, N_ROWS * i, mean, std, lbound, ubound])

        with open(f"memory_child_1Run_{i}xDataset.json") as ifile:
            data = json.load(ifile)

            peak_mem, inc = get_peak_mem_and_inc(data)

            mem_stats.append([i, N_ROWS * i, peak_mem, inc])

    time_df = pd.DataFrame(
        time_stats,
        columns=[
            "relative_size",
            "rows",
            "mean",
            "stdev",
            "lower_bound",
            "upper_bound",
        ],
    )

    mem_df = pd.DataFrame(
        mem_stats, columns=["relative_size", "rows", "peak_mem", "inc",],
    )

    chart = time_perf_chart(
        time_df,
        "relative_size",
        "mean",
        "lower_bound",
        "upper_bound",
        "Execution␣time␣of␣the␣data␣processing␣pipeline",
        "Main␣dataset",
    )

    chart.save("time_perf_chart.png", scale_factor=6.0)

    time_df["relative_size"] = time_df["relative_size"].astype(str) + "x"
    mem_df["relative_size"] = mem_df["relative_size"].astype(str) + "x"

    time_df.to_latex(
        "table8.tex",
        index=False,
        columns=["relative_size", "rows", "mean", "stdev"],
        header=["Relative␣Size", "#␣Rows", "Mean␣(s)", "Standard␣Deviation␣(s)"],
        label="tab:time-perf",
        caption="",
        formatters={"rows": NUM_FORMATTER},
    )

    mem_df.to_latex(
        "table9.tex",
        index=False,
        header=["Relative␣Size", "#␣Rows", "Peak␣Memory␣(MiB)", "Increment␣(MiB)"],
        label="tab:mem-perf",
        caption="",
        formatters={"rows": NUM_FORMATTER},
    )
```

Appendix M

Laptop technical specification

The laptop used during this project has the following main characteristics (platform's identifying data) [20]:

- **Model Identifier**: MacBookPro14,1.
- **Processor (Model/Number)**: 2.5 GHz Dual-Core Intel Core i7 (i7-7660U).
- **L2 Cache (per Core)**: 256 KB.
- **L3 Cache**: 4 MB.
- **Memory**: 16 GB of 2133 MHz LPDDR3.
- **Graphics (Video RAM)**: Intel Iris Plus Graphics 640 (1536 MB).
- **Storage**: 512GB SSD.
- **Operating System (Version)**: macOS (10.15.5 Catalina).
- **Kernel (Version)**: Darwin (19.5.0).
- **Display Size**: 13.3 inches (diagonal).

The environment variable/line *export OBJC_DISABLE_INITIALIZE_FORK_SAFETY=YES* was added to the *.bash_profile* file in order to be able to use PySpark (including UDFs) locally at full power on macOS Catalina (Apple introduced some security changes at the multithreading level in macOS High Sierra) [19, 116, 155].

Appendix N

Type hints for the *__init__()* method of the *EvaluationManager* class.

1. ***target_field***: *str*.
2. ***target_value***: *Union[str, int]*.
3. ***score_field***: *str*.
4. ***timestamp_field***: *str*.
5. ***performance_metrics***: *Union[str, Sequence[str]]*.
6. ***score_files***: *Optional[Dict[str, Union[str, Sequence[str]]]]*.
7. ***fi_files***: *Optional[Dict[str, Union[str, Sequence[str]]]]*.
8. ***metrics_dfs***: *Optional[Union[pd.DataFrame, pyspark.sql.DataFrame]]*.
9. ***agg_scores_dfs***: *Optional[Union[pd.DataFrame, pyspark.sql.DataFrame]]*.
10. ***fi_dfs***: *Optional[Union[pd.DataFrame, pyspark.sql.DataFrame]]*.
11. ***breakdowns***: *Optional[Union[str, Sequence[Union[str, Dict[str, Union[str, Sequence[str]]]]]]]*.
12. ***non_target_value***: *Optional[Union[str, int]]*.
13. ***cost_field***: *Optional[str]*.
14. ***truncate_score_decimals***: *Optional[int]*.
15. ***compute_cost_metrics***: *bool*.
16. ***compute_ci_for_metrics***: *bool*.
17. ***keep_confusion_categories***: *bool*.
18. ***timestamp_fmt***: *str*.
19. ***threshold_listing_strategy***: *str*.

20. ***thresholds_to_analyze***: *Optional[Union[float, Sequence[float]]]*.
21. ***feature_cols***: *Union[str, Sequence[str]]*.
22. ***fi_cols***: *Union[str, Sequence[str]]*.
23. ***score_files_sep***: *str*.
24. ***fi_files_sep***: *str*.

It is assumed that a *Sequence* will be, in practice, a *List* or a *Tuple*.

Appendix O

Script to generate a label bar chart

Listing O.1: Script to generate the label bar chart for the test dataset (Appendix K) used to exemplify the charts implemented in MevaL.

```python
import altair as alt
import pandas as pd

COLORS = {
    "black": "#44475A",
    "lgray": "#EBEBEB",
    "gray": "#858585",
    "white": "#FFFFFF",
    "green": "#368F8B",
    "red": "#EE6C4D",
}


def theme(font="Fira␣Sans"):
    return {
        "config": {
            "title": {"font": font, "color": COLORS["black"]},
            "axisX": {"labelAngle": 0,},
            "axisY": {"titleAngle": 0, "titleAlign": "left", "titleY": -5},
            "axis": {
                "labelFont": font,
                "titleFont": font,
                "gridColor": COLORS["lgray"],
                "labelColor": COLORS["black"],
                "tickColor": COLORS["black"],
                "titleColor": COLORS["black"],
                "domainColor": COLORS["black"],
            },
            "title": {
                "subtitleFont": font,
                "subtitleColor": COLORS["black"],
                "color": COLORS["black"],
                "font": font,
            },
            "view": {
                "fill": COLORS["white"],
                "stroke": COLORS["lgray"],
                "strokeWidth": 1,
            },
            "rule": {"color": COLORS["black"]},
            "line": {"color": COLORS["lgray"]},
            "text": {"font": font, "color": COLORS["black"], "fontSize": 10},
        },
    }


def label_bar_chart(
    df,
    xvar="index",
    yvar_cnt="isFraud",
    yvar_pct="percentage",
    colorvar="color",
    n_ticks=6,
):
    n_rows = df[yvar_cnt].sum()
    yvals = [(n_rows * (y / 100)) for y in range(0, 101, 10)]

    base = alt.Chart(df, width=300, height=300)

    count = base.mark_bar().encode(
        x=alt.X(f"{xvar}:N", title="Count␣←␣|␣Labels␣|␣→␣Percentage"),
        y=alt.Y(
            f"{yvar_cnt}:Q",
            title=None,
            axis=alt.Axis(values=yvals, grid=True, tickCount=n_ticks),
            scale=alt.Scale(nice=False, domain=[0, n_rows]),
        ),
    )

    percentage = base.mark_bar().encode(
        x=alt.X(f"{xvar}:N"),
        y=alt.Y(
            f"{yvar_pct}:Q",
            axis=alt.Axis(grid=True, tickCount=n_ticks, format="%"),
            title=None,
        ),
        color=alt.Color(f"{colorvar}:N", scale=None),
    )
```

```python

    text = base.mark_text(dy=-5).encode(
        x=alt.X(f"{xvar}:N"),
        y=alt.Y(
            f"{yvar_pct}:Q",
            axis=alt.Axis(
                domain=False, grid=False, ticks=False, labels=False, title=None
            ),
        ),
        text=alt.Text(f"{yvar_pct}:Q", format=".2%"),
    )

    return alt.layer(count, percentage, text).resolve_scale(y="independent")


if __name__ == "__main__":
    alt.themes.register(theme, theme)
    alt.themes.enable(theme)

    data = pd.read_csv("raw_thesis.csv")

    data_to_plot = data["isFraud"].value_counts().to_frame().reset_index()
    data_to_plot["percentage"] = data_to_plot["isFraud"] / data_to_plot["isFraud"].sum()
    data_to_plot["color"] = [COLORS["green"], COLORS["red"]]

    chart = label_bar_chart(data_to_plot)

    chart.save("label_bar_chart.png", scale_factor=6.0)
```

Appendix P

Helper function to save an Altair chart as an HTML file and include the tooltip theme

Listing P.1: The snippet with the monkey patched Jinja HTML template [252] and the helper function to save an Altair chart as an HTML file with MevaL's tooltip theme. Basically, this function adapts the Jinja HTML template used by Altair [297] (2.4.1) to include some extra CSS properties and executes the *save()* method of an arbitrary chart with an argument that selects the desired tooltip theme through Vega-Embed [198]. In the end, the original template is *restored* (using Python's *importlib*).

```
import importlib

import altair as alt
import jinja2

MONKEY_HTML_TEMPLATE = jinja2.Template(
    """
{%- if fullhtml -%}
<!DOCTYPE html>
<html>
<head>
{%- endif %}
  <style>
    .vega-actions a {
        margin-right: 12px;
        color: #757575;
        font-weight: normal;
        font-size: 13px;
    }
    .error {
        color: red;
    }
    #vg-tooltip-element.vg-tooltip.main-theme {
        color: #2f2f2f;
        border: 1px solid #ebebeb;
        font-family: Arial;
        font-size: 11px;
      }
    #vg-tooltip-element.vg-tooltip.main-theme td.key {
        color: #2f2f2f;
        font-weight: bold;
    }
  </style>
{%- if not requirejs %}
  <script type="text/javascript" src="{{ base_url }}/vega@{{ vega_version }}"></script>
  {%- if mode == 'vega-lite' %}
  <script type="text/javascript" src="{{ base_url }}/vega-lite@{{ vegalite_version }}"></script>
  {%- endif %}
  <script type="text/javascript" src="{{ base_url }}/vega-embed@{{ vegaembed_version }}"></script>
{%- endif %}
{%- if fullhtml %}
{%- if requirejs %}
<script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/require.js/2.3.6/require.min.js"></script>
<script>
requirejs.config({
    "paths": {
        "vega": "{{ base_url }}/vega@{{ vega_version }}?noext",
        "vega-lib": "{{ base_url }}/vega-lib?noext",
        "vega-lite": "{{ base_url }}/vega-lite@{{ vegalite_version }}?noext",
        "vega-embed": "{{ base_url }}/vega-embed@{{ vegaembed_version }}?noext",
    }
});
</script>
{%- endif %}
</head>
<body>
{%- endif %}
  <div id="{{ output_div }}"></div>
  <script>
    {%- if requirejs %}
    {%- if not fullhtml %}
    requirejs.config({
        "paths": {
            "vega": "{{ base_url }}/vega@{{ vega_version }}?noext",
            "vega-lib": "{{ base_url }}/vega-lib?noext",
            "vega-lite": "{{ base_url }}/vega-lite@{{ vegalite_version }}?noext",
            "vega-embed": "{{ base_url }}/vega-embed@{{ vegaembed_version }}?noext",
        }
    });
    {%- endif %}
    require(['vega-embed'], function(vegaEmbed){
    {%- endif %}
      var spec = {{ spec }};
      var embedOpt = {{ embed_options }};

      function showError(el, error){
          el.innerHTML = ('<div class="error" style="color:red;">'
                         + '<p>JavaScript Error: ' + error.message + '</p>'
                         + "<p>This usually means there's a typo in your chart specification. "
                         + "See the javascript console for the full traceback.</p>"
                         + '</div>');
          throw error;
      }
      const el = document.getElementById('{{ output_div }}');
      vegaEmbed("#{{ output_div }}", spec, embedOpt)
        .catch(error => showError(el, error));
    {%- if requirejs %}
    });
    {%- endif %}

  </script>
{%- if fullhtml %}
</body>
</html>
{%- endif %}
"""
)


def save_as_html(chart, filename):
    alt.utils.html.HTML_TEMPLATE = MONKEY_HTML_TEMPLATE

    try:
        chart.save(
            filename, embed_options={"renderer": "svg", "tooltip": {"theme": "main"}}
        )
    except AttributeError:
        chart.show().save(
            filename, embed_options={"renderer": "svg", "tooltip": {"theme": "main"}}
        )

    importlib.reload(alt.utils.html)
```

Appendix Q

Postdevelopment interview guidelines

Agenda:

1. Quick introduction to the subject of the master's thesis and to MevaL (if necessary), as well as the motivation for this feedback round (obtain input from potential future users and *validate* what has been done so far).

   Mention that they can interrupt whenever they want. Interlude the expository part with pauses/general questions.

2. Present a summary of the package development so far (first version). Some notes:

   Present the overview of the available features (divided into different subpackages) and the workflow designed with the help of the *EvaluationManager* class.

   The features take advantage of two types of raw data: scored instances and FI data. These data sources are processed (if necessary since the user can add preprocessed DataFrames that respect a given schema) in three types of data: performance metrics, aggregate scores, and FI.

   Give an overview of the data layer (it is not our focus, but we tried to have a first version of the necessary operations).

   Features have workable but also customizable defaults.

   Ask for questions.

3. Show the list of features, as well as an exemplary chart/table for each one of them (keep it simple).

   Ask for questions/comments/suggestions.

   In your current project, how would you use these features?

Are there any features/types of features that have caught your attention? Which ones and why? Show different versions in cells ready to run according to the conversation (if necessary).

If not all features are covered, there is no problem — the most important thing is to validate the general structure and show concrete things to try to get more concrete feedback than in past interviews.

Change the list order between conversations.

When trying to compare two distributions in a histogram, what type of histogram do you normally use? Why?

Have you applied any type of style to the tables you check on notebooks? If not, do you imagine any scenario in which this would help?

Do you usually customize the aesthetics of your charts or do you usually use the standard options? If so, what do you usually do?

4. Open questions: What features do you find most useful? What would you like to see in the next version of the package?
5. Wrap-up summary + What haven't I asked you today that you think would be valuable for us to know? + Thanks.

Appendix R

Internal project interview guidelines

Goal: Gather information about the internal initiatives that are currently using MevaL in order to write a few paragraphs about them in the master's thesis.
Agenda:

1. Share the motivation for this meeting.
2. First of all, what is the project about? Can you summarize it in a few words?
3. What is the motivation for this project?

   How did it come about and why?

   What are you trying to cover with this project?

   What are the main success criteria?
4. Can you elaborate on the format and main (technical) characteristics of this project?

   What technologies are you using (in addition to MevaL)?

   What is the expected output?

   What is the target group?

   What is the use case?
5. What is the current status of the project? And the next steps?
6. Have any end-users used the project? If so, is there any feedback?
7. Why are you using MevaL?
8. How are you using MevaL? What features are you using?
9. So far, did you use any alternative to implement some functionality that you expected to cover with MevaL?

10. Based on your experience, what are the main 2-3 positive aspects of MevaL? And the negative ones? What would you like to see in the package in the future?
11. Clarify the next steps and ask if I can get back in touch to clarify potential doubts.

Appendix S

Script to generate the coauthorship network

Listing S.1: Python (3.5.9) script to generate the coauthorship network (Figure 2.15) based on a sample of 46 papers and 186 authors discussed in the Literature Review section. The Tethne package [219], a bibliometrics package, was used to obtain a graph describing coauthorship from the Resource Description Framework (RDF) file downloaded from Zotero, an open-source reference management software, with all relevant papers (this graph was persisted as a GraphML file). A variable and a helper function from the Tethne package were monkey patched in order to update a FOAF (a machine-readable ontology) property (*givenName*) [40] in the Zotero RDF file parser and to add isolated nodes (the nodes that correspond to solo authors) to the generated graph. The Python interface for igraph (python-igraph package) [52] was used to hold/compute the data (the labels with the names of the authors, the vertices, the edges, and the number of papers of each author) to be plotted. It was chosen instead of the NetworkX package [95] simply because of its simpler data structures to manipulate and prepare some Python lists with the coordinates for Plotly, the interactive visualization package used [60, 120, 151]. The layout algorithm for positioning the nodes was the Fruchterman-Reingold force-directed algorithm [52] (in practice, the graph is plotted as a scatterplot and the layout algorithm allows the calculation of X and Y coordinates).

```
import os
import random

import chart_studio.plotly as py
import igraph as ig
import rdflib
import tethne
from plotly.graph_objects import Figure, Layout, Scatter, layout
from tethne.networks.authors import coauthors
from tethne.readers.zotero import FOAF, read
from tethne.writers.graph import to_graphml

SEED = 1234
FILENAME = "46papers"
LABEL_ID = "id"


def seedify(seed):
    random.seed(seed)
    os.environ["PYTHONHASHSEED"] = str(seed)


def zotero2coauthors(path):
    # Note: If necessary, change the `rdf:Description` tag to `bib:Article`.
    corpus = read(path, corpus=True, index_fields=["authors"])

    G = coauthors(corpus, edge_attrs=[])

    return G


def _monkey_generate_graph(
    graph_class,
    pairs,
    node_attrs={},
    edge_attrs={},
    min_weight=1,
    keep_isolated_nodes=True,
):
    graph = graph_class()
    for combo, count in list(pairs.items()):
        if count >= min_weight:
            if combo in edge_attrs:
                attrs = edge_attrs[combo]
            else:
                attrs = {}
            graph.add_edge(combo[0], combo[1], weight=count, **attrs)

    for k, attrs in list(node_attrs.items()):
        if k in graph.nodes:
            graph.nodes[k].update(attrs)
        else:
            if keep_isolated_nodes:
                graph.add_node(k, **attrs)
    return graph


def plot_coauthorship_graph(Xn, Yn, Xe, Ye, labels, size):
    trace1 = Scatter(
        x=Xe, y=Ye, mode="lines", line=dict(color="#D0D0D0", width=1), hoverinfo="none"
    )
    trace2 = Scatter(
        x=Xn,
        y=Yn,
        mode="markers",
        marker=dict(
            symbol="circle-dot",
            size=[s * 5 for s in size],
            color="#44475A",
            line=dict(color="#44475A", width=0.5),
        ),
        text=labels,
        hovertext=n_papers,
        hovertemplate="<b>Author</b>:␣%{text}"
        + "<br><b>#␣Papers</b>:␣%{hovertext:~}"
        + "<extra></extra>",
    )

    axis = dict(
        showline=False,
        zeroline=False,
        showgrid=False,
        showticklabels=False,
        title="",
        automargin=False,
    )

    layout_ = Layout(
        font=dict(size=12),
        showlegend=False,
        autosize=False,
        width=800,
        height=800,
        xaxis=layout.XAxis(axis),
        yaxis=layout.YAxis(axis),
        margin=layout.Margin(l=0, r=0, b=0, t=0,),
        hovermode="closest",
        plot_bgcolor="#FFFFFF",
    )

    data = [trace1, trace2]
    fig = Figure(data=data, layout=layout_)
    return fig


if __name__ == "__main__":
    # Monkey patches:
    tethne.readers.zotero.FORENAME_ELEM = rdflib.URIRef(
        FOAF + "givenName"
    ) # Fix FOAF property
    tethne.networks.base._generate_graph = (
        _monkey_generate_graph # Keep isolated nodes (papers with a single author)
    )

    # Graph:
    G_to_save = zotero2coauthors(FILENAME)
    to_graphml(G_to_save, "{}.graphml".format(FILENAME))

    G_to_plot = ig.Graph.Read_GraphML("{}.graphml".format(FILENAME))

    labels = list(G_to_plot.vs[LABEL_ID])
    N = len(labels)
    print("Number␣of␣authors:", N)

    E = [e.tuple for e in G_to_plot.es]
    g_layout = G_to_plot.layout("fr") # Fruchterman-Reingold force-directed algorithm
    n_papers = list(G_to_plot.vs["count"])

    # Prepare the vertices and edges to be plotted:
    Xn = [g_layout[k][0] for k in range(N)]
    Yn = [g_layout[k][1] for k in range(N)]
    Xe = []
    Ye = []
    for e in E:
        Xe += [g_layout[e[0]][0], g_layout[e[1]][0], None]
        Ye += [g_layout[e[0]][1], g_layout[e[1]][1], None]

    plot = plot_coauthorship_graph(Xn, Yn, Xe, Ye, labels, n_papers)

    plot.write_image("coauthorship_graph.png", scale=6)

    # Upload the graph and open the URL:
    py.plot(plot, filename="coauthorship-graph", auto_open=True)
```

Appendix T

Complementary example gallery for MevaL

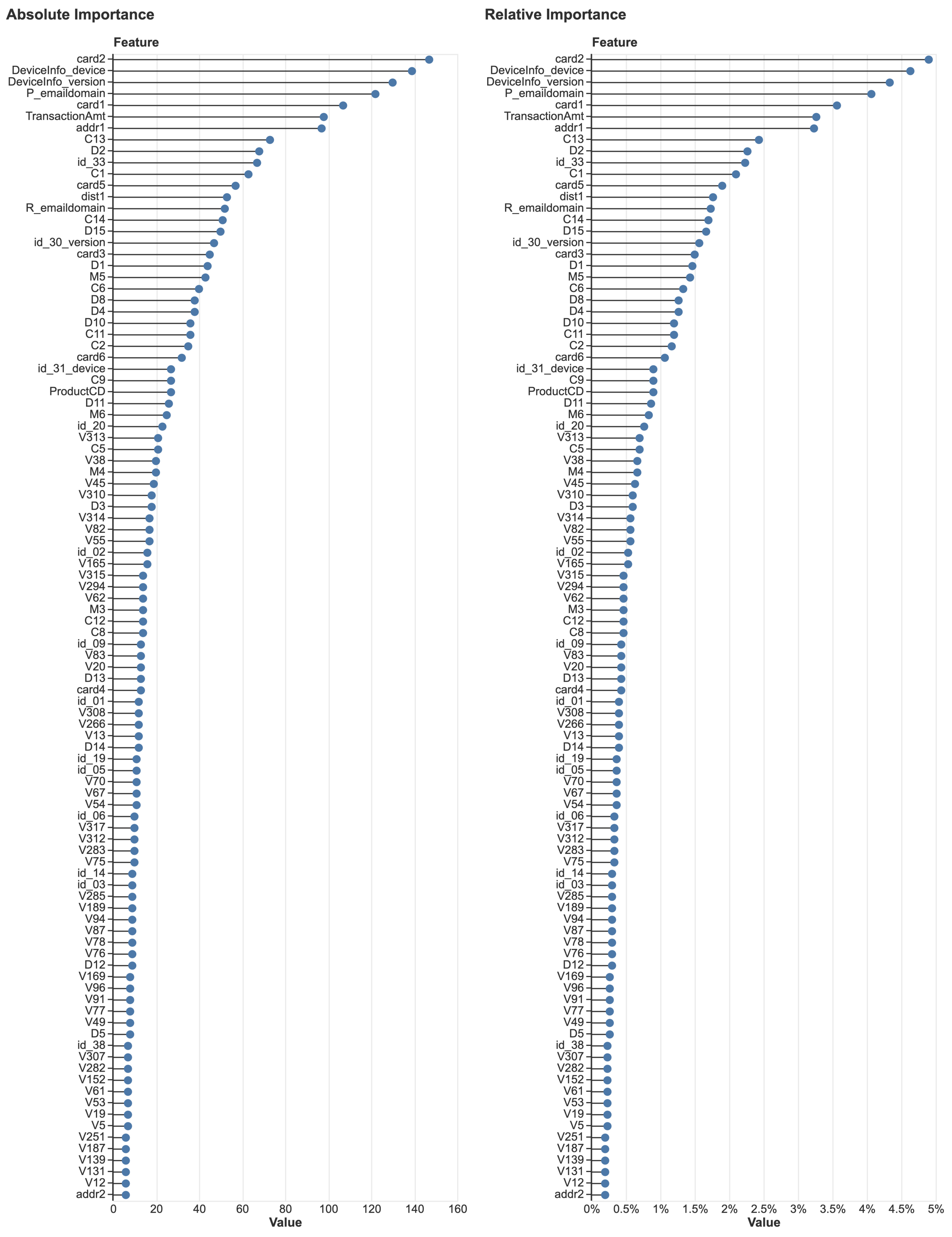


Figure T.1: Lollipop version for the FI ranking chart. This example shows the top 100 features.

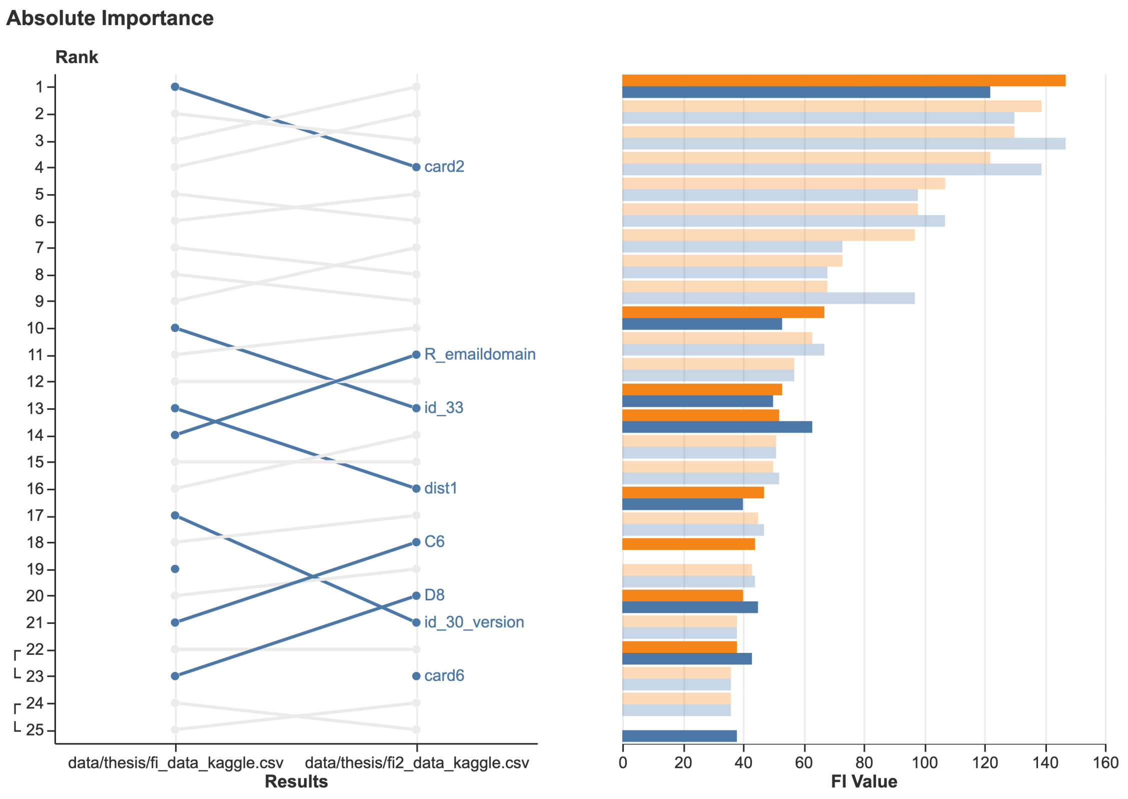


Figure T.2: A combination of a slopegraph and a horizontal grouped bar chart for comparing a reduced number of features in terms of FI values.

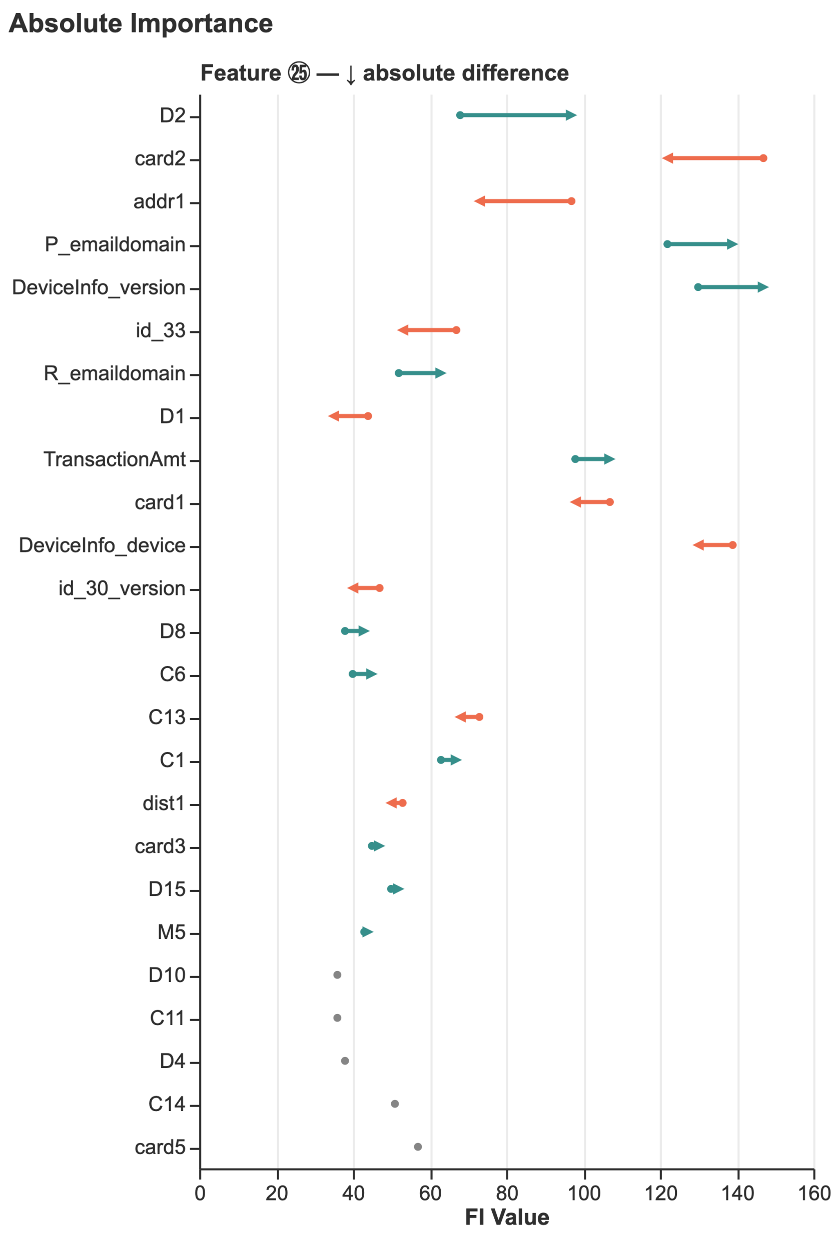
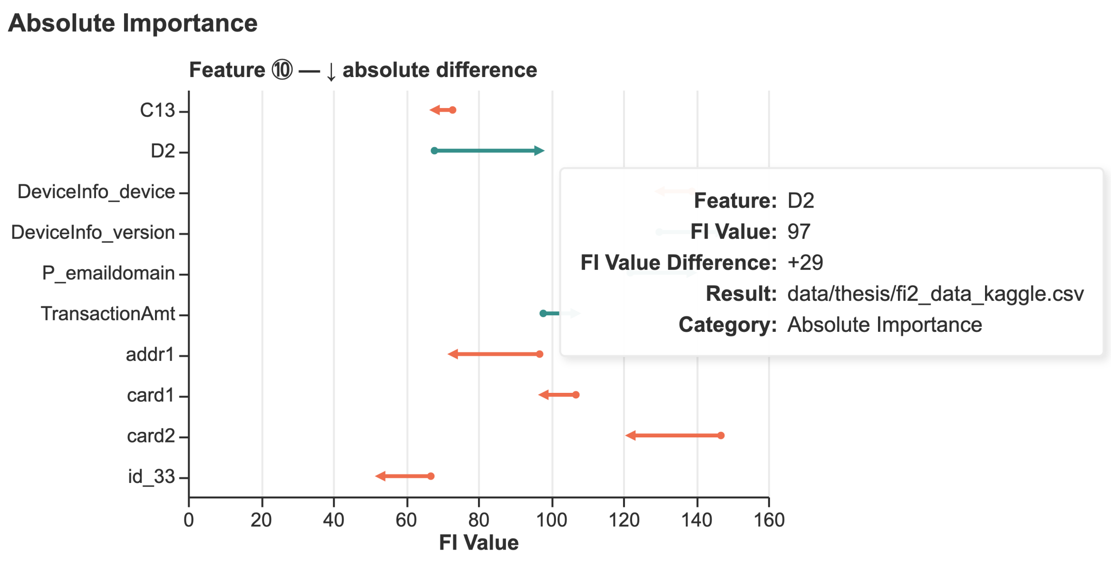
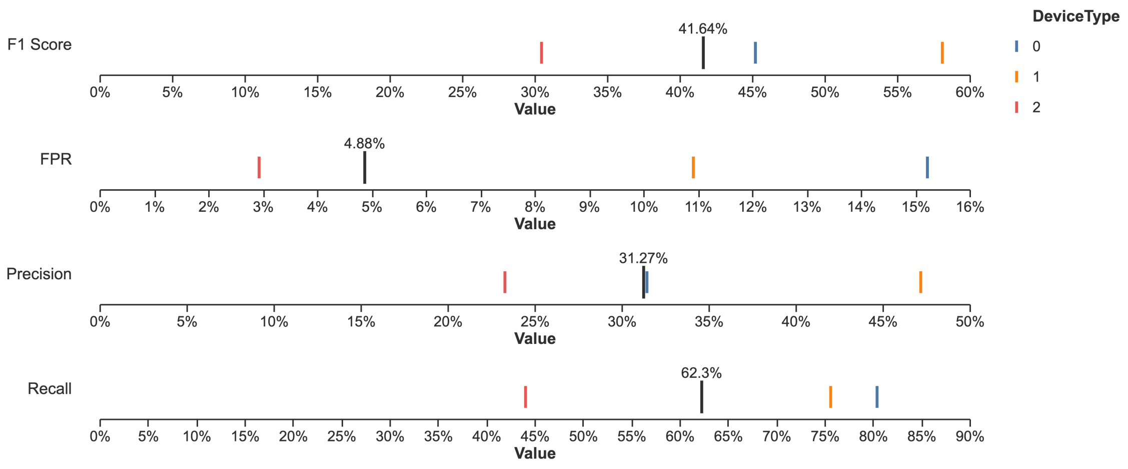


Figure T.3: Cleveland dot/arrow version for the FI comparison chart. In this example, the features are sorted in descending order of absolute difference in FI value.

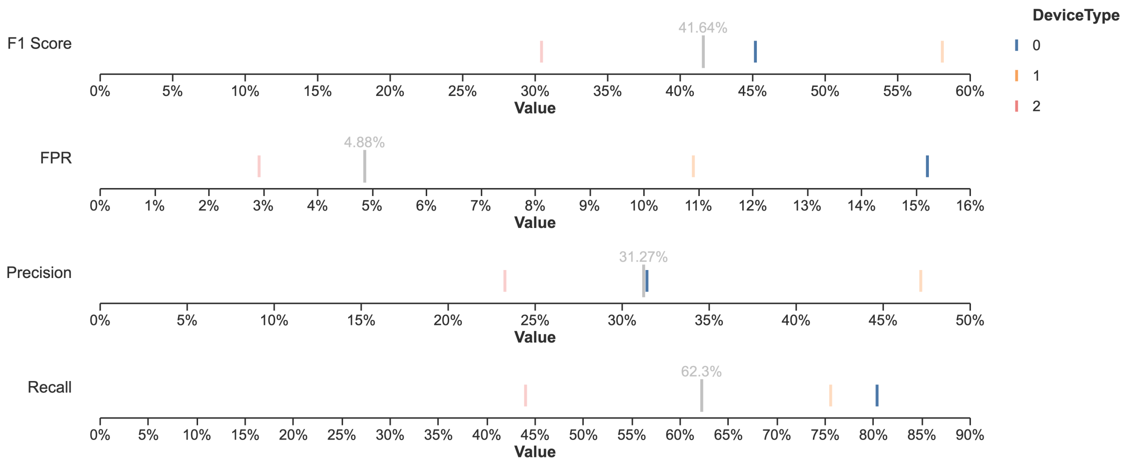


Feature: D2
FI Value: 97
FI Value Difference: +29
Result: data/thesis/fi2_data_kaggle.csv
Category: Absolute Importance

Figure T.4: Cleveland dot/arrow version for the FI comparison chart with a visible tooltip.



(a) This version is designed mainly for static use of this chart.



(b) In this example, one of the breakdown values (the *0 DeviceType*) is highlighted (the opacity value of the other breakdown values has decreased) after clicking on the respective tick or the respective mark in the legend. This interactive filtering mechanism works the same way for the non-colored version.

Figure T.5: Colored version for the breakdown strip chart. The marks in the legend mimic the ticks on the chart itself.

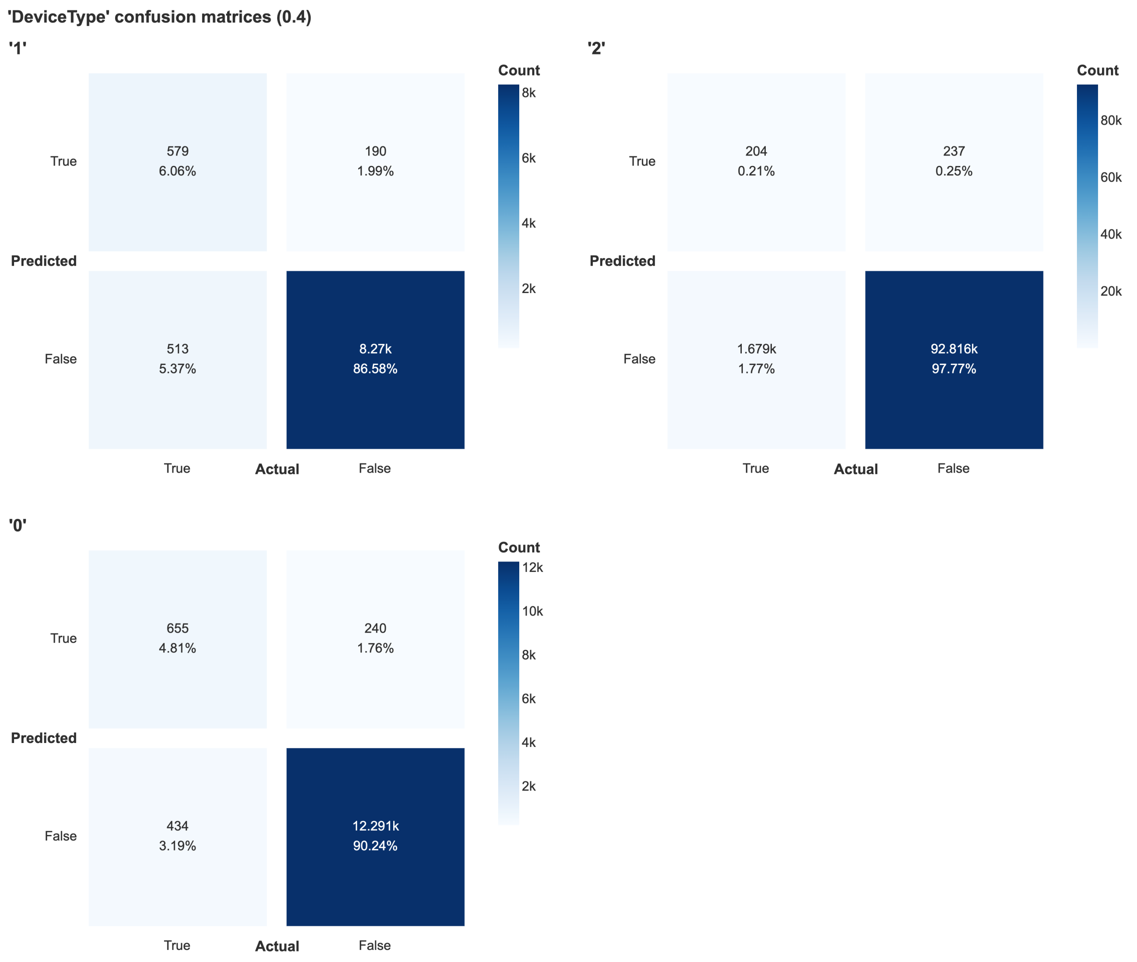


Figure T.6: Breakdown confusion matrix. In this example, there is an independent confusion matrix for each breakdown value in the *DeviceType* field.

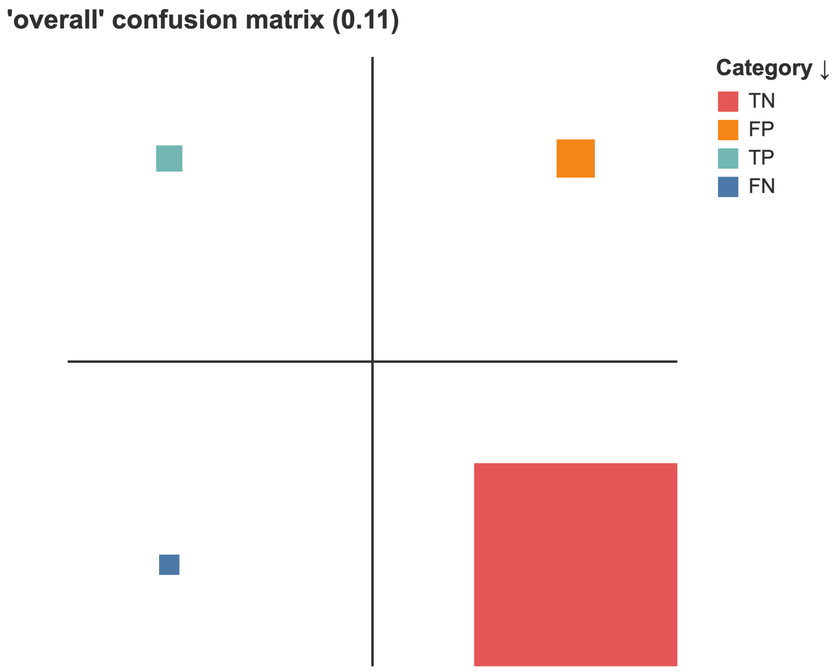


Figure T.7: Quadrant version [7] for the confusion matrix. In this example, the area encodes the number of instances associated with each confusion category.

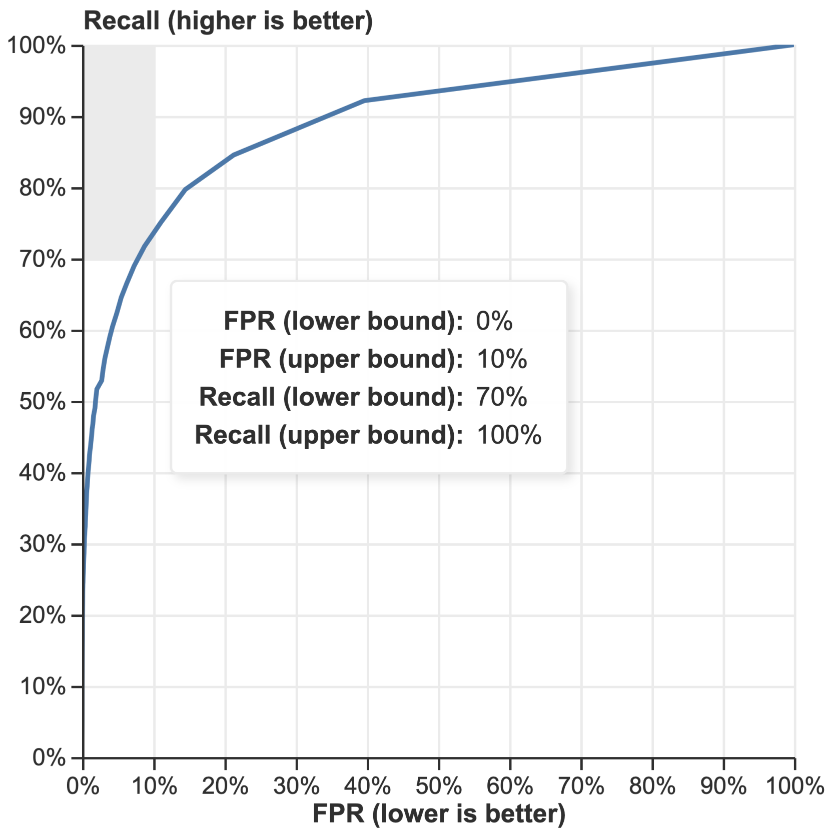
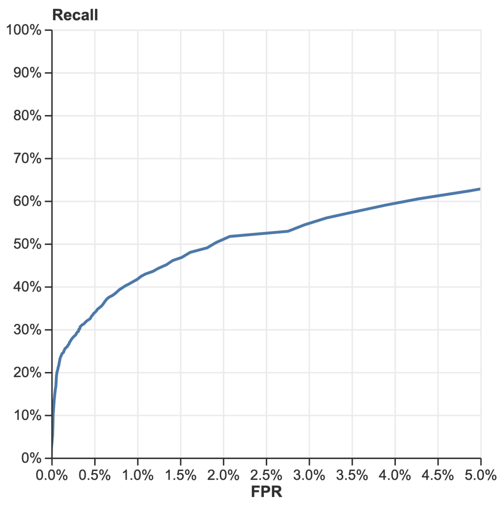
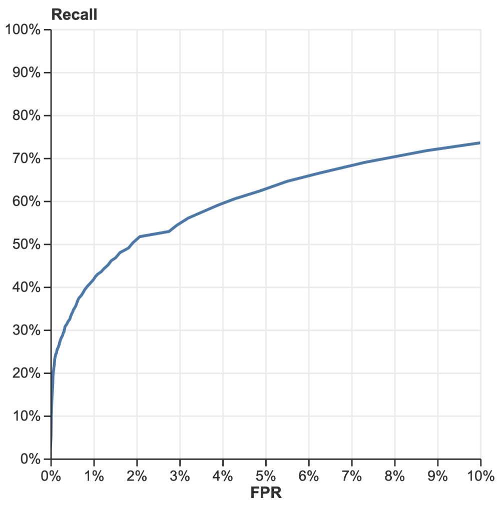


Figure T.8: ROC curve chart with a highlighted area. In this example, it is also possible to see the tooltip that appears when the data scientist hovers over this area. This area can be used to highlight the desired value ranges (the Recall values above a pre-established minimum up to a certain FPR value, such as 5%, for example) and contrast them with the obtained curve. This functionality works in a similar way for the PR and Gain curve charts.

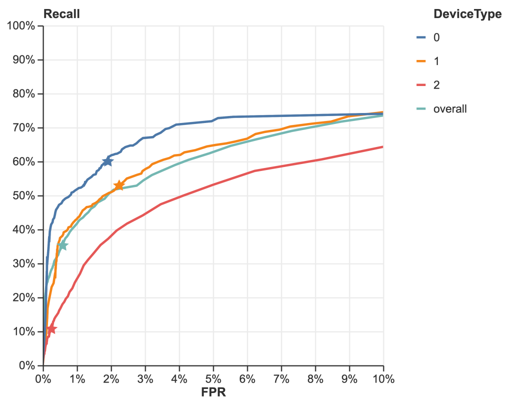


(a) In this example, FPR is limited to 5% (X-axis).

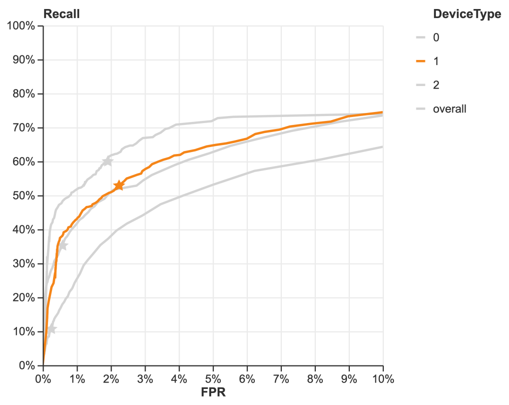


(b) In this example, FPR is limited to 10% (X-axis).

Figure T.9: Partial ROC curve chart. It is also possible to plot partial PR and Gain curves.



(a) In this example, it is possible to see that the choice of a certain classification threshold, when put in perspective in relation to a breakdown field, results in significantly different performance values (which may lead the data scientist to choose another threshold or to consider more than a threshold according to the breakdown value).



(b) By clicking on one of the legend entries, it is possible to highlight one line compared to the others (which remain in the chart to provide some context).

Figure T.10: ROC curve chart broken by *DeviceType* with the classification threshold 0.4 highlighted (on each line) using a star mark. This functionality works in a similar way for the PR and Gain curve charts.

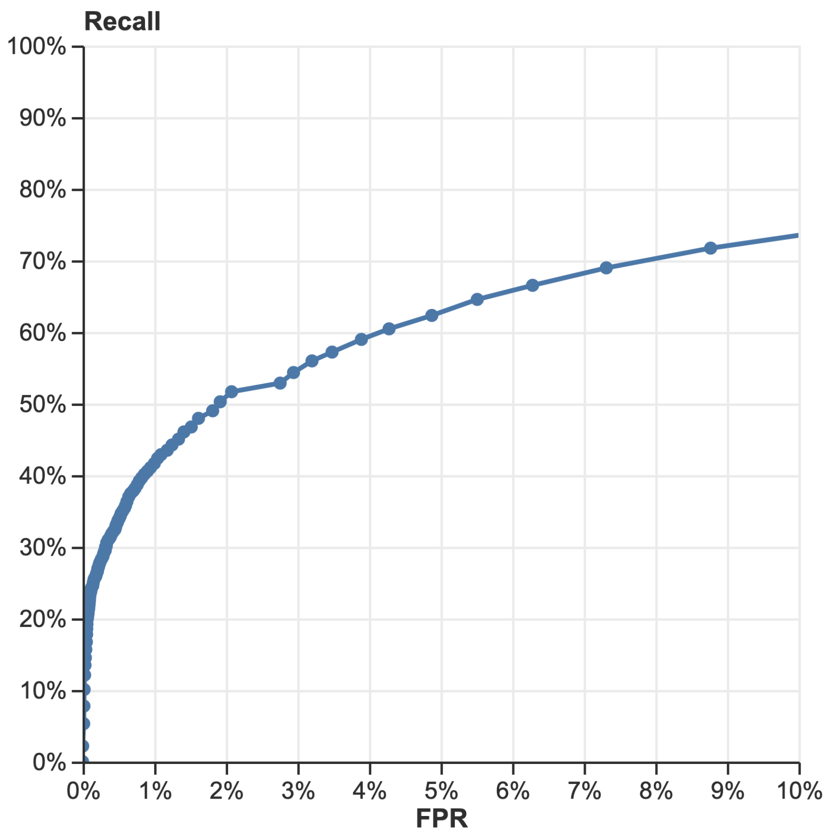


Figure T.11: Partial ROC curve chart with the concrete points that make up the line visible.

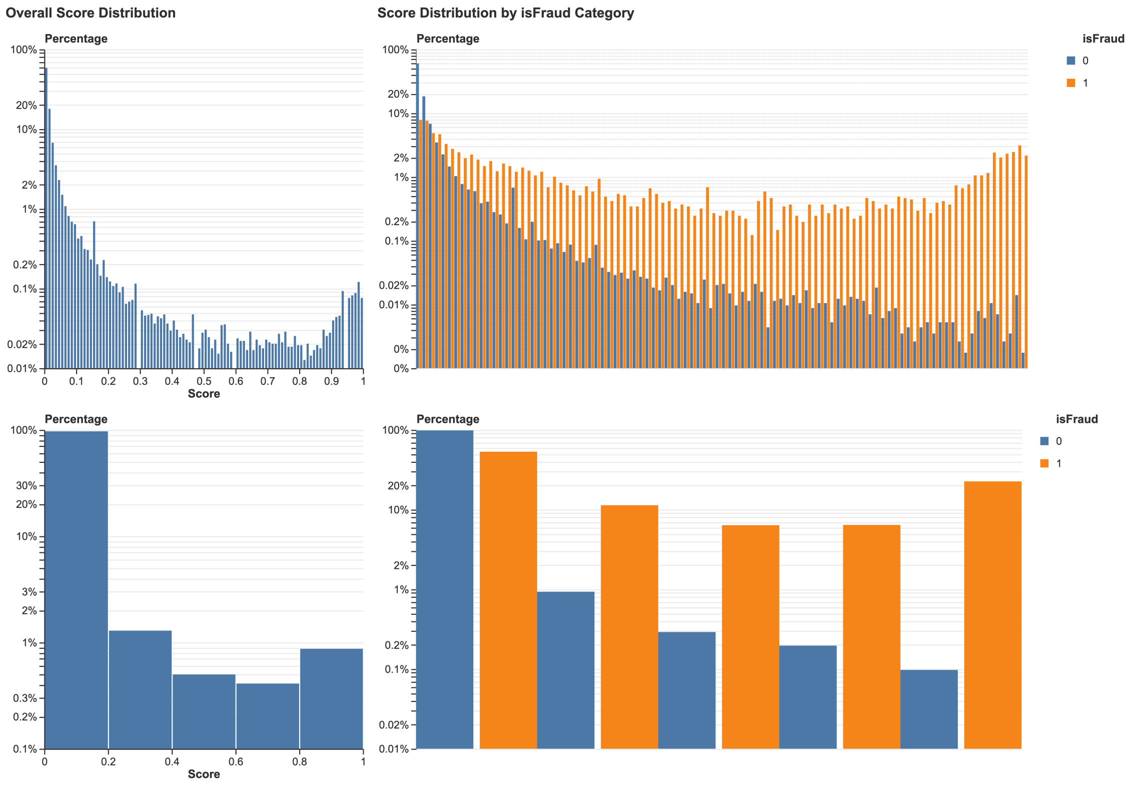


Figure T.12: Grouped score distribution chart. Although the color palette (Tableau 10) is not the most suitable for the semantic value associated with these breakdown values (the classes of the target field), this default choice was maintained due to its flexibility for any breakdown field that can be chosen for this chart (agnostic to the breakdown field, basically).

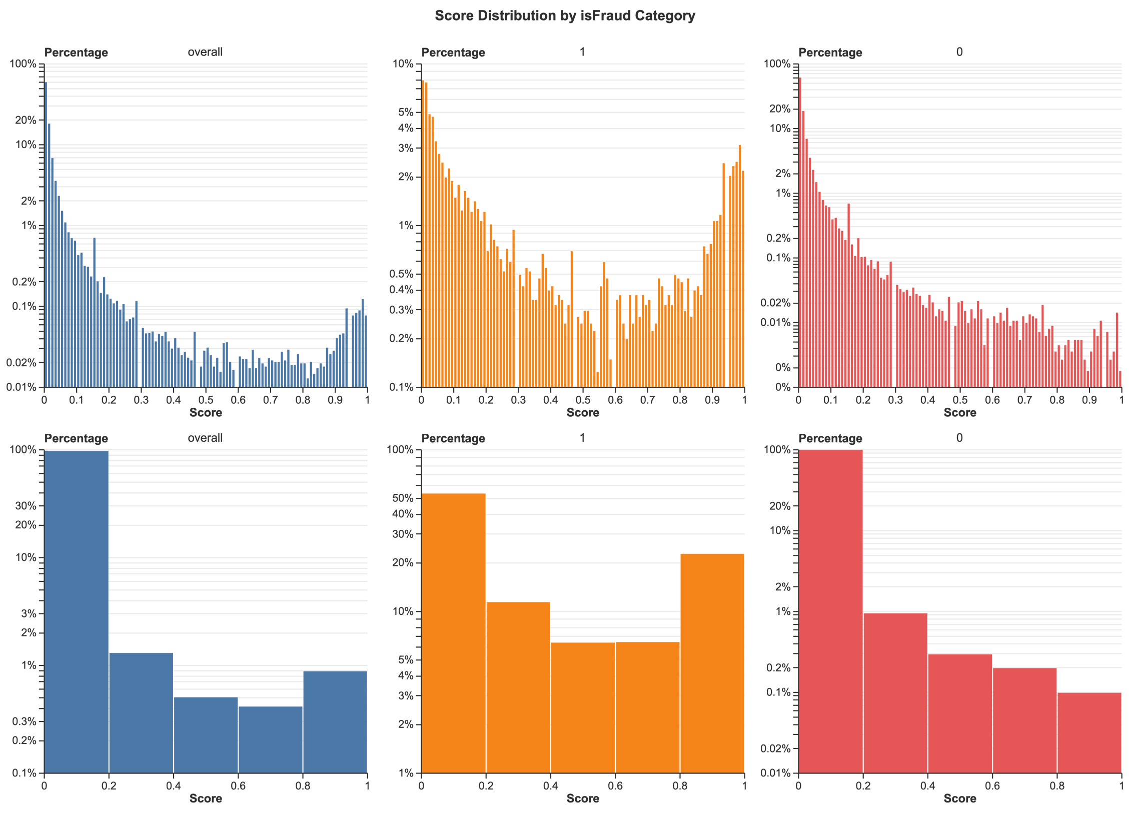


Figure T.13: Faceted score distribution chart. Although the color palette (Tableau 10) is not the most suitable for the semantic value associated with these breakdown values (the classes of the target field), this default choice was maintained due to its flexibility for any breakdown field that can be chosen for this chart (agnostic to the breakdown field, basically).

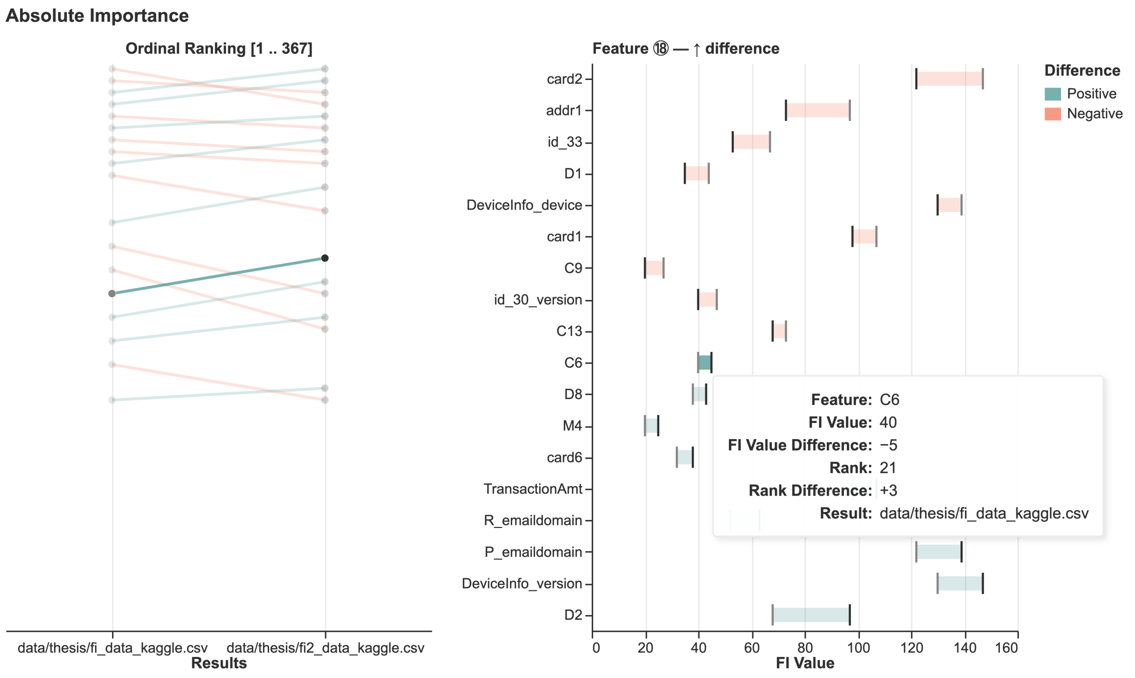


Figure T.14: FI comparison chart with an example of a tooltip and one of the features highlighted after clicking on one of the bands on the ranged strip (sub)chart. The two difference values in the tooltip must be read, in this case, from left to right, that is, they must be read taking into account the change compared to the other result. Therefore, for feature *C6*, the FI value in the first result is 40 instead of 45 ($40-45=-5$), while the place in the first ranking is three positions lower than in the second one.

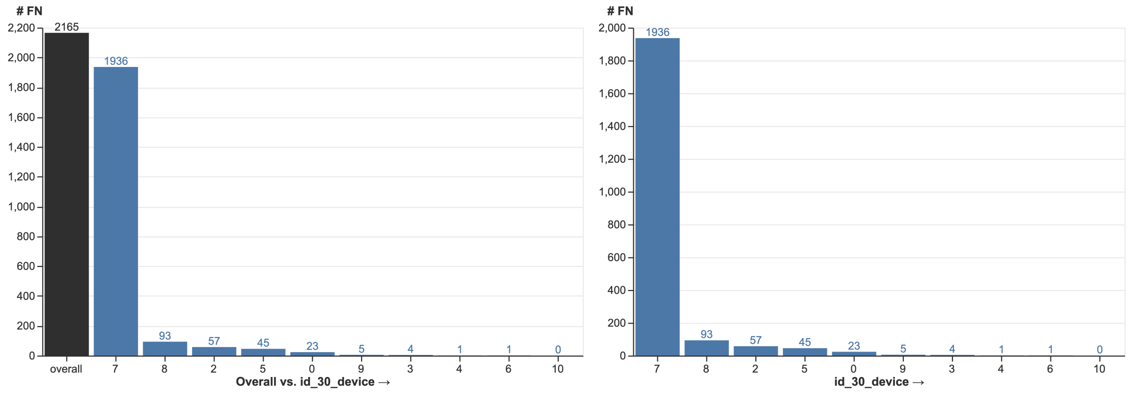
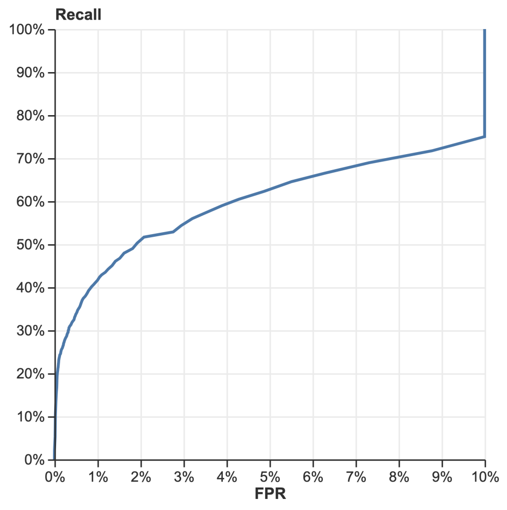
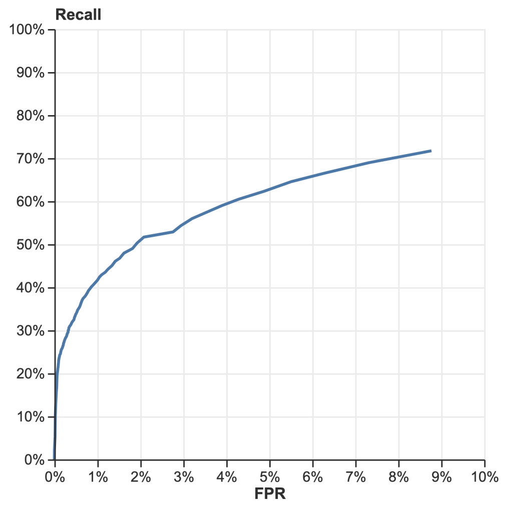


Figure T.15: FN-based confusion category bar chart. In this example, the *id_30_device* breakdown field is analyzed and the classification threshold is 0.2.



(a) Clamp-based partial ROC curve chart.



(b) Filter-based partial ROC curve chart.

Figure T.16: The other two partial modes (*partial_mode* parameter) for curve charts.

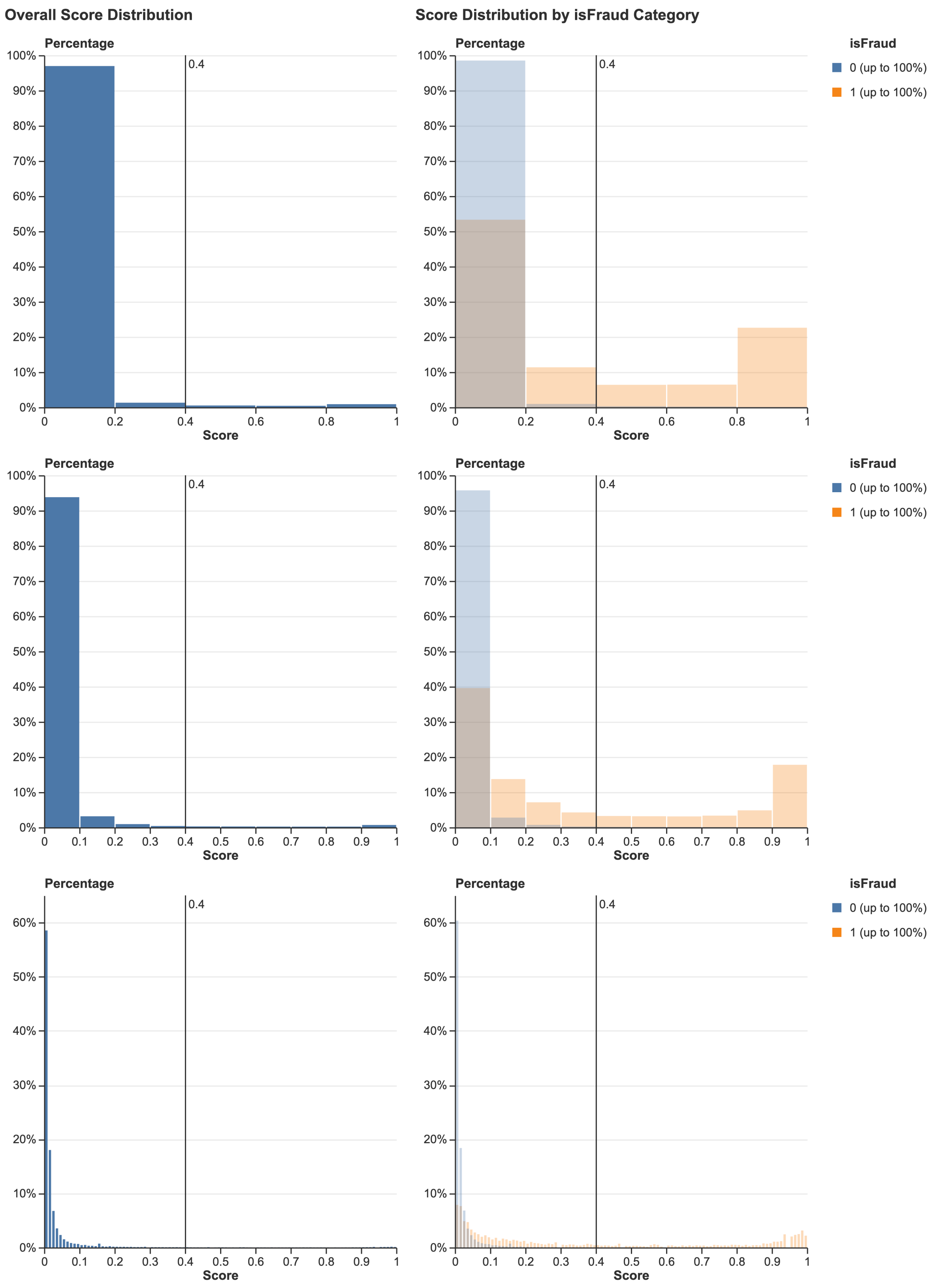
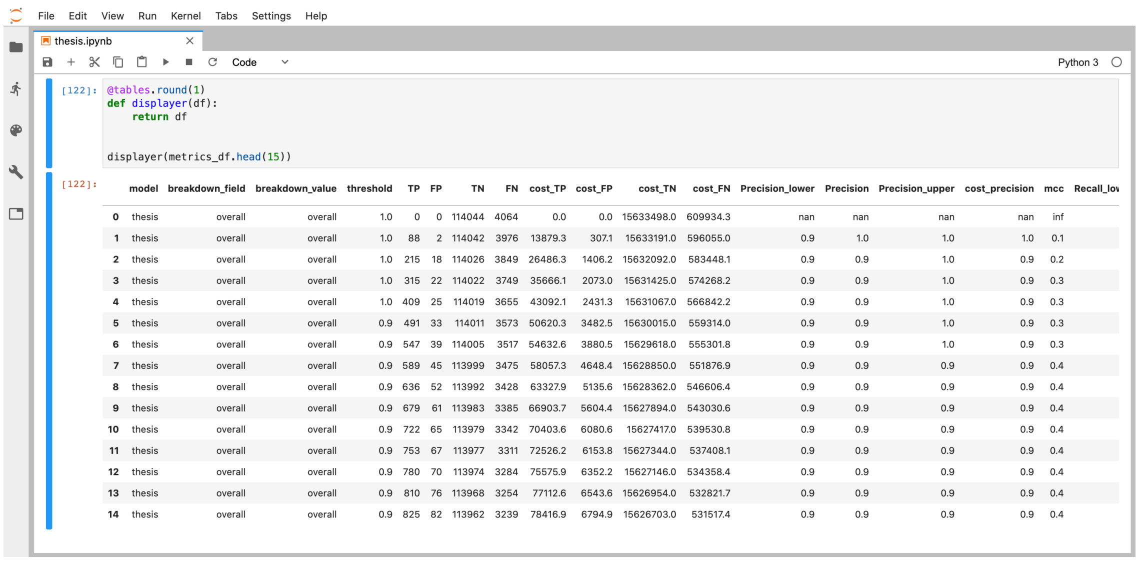


Figure T.17: An example of a threshold marker (with the respective numeric label) on a score distribution chart.



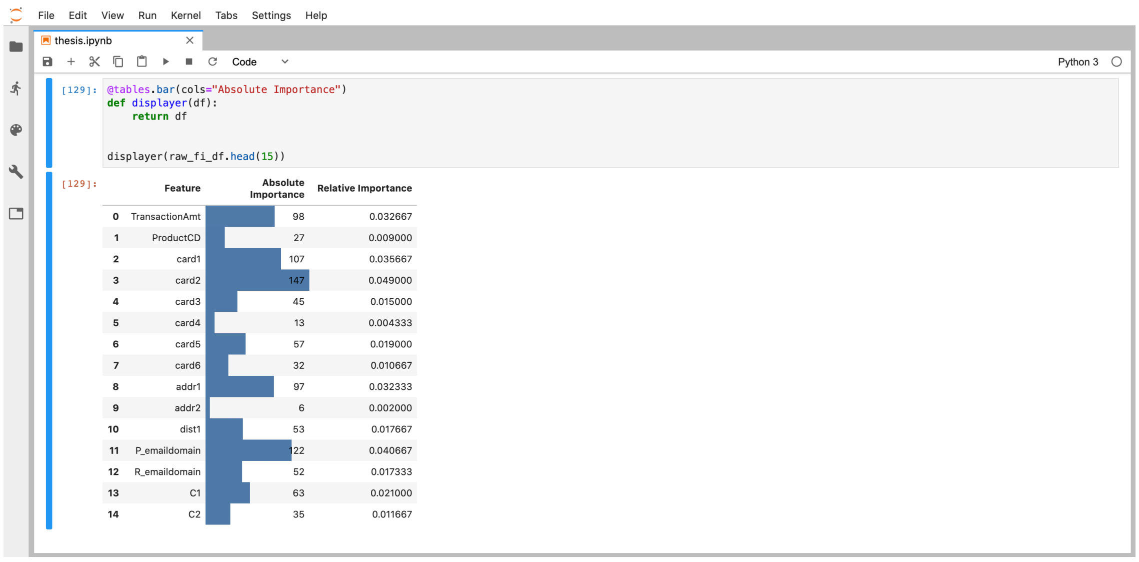
```
[122]: @tables.round(1)
       def displayer(df):
           return df


       displayer(metrics_df.head(15))
```

[122]:

	model	breakdown_field	breakdown_value	threshold	TP	FP	TN	FN	cost_TP	cost_FP	cost_TN	cost_FN	Precision_lower	Precision	Precision_upper	cost_precision	mcc	Recall_lov
0	thesis	overall	overall	1.0	0	0	114044	4064	0.0	0.0	15633498.0	609934.3	nan	nan	nan	nan	inf	
1	thesis	overall	overall	1.0	88	2	114042	3976	13879.3	307.1	15633191.0	596055.0	0.9	1.0	1.0	1.0	0.1	
2	thesis	overall	overall	1.0	215	18	114026	3849	26486.3	1406.2	15632092.0	583448.1	0.9	0.9	1.0	0.9	0.2	
3	thesis	overall	overall	1.0	315	22	114022	3749	35666.1	2073.0	15631425.0	574268.2	0.9	0.9	1.0	0.9	0.3	
4	thesis	overall	overall	1.0	409	25	114019	3655	43092.1	2431.3	15631067.0	566842.2	0.9	0.9	1.0	0.9	0.3	
5	thesis	overall	overall	0.9	491	33	114011	3573	50620.3	3482.5	15630015.0	559314.0	0.9	0.9	1.0	0.9	0.3	
6	thesis	overall	overall	0.9	547	39	114005	3517	54632.6	3880.5	15629618.0	555301.8	0.9	0.9	1.0	0.9	0.3	
7	thesis	overall	overall	0.9	589	45	113999	3475	58057.3	4648.4	15628850.0	551876.9	0.9	0.9	0.9	0.9	0.4	
8	thesis	overall	overall	0.9	636	52	113992	3428	63327.9	5135.6	15628362.0	546606.4	0.9	0.9	0.9	0.9	0.4	
9	thesis	overall	overall	0.9	679	61	113983	3385	66903.7	5604.4	15627894.0	543030.6	0.9	0.9	0.9	0.9	0.4	
10	thesis	overall	overall	0.9	722	65	113979	3342	70403.6	6080.6	15627417.0	539530.8	0.9	0.9	0.9	0.9	0.4	
11	thesis	overall	overall	0.9	753	67	113977	3311	72526.2	6153.8	15627344.0	537408.1	0.9	0.9	0.9	0.9	0.4	
12	thesis	overall	overall	0.9	780	70	113974	3284	75575.9	6352.2	15627146.0	534358.4	0.9	0.9	0.9	0.9	0.4	
13	thesis	overall	overall	0.9	810	76	113968	3254	77112.6	6543.6	15626954.0	532821.7	0.9	0.9	0.9	0.9	0.4	
14	thesis	overall	overall	0.9	825	82	113962	3239	78416.9	6794.9	15626703.0	531517.4	0.9	0.9	0.9	0.9	0.4	

Figure T.18: An example of the *round()* decorator to style pandas DataFrames.



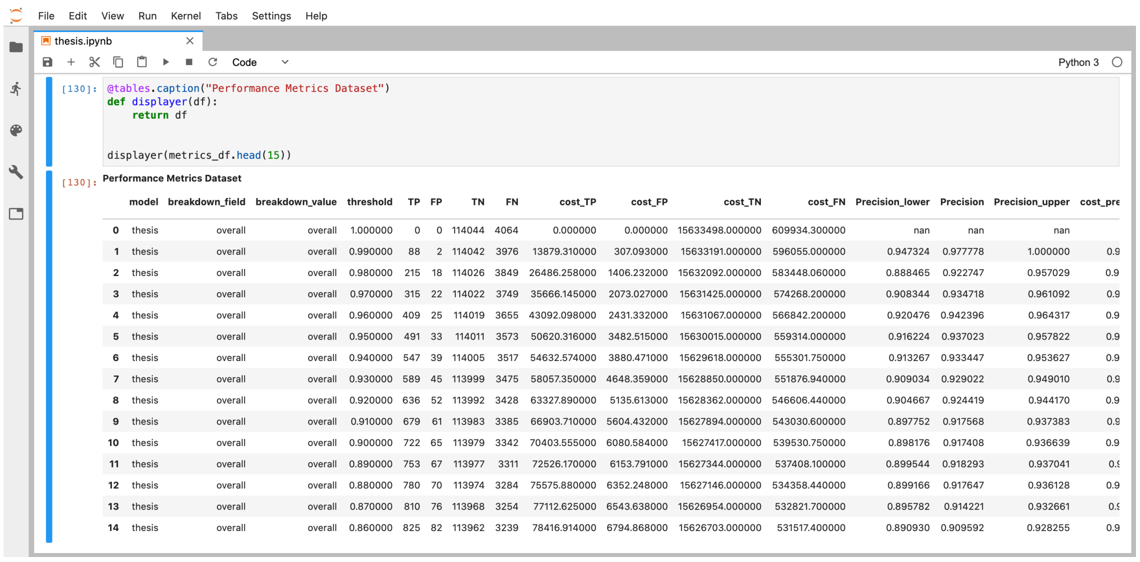
```
[129]: @tables.bar(cols="Absolute Importance")
       def displayer(df):
           return df


       displayer(raw_fi_df.head(15))
```

[129]:

	Feature	Absolute Importance	Relative Importance
0	TransactionAmt	98	0.032667
1	ProductCD	27	0.009000
2	card1	107	0.035667
3	card2	147	0.049000
4	card3	45	0.015000
5	card4	13	0.004333
6	card5	57	0.019000
7	card6	32	0.010667
8	addr1	97	0.032333
9	addr2	6	0.002000
10	dist1	53	0.017667
11	P_emaildomain	122	0.040667
12	R_emaildomain	52	0.017333
13	C1	63	0.021000
14	C2	35	0.011667

Figure T.19: An example of the *bar()* decorator to style pandas DataFrames.



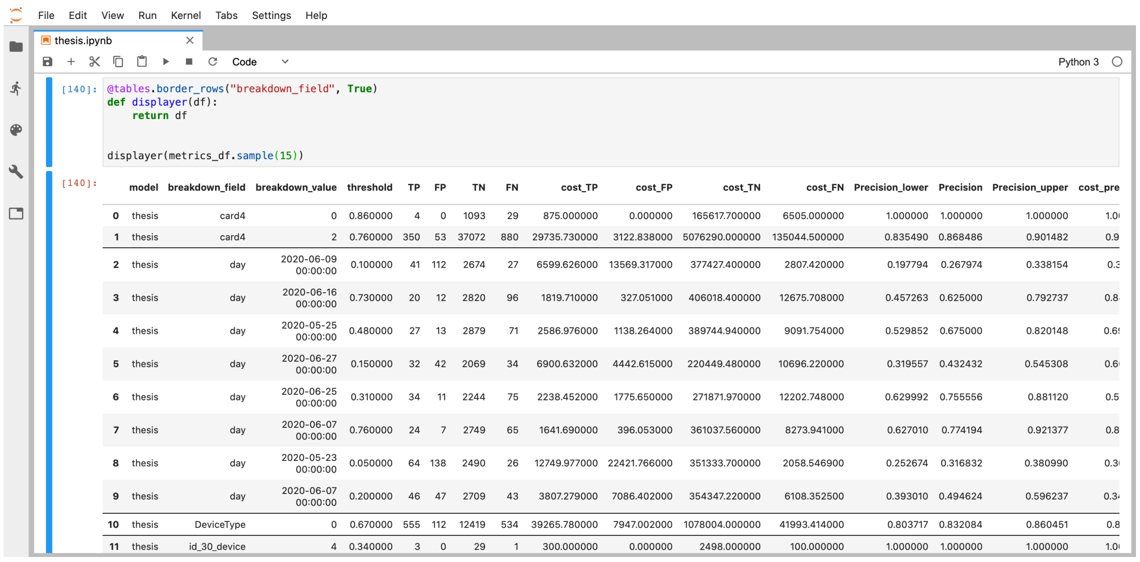
```
[130]: @tables.caption("Performance Metrics Dataset")
       def displayer(df):
           return df


       displayer(metrics_df.head(15))
```

[130]: **Performance Metrics Dataset**

	model	breakdown_field	breakdown_value	threshold	TP	FP	TN	FN	cost_TP	cost_FP	cost_TN	cost_FN	Precision_lower	Precision	Precision_upper	cost_pre
0	thesis	overall	overall	1.000000	0	0	114044	4064	0.000000	0.000000	15633498.000000	609934.300000	nan	nan	nan	
1	thesis	overall	overall	0.990000	88	2	114042	3976	13879.310000	307.093000	15633191.000000	596055.000000	0.947324	0.977778	1.000000	0.9
2	thesis	overall	overall	0.980000	215	18	114026	3849	26486.258000	1406.232000	15632092.000000	583448.060000	0.888465	0.922747	0.957029	0.9
3	thesis	overall	overall	0.970000	315	22	114022	3749	35666.145000	2073.027000	15631425.000000	574268.200000	0.908344	0.934718	0.961092	0.9
4	thesis	overall	overall	0.960000	409	25	114019	3655	43092.098000	2431.332000	15631067.000000	566842.200000	0.920476	0.942396	0.964317	0.9
5	thesis	overall	overall	0.950000	491	33	114011	3573	50620.316000	3482.515000	15630015.000000	559314.000000	0.916224	0.937023	0.957822	0.9
6	thesis	overall	overall	0.940000	547	39	114005	3517	54632.574000	3880.471000	15629618.000000	555301.750000	0.913267	0.933447	0.953627	0.9
7	thesis	overall	overall	0.930000	589	45	113999	3475	58057.350000	4648.359000	15628850.000000	551876.940000	0.909034	0.929022	0.949010	0.9
8	thesis	overall	overall	0.920000	636	52	113992	3428	63327.890000	5135.613000	15628362.000000	546606.440000	0.904667	0.924419	0.944170	0.9
9	thesis	overall	overall	0.910000	679	61	113983	3385	66903.710000	5604.432000	15627894.000000	543030.600000	0.897752	0.917568	0.937383	0.9
10	thesis	overall	overall	0.900000	722	65	113979	3342	70403.555000	6080.584000	15627417.000000	539530.750000	0.898176	0.917408	0.936639	0.9
11	thesis	overall	overall	0.890000	753	67	113977	3311	72526.170000	6153.791000	15627344.000000	537408.100000	0.899544	0.918293	0.937041	0.9
12	thesis	overall	overall	0.880000	780	70	113974	3284	75575.880000	6352.248000	15627146.000000	534358.440000	0.899166	0.917647	0.936128	0.9
13	thesis	overall	overall	0.870000	810	76	113968	3254	77112.625000	6543.638000	15626954.000000	532821.700000	0.895782	0.914221	0.932661	0.9
14	thesis	overall	overall	0.860000	825	82	113962	3239	78416.914000	6794.868000	15626703.000000	531517.400000	0.890930	0.909592	0.928255	0.9

Figure T.20: An example of the *caption()* decorator to style pandas DataFrames.



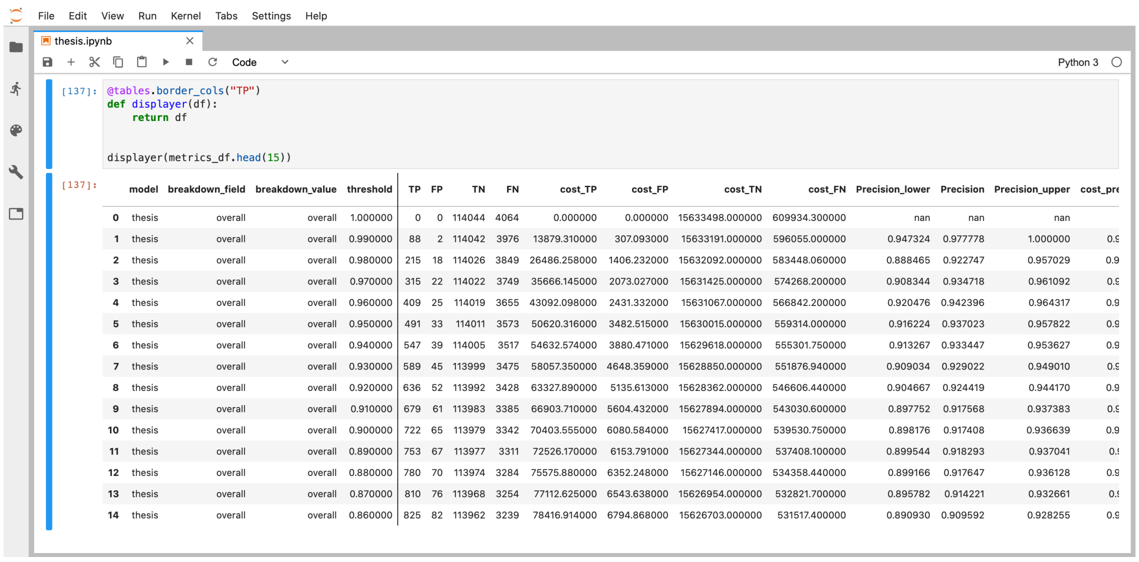
```
[140]: @tables.border_rows("breakdown_field", True)
       def displayer(df):
           return df


       displayer(metrics_df.sample(15))
```

[140]:

	model	breakdown_field	breakdown_value	threshold	TP	FP	TN	FN	cost_TP	cost_FP	cost_TN	cost_FN	Precision_lower	Precision	Precision_upper	cost_pre
0	thesis	card4	0	0.860000	4	0	1093	29	875.000000	0.000000	165617.700000	6505.000000	1.000000	1.000000	1.000000	1.0
1	thesis	card4	2	0.760000	350	53	37072	880	29735.730000	3122.838000	5076290.000000	135044.500000	0.835490	0.868486	0.901482	0.9
2	thesis	day	2020-06-09 00:00:00	0.100000	41	112	2674	27	6599.626000	13569.317000	377427.400000	2807.420000	0.197794	0.267974	0.338154	0.3
3	thesis	day	2020-06-16 00:00:00	0.730000	20	12	2820	96	1819.710000	327.051000	406018.400000	12675.708000	0.457263	0.625000	0.792737	0.8
4	thesis	day	2020-05-25 00:00:00	0.480000	27	13	2879	71	2586.976000	1138.264000	389744.940000	9091.754000	0.529852	0.675000	0.820148	0.6
5	thesis	day	2020-06-27 00:00:00	0.150000	32	42	2069	34	6900.632000	4442.615000	220449.480000	10696.220000	0.319557	0.432432	0.545308	0.6
6	thesis	day	2020-06-25 00:00:00	0.310000	34	11	2244	75	2238.452000	1775.650000	271871.970000	12202.748000	0.629992	0.755556	0.881120	0.5
7	thesis	day	2020-06-07 00:00:00	0.760000	24	7	2749	65	1641.690000	396.053000	361037.560000	8273.941000	0.627010	0.774194	0.921377	0.8
8	thesis	day	2020-05-23 00:00:00	0.050000	64	138	2490	26	12749.977000	22421.766000	351333.700000	2058.546900	0.252674	0.316832	0.380990	0.3
9	thesis	day	2020-06-07 00:00:00	0.200000	46	47	2709	43	3807.279000	7086.402000	354347.220000	6108.352500	0.393010	0.494624	0.596237	0.3
10	thesis	DeviceType	0	0.670000	555	112	12419	534	39265.780000	7947.002000	1078004.000000	41993.414000	0.803717	0.832084	0.860451	0.8
11	thesis	id_30_device	4	0.340000	3	0	29	1	300.000000	0.000000	2498.000000	100.000000	1.000000	1.000000	1.000000	1.0

Figure T.21: An example of the *border_rows()* decorator to style pandas DataFrames.



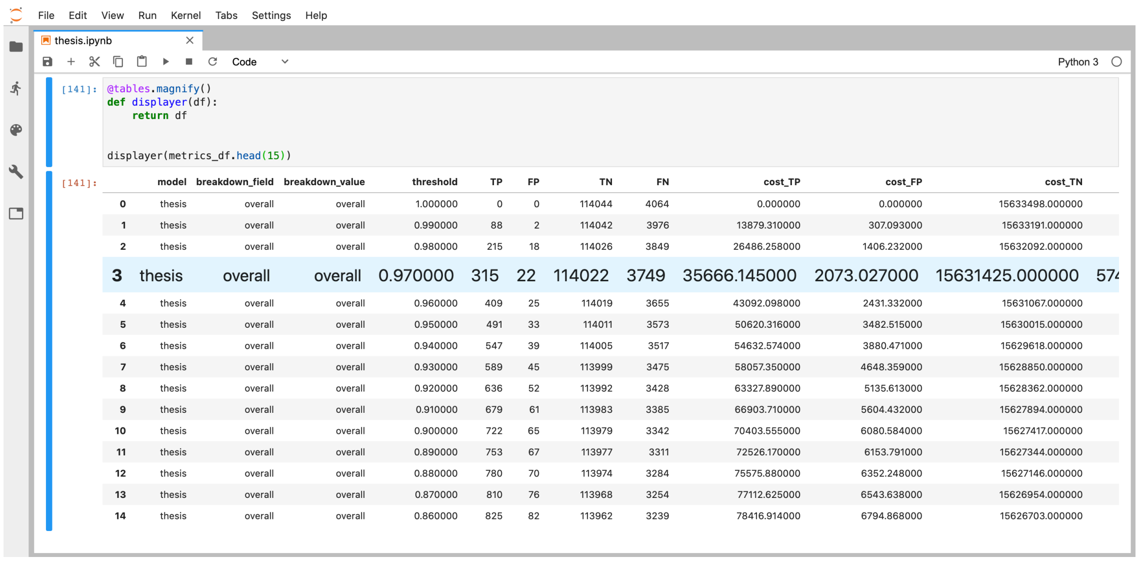
```
[137]: @tables.border_cols("TP")
       def displayer(df):
           return df


       displayer(metrics_df.head(15))
```

[137]:

	model	breakdown_field	breakdown_value	threshold	TP	FP	TN	FN	cost_TP	cost_FP	cost_TN	cost_FN	Precision_lower	Precision	Precision_upper	cost_pre
0	thesis	overall	overall	1.000000	0	0	114044	4064	0.000000	0.000000	15633498.000000	609934.300000	nan	nan	nan	
1	thesis	overall	overall	0.990000	88	2	114042	3976	13879.310000	307.093000	15633191.000000	596055.000000	0.947324	0.977778	1.000000	0.9
2	thesis	overall	overall	0.980000	215	18	114026	3849	26486.258000	1406.232000	15632092.000000	583448.060000	0.888465	0.922747	0.957029	0.9
3	thesis	overall	overall	0.970000	315	22	114022	3749	35666.145000	2073.027000	15631425.000000	574268.200000	0.908344	0.934718	0.961092	0.9
4	thesis	overall	overall	0.960000	409	25	114019	3655	43092.098000	2431.332000	15631067.000000	566842.200000	0.920476	0.942396	0.964317	0.9
5	thesis	overall	overall	0.950000	491	33	114011	3573	50620.316000	3482.515000	15630015.000000	559314.000000	0.916224	0.937023	0.957822	0.9
6	thesis	overall	overall	0.940000	547	39	114005	3517	54632.574000	3880.471000	15629618.000000	555301.750000	0.913267	0.933447	0.953627	0.9
7	thesis	overall	overall	0.930000	589	45	113999	3475	58057.350000	4648.359000	15628850.000000	551876.940000	0.909034	0.929022	0.949010	0.9
8	thesis	overall	overall	0.920000	636	52	113992	3428	63327.890000	5135.613000	15628362.000000	546606.440000	0.904667	0.924419	0.944170	0.9
9	thesis	overall	overall	0.910000	679	61	113983	3385	66903.710000	5604.432000	15627894.000000	543030.600000	0.897752	0.917568	0.937383	0.9
10	thesis	overall	overall	0.900000	722	65	113979	3342	70403.555000	6080.584000	15627417.000000	539530.750000	0.898176	0.917408	0.936639	0.9
11	thesis	overall	overall	0.890000	753	67	113977	3311	72526.170000	6153.791000	15627344.000000	537408.100000	0.899544	0.918293	0.937041	0.
12	thesis	overall	overall	0.880000	780	70	113974	3284	75575.880000	6352.248000	15627146.000000	534358.440000	0.899166	0.917647	0.936128	0.9
13	thesis	overall	overall	0.870000	810	76	113968	3254	77112.625000	6543.638000	15626954.000000	532821.700000	0.895782	0.914221	0.932661	0.
14	thesis	overall	overall	0.860000	825	82	113962	3239	78416.914000	6794.868000	15626703.000000	531517.400000	0.890930	0.909592	0.928255	0.9

Figure T.22: An example of the *border_cols()* decorator to style pandas DataFrames.



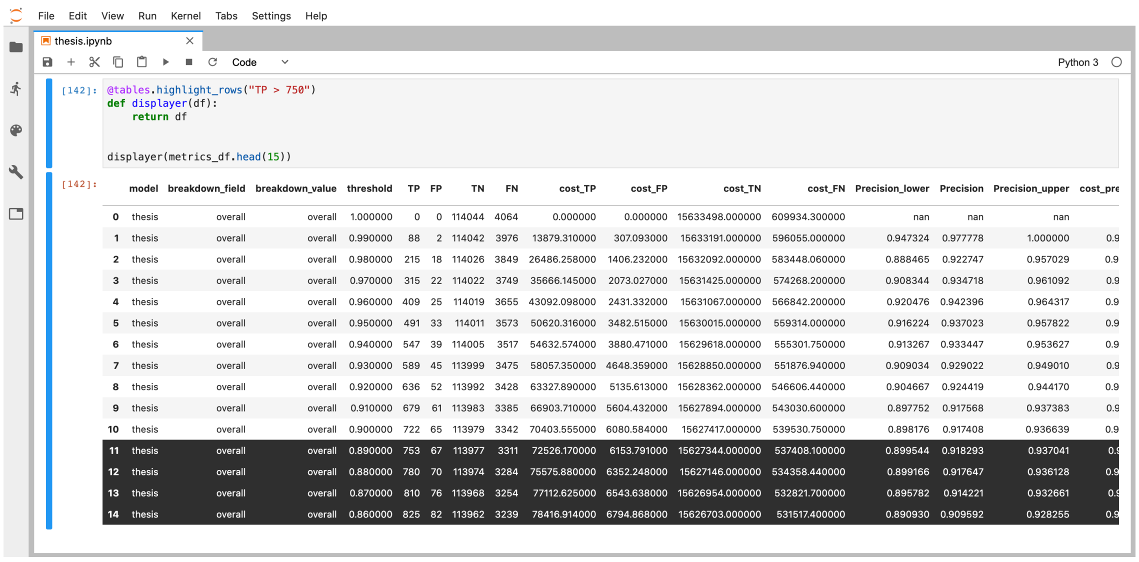
```
[141]: @tables.magnify()
       def displayer(df):
           return df


       displayer(metrics_df.head(15))
```

[141]:

	model	breakdown_field	breakdown_value	threshold	TP	FP	TN	FN	cost_TP	cost_FP	cost_TN	
0	thesis	overall	overall	1.000000	0	0	114044	4064	0.000000	0.000000	15633498.000000	
1	thesis	overall	overall	0.990000	88	2	114042	3976	13879.310000	307.093000	15633191.000000	
2	thesis	overall	overall	0.980000	215	18	114026	3849	26486.258000	1406.232000	15632092.000000	
3	thesis	overall	overall	0.970000	315	22	114022	3749	35666.145000	2073.027000	15631425.000000	574
4	thesis	overall	overall	0.960000	409	25	114019	3655	43092.098000	2431.332000	15631067.000000	
5	thesis	overall	overall	0.950000	491	33	114011	3573	50620.316000	3482.515000	15630015.000000	
6	thesis	overall	overall	0.940000	547	39	114005	3517	54632.574000	3880.471000	15629618.000000	
7	thesis	overall	overall	0.930000	589	45	113999	3475	58057.350000	4648.359000	15628850.000000	
8	thesis	overall	overall	0.920000	636	52	113992	3428	63327.890000	5135.613000	15628362.000000	
9	thesis	overall	overall	0.910000	679	61	113983	3385	66903.710000	5604.432000	15627894.000000	
10	thesis	overall	overall	0.900000	722	65	113979	3342	70403.555000	6080.584000	15627417.000000	
11	thesis	overall	overall	0.890000	753	67	113977	3311	72526.170000	6153.791000	15627344.000000	
12	thesis	overall	overall	0.880000	780	70	113974	3284	75575.880000	6352.248000	15627146.000000	
13	thesis	overall	overall	0.870000	810	76	113968	3254	77112.625000	6543.638000	15626954.000000	
14	thesis	overall	overall	0.860000	825	82	113962	3239	78416.914000	6794.868000	15626703.000000	

Figure T.23: An example of the *magnify()* decorator to style pandas DataFrames.



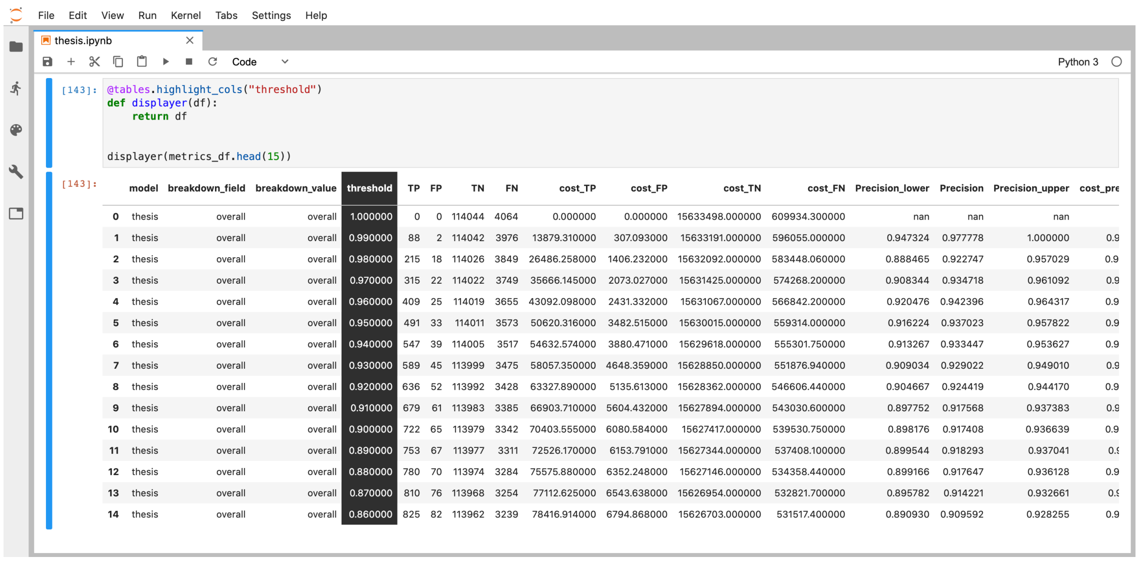
```
[142]: @tables.highlight_rows("TP > 750")
       def displayer(df):
           return df


       displayer(metrics_df.head(15))
```

[142]:

	model	breakdown_field	breakdown_value	threshold	TP	FP	TN	FN	cost_TP	cost_FP	cost_TN	cost_FN	Precision_lower	Precision	Precision_upper	cost_pre
0	thesis	overall	overall	1.000000	0	0	114044	4064	0.000000	0.000000	15633498.000000	609934.300000	nan	nan	nan	
1	thesis	overall	overall	0.990000	88	2	114042	3976	13879.310000	307.093000	15633191.000000	596055.000000	0.947324	0.977778	1.000000	0.9
2	thesis	overall	overall	0.980000	215	18	114026	3849	26486.258000	1406.232000	15632092.000000	583448.060000	0.888465	0.922747	0.957029	0.9
3	thesis	overall	overall	0.970000	315	22	114022	3749	35666.145000	2073.027000	15631425.000000	574268.200000	0.908344	0.934718	0.961092	0.9
4	thesis	overall	overall	0.960000	409	25	114019	3655	43092.098000	2431.332000	15631067.000000	566842.200000	0.920476	0.942396	0.964317	0.9
5	thesis	overall	overall	0.950000	491	33	114011	3573	50620.316000	3482.515000	15630015.000000	559314.000000	0.916224	0.937023	0.957822	0.9
6	thesis	overall	overall	0.940000	547	39	114005	3517	54632.574000	3880.471000	15629618.000000	555301.750000	0.913267	0.933447	0.953627	0.9
7	thesis	overall	overall	0.930000	589	45	113999	3475	58057.350000	4648.359000	15628850.000000	551876.940000	0.909034	0.929022	0.949010	0.9
8	thesis	overall	overall	0.920000	636	52	113992	3428	63327.890000	5135.613000	15628362.000000	546606.440000	0.904667	0.924419	0.944170	0.9
9	thesis	overall	overall	0.910000	679	61	113983	3385	66903.710000	5604.432000	15627894.000000	543030.600000	0.897752	0.917568	0.937383	0.9
10	thesis	overall	overall	0.900000	722	65	113979	3342	70403.555000	6080.584000	15627417.000000	539530.750000	0.898176	0.917408	0.936639	0.9
11	thesis	overall	overall	0.890000	753	67	113977	3311	72526.170000	6153.791000	15627344.000000	537408.100000	0.899544	0.918293	0.937041	0.9
12	thesis	overall	overall	0.880000	780	70	113974	3284	75575.880000	6352.248000	15627146.000000	534358.440000	0.899166	0.917647	0.936128	0.9
13	thesis	overall	overall	0.870000	810	76	113968	3254	77112.625000	6543.638000	15626954.000000	532821.700000	0.895782	0.914221	0.932661	0.9
14	thesis	overall	overall	0.860000	825	82	113962	3239	78416.914000	6794.868000	15626703.000000	531517.400000	0.890930	0.909592	0.928255	0.9

Figure T.24: An example of the *highlight_rows()* decorator to style pandas DataFrames.



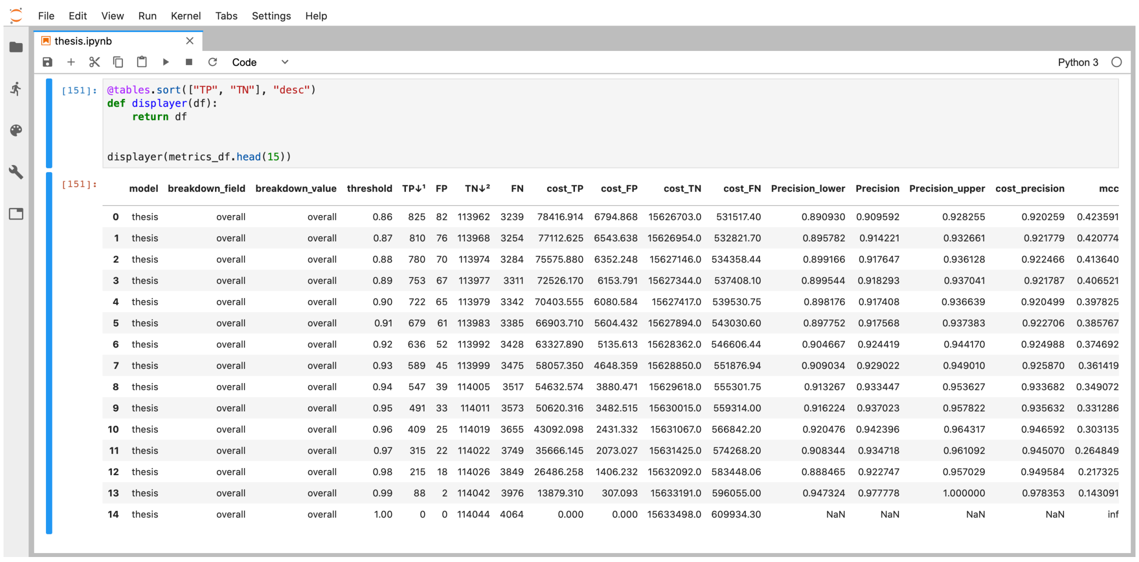
```
[143]: @tables.highlight_cols("threshold")
       def displayer(df):
           return df


       displayer(metrics_df.head(15))
```

[143]:

	model	breakdown_field	breakdown_value	threshold	TP	FP	TN	FN	cost_TP	cost_FP	cost_TN	cost_FN	Precision_lower	Precision	Precision_upper	cost_pre
0	thesis	overall	overall	1.000000	0	0	114044	4064	0.000000	0.000000	15633498.000000	609934.300000	nan	nan	nan	
1	thesis	overall	overall	0.990000	88	2	114042	3976	13879.310000	307.093000	15633191.000000	596055.000000	0.947324	0.977778	1.000000	0.9
2	thesis	overall	overall	0.980000	215	18	114026	3849	26486.258000	1406.232000	15632092.000000	583448.060000	0.888465	0.922747	0.957029	0.9
3	thesis	overall	overall	0.970000	315	22	114022	3749	35666.145000	2073.027000	15631425.000000	574268.200000	0.908344	0.934718	0.961092	0.9
4	thesis	overall	overall	0.960000	409	25	114019	3655	43092.098000	2431.332000	15631067.000000	566842.200000	0.920476	0.942396	0.964317	0.9
5	thesis	overall	overall	0.950000	491	33	114011	3573	50620.316000	3482.515000	15630015.000000	559314.000000	0.916224	0.937023	0.957822	0.9
6	thesis	overall	overall	0.940000	547	39	114005	3517	54632.574000	3880.471000	15629618.000000	555301.750000	0.913267	0.933447	0.953627	0.9
7	thesis	overall	overall	0.930000	589	45	113999	3475	58057.350000	4648.359000	15628850.000000	551876.940000	0.909034	0.929022	0.949010	0.9
8	thesis	overall	overall	0.920000	636	52	113992	3428	63327.890000	5135.613000	15628362.000000	546606.440000	0.904667	0.924419	0.944170	0.9
9	thesis	overall	overall	0.910000	679	61	113983	3385	66903.710000	5604.432000	15627894.000000	543030.600000	0.897752	0.917568	0.937383	0.9
10	thesis	overall	overall	0.900000	722	65	113979	3342	70403.555000	6080.584000	15627417.000000	539530.750000	0.898176	0.917408	0.936639	0.9
11	thesis	overall	overall	0.890000	753	67	113977	3311	72526.170000	6153.791000	15627344.000000	537408.100000	0.899544	0.918293	0.937041	0.9
12	thesis	overall	overall	0.880000	780	70	113974	3284	75575.880000	6352.248000	15627146.000000	534358.440000	0.899166	0.917647	0.936128	0.9
13	thesis	overall	overall	0.870000	810	76	113968	3254	77112.625000	6543.638000	15626954.000000	532821.700000	0.895782	0.914221	0.932661	0.9
14	thesis	overall	overall	0.860000	825	82	113962	3239	78416.914000	6794.868000	15626703.000000	531517.400000	0.890930	0.909592	0.928255	0.9

Figure T.25: An example of the *highlight_cols()* decorator to style pandas DataFrames.



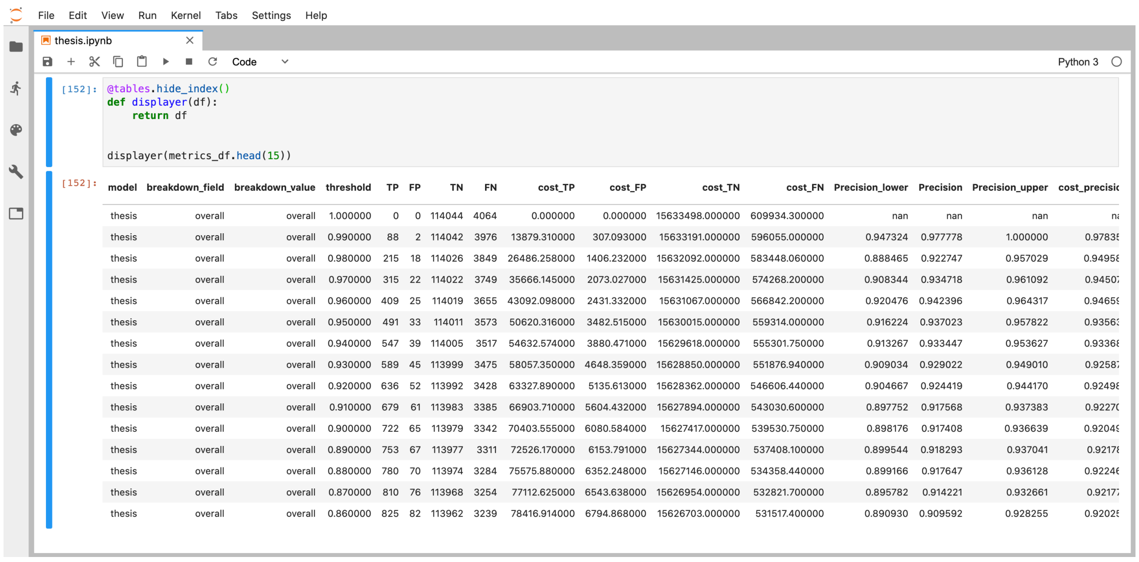
```
[151]: @tables.sort(["TP", "TN"], "desc")
       def displayer(df):
           return df


       displayer(metrics_df.head(15))
```

[151]:

	model	breakdown_field	breakdown_value	threshold	TP↓^{1}	FP	TN↓^{2}	FN	cost_TP	cost_FP	cost_TN	cost_FN	Precision_lower	Precision	Precision_upper	cost_precision	mcc
0	thesis	overall	overall	0.86	825	82	113962	3239	78416.914	6794.868	15626703.0	531517.40	0.890930	0.909592	0.928255	0.920259	0.423591
1	thesis	overall	overall	0.87	810	76	113968	3254	77112.625	6543.638	15626954.0	532821.70	0.895782	0.914221	0.932661	0.921779	0.420774
2	thesis	overall	overall	0.88	780	70	113974	3284	75575.880	6352.248	15627146.0	534358.44	0.899166	0.917647	0.936128	0.922466	0.413640
3	thesis	overall	overall	0.89	753	67	113977	3311	72526.170	6153.791	15627344.0	537408.10	0.899544	0.918293	0.937041	0.921787	0.406521
4	thesis	overall	overall	0.90	722	65	113979	3342	70403.555	6080.584	15627417.0	539530.75	0.898176	0.917408	0.936639	0.920499	0.397825
5	thesis	overall	overall	0.91	679	61	113983	3385	66903.710	5604.432	15627894.0	543030.60	0.897752	0.917568	0.937383	0.922706	0.385767
6	thesis	overall	overall	0.92	636	52	113992	3428	63327.890	5135.613	15628362.0	546606.44	0.904667	0.924419	0.944170	0.924988	0.374692
7	thesis	overall	overall	0.93	589	45	113999	3475	58057.350	4648.359	15628850.0	551876.94	0.909034	0.929022	0.949010	0.925870	0.361419
8	thesis	overall	overall	0.94	547	39	114005	3517	54632.574	3880.471	15629618.0	555301.75	0.913267	0.933447	0.953627	0.933682	0.349072
9	thesis	overall	overall	0.95	491	33	114011	3573	50620.316	3482.515	15630015.0	559314.00	0.916224	0.937023	0.957822	0.935632	0.331286
10	thesis	overall	overall	0.96	409	25	114019	3655	43092.098	2431.332	15631067.0	566842.20	0.920476	0.942396	0.964317	0.946592	0.303135
11	thesis	overall	overall	0.97	315	22	114022	3749	35666.145	2073.027	15631425.0	574268.20	0.908344	0.934718	0.961092	0.945070	0.264849
12	thesis	overall	overall	0.98	215	18	114026	3849	26486.258	1406.232	15632092.0	583448.06	0.888465	0.922747	0.957029	0.949584	0.217325
13	thesis	overall	overall	0.99	88	2	114042	3976	13879.310	307.093	15633191.0	596055.00	0.947324	0.977778	1.000000	0.978353	0.143091
14	thesis	overall	overall	1.00	0	0	114044	4064	0.000	0.000	15633498.0	609934.30	NaN	NaN	NaN	NaN	inf

Figure T.26: An example of the *sort()* decorator to style pandas DataFrames.



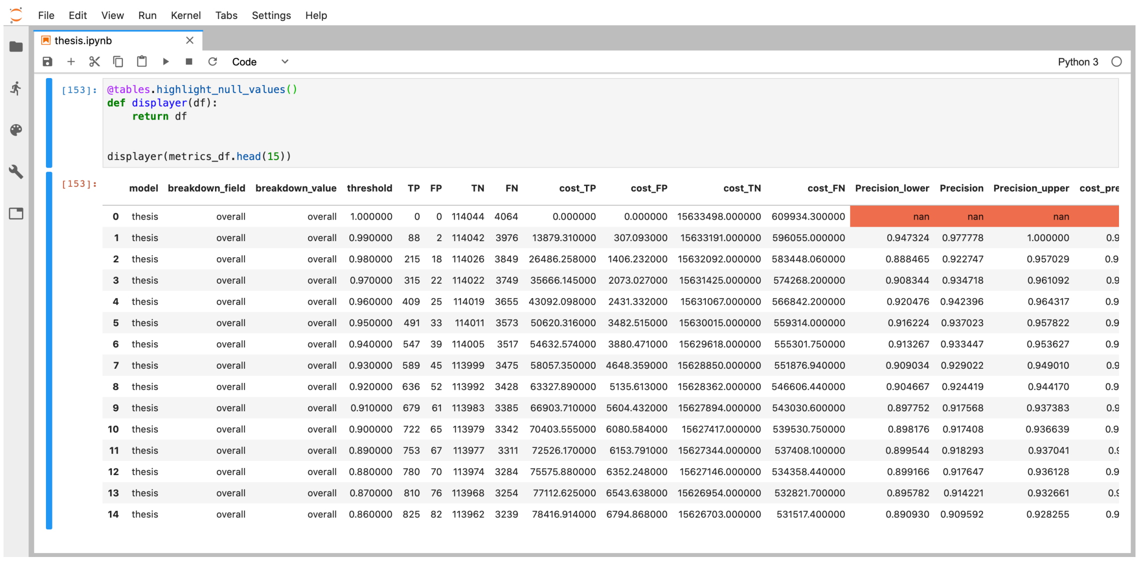
```
[152]: @tables.hide_index()
       def displayer(df):
           return df


       displayer(metrics_df.head(15))
```

[152]:

model	breakdown_field	breakdown_value	threshold	TP	FP	TN	FN	cost_TP	cost_FP	cost_TN	cost_FN	Precision_lower	Precision	Precision_upper	cost_precisi
thesis	overall	overall	1.000000	0	0	114044	4064	0.000000	0.000000	15633498.000000	609934.300000	nan	nan	nan	na
thesis	overall	overall	0.990000	88	2	114042	3976	13879.310000	307.093000	15633191.000000	596055.000000	0.947324	0.977778	1.000000	0.97835
thesis	overall	overall	0.980000	215	18	114026	3849	26486.258000	1406.232000	15632092.000000	583448.060000	0.888465	0.922747	0.957029	0.94958
thesis	overall	overall	0.970000	315	22	114022	3749	35666.145000	2073.027000	15631425.000000	574268.200000	0.908344	0.934718	0.961092	0.94507
thesis	overall	overall	0.960000	409	25	114019	3655	43092.098000	2431.332000	15631067.000000	566842.200000	0.920476	0.942396	0.964317	0.94659
thesis	overall	overall	0.950000	491	33	114011	3573	50620.316000	3482.515000	15630015.000000	559314.000000	0.916224	0.937023	0.957822	0.93563
thesis	overall	overall	0.940000	547	39	114005	3517	54632.574000	3880.471000	15629618.000000	555301.750000	0.913267	0.933447	0.953627	0.93368
thesis	overall	overall	0.930000	589	45	113999	3475	58057.350000	4648.359000	15628850.000000	551876.940000	0.909034	0.929022	0.949010	0.92587
thesis	overall	overall	0.920000	636	52	113992	3428	63327.890000	5135.613000	15628362.000000	546606.440000	0.904667	0.924419	0.944170	0.92498
thesis	overall	overall	0.910000	679	61	113983	3385	66903.710000	5604.432000	15627894.000000	543030.600000	0.897752	0.917568	0.937383	0.92270
thesis	overall	overall	0.900000	722	65	113979	3342	70403.555000	6080.584000	15627417.000000	539530.750000	0.898176	0.917408	0.936639	0.92049
thesis	overall	overall	0.890000	753	67	113977	3311	72526.170000	6153.791000	15627344.000000	537408.100000	0.899544	0.918293	0.937041	0.92178
thesis	overall	overall	0.880000	780	70	113974	3284	75575.880000	6352.248000	15627146.000000	534358.440000	0.899166	0.917647	0.936128	0.92246
thesis	overall	overall	0.870000	810	76	113968	3254	77112.625000	6543.638000	15626954.000000	532821.700000	0.895782	0.914221	0.932661	0.92177
thesis	overall	overall	0.860000	825	82	113962	3239	78416.914000	6794.868000	15626703.000000	531517.400000	0.890930	0.909592	0.928255	0.92025

Figure T.27: An example of the *hide_index()* decorator to style pandas DataFrames.



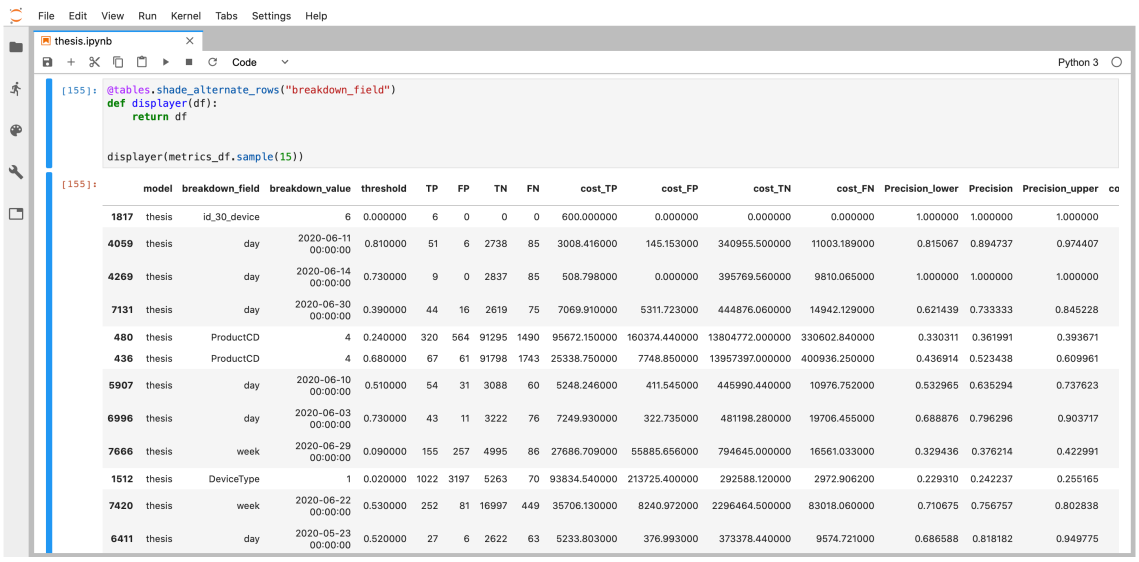
```
[153]: @tables.highlight_null_values()
       def displayer(df):
           return df


       displayer(metrics_df.head(15))
```

[153]:

	model	breakdown_field	breakdown_value	threshold	TP	FP	TN	FN	cost_TP	cost_FP	cost_TN	cost_FN	Precision_lower	Precision	Precision_upper	cost_pre
0	thesis	overall	overall	1.000000	0	0	114044	4064	0.000000	0.000000	15633498.000000	609934.300000	nan	nan	nan	
1	thesis	overall	overall	0.990000	88	2	114042	3976	13879.310000	307.093000	15633191.000000	596055.000000	0.947324	0.977778	1.000000	0.9
2	thesis	overall	overall	0.980000	215	18	114026	3849	26486.258000	1406.232000	15632092.000000	583448.060000	0.888465	0.922747	0.957029	0.9
3	thesis	overall	overall	0.970000	315	22	114022	3749	35666.145000	2073.027000	15631425.000000	574268.200000	0.908344	0.934718	0.961092	0.9
4	thesis	overall	overall	0.960000	409	25	114019	3655	43092.098000	2431.332000	15631067.000000	566842.200000	0.920476	0.942396	0.964317	0.9
5	thesis	overall	overall	0.950000	491	33	114011	3573	50620.316000	3482.515000	15630015.000000	559314.000000	0.916224	0.937023	0.957822	0.9
6	thesis	overall	overall	0.940000	547	39	114005	3517	54632.574000	3880.471000	15629618.000000	555301.750000	0.913267	0.933447	0.953627	0.9
7	thesis	overall	overall	0.930000	589	45	113999	3475	58057.350000	4648.359000	15628850.000000	551876.940000	0.909034	0.929022	0.949010	0.9
8	thesis	overall	overall	0.920000	636	52	113992	3428	63327.890000	5135.613000	15628362.000000	546606.440000	0.904667	0.924419	0.944170	0.9
9	thesis	overall	overall	0.910000	679	61	113983	3385	66903.710000	5604.432000	15627894.000000	543030.600000	0.897752	0.917568	0.937383	0.9
10	thesis	overall	overall	0.900000	722	65	113979	3342	70403.555000	6080.584000	15627417.000000	539530.750000	0.898176	0.917408	0.936639	0.9
11	thesis	overall	overall	0.890000	753	67	113977	3311	72526.170000	6153.791000	15627344.000000	537408.100000	0.899544	0.918293	0.937041	0.
12	thesis	overall	overall	0.880000	780	70	113974	3284	75575.880000	6352.248000	15627146.000000	534358.440000	0.899166	0.917647	0.936128	0.9
13	thesis	overall	overall	0.870000	810	76	113968	3254	77112.625000	6543.638000	15626954.000000	532821.700000	0.895782	0.914221	0.932661	0.
14	thesis	overall	overall	0.860000	825	82	113962	3239	78416.914000	6794.868000	15626703.000000	531517.400000	0.890930	0.909592	0.928255	0.9

Figure T.28: An example of the *highlight_null_values()* decorator to style pandas DataFrames.



```
[155]: @tables.shade_alternate_rows("breakdown_field")
       def displayer(df):
           return df


       displayer(metrics_df.sample(15))
```

[155]:

	model	breakdown_field	breakdown_value	threshold	TP	FP	TN	FN	cost_TP	cost_FP	cost_TN	cost_FN	Precision_lower	Precision	Precision_upper	co
1817	thesis	id_30_device	6	0.000000	6	0	0	0	600.000000	0.000000	0.000000	0.000000	1.000000	1.000000	1.000000	
4059	thesis	day	2020-06-11 00:00:00	0.810000	51	6	2738	85	3008.416000	145.153000	340955.500000	11003.189000	0.815067	0.894737	0.974407	
4269	thesis	day	2020-06-14 00:00:00	0.730000	9	0	2837	85	508.798000	0.000000	395769.560000	9810.065000	1.000000	1.000000	1.000000	
7131	thesis	day	2020-06-30 00:00:00	0.390000	44	16	2619	75	7069.910000	5311.723000	444876.060000	14942.129000	0.621439	0.733333	0.845228	
480	thesis	ProductCD	4	0.240000	320	564	91295	1490	95672.150000	160374.440000	13804772.000000	330602.840000	0.330311	0.361991	0.393671	
436	thesis	ProductCD	4	0.680000	67	61	91798	1743	25338.750000	7748.850000	13957397.000000	400936.250000	0.436914	0.523438	0.609961	
5907	thesis	day	2020-06-10 00:00:00	0.510000	54	31	3088	60	5248.246000	411.545000	445990.440000	10976.752000	0.532965	0.635294	0.737623	
6996	thesis	day	2020-06-03 00:00:00	0.730000	43	11	3222	76	7249.930000	322.735000	481198.280000	19706.455000	0.688876	0.796296	0.903717	
7666	thesis	week	2020-06-29 00:00:00	0.090000	155	257	4995	86	27686.709000	55885.656000	794645.000000	16561.033000	0.329436	0.376214	0.422991	
1512	thesis	DeviceType	1	0.020000	1022	3197	5263	70	93834.540000	213725.400000	292588.120000	2972.906200	0.229310	0.242237	0.255165	
7420	thesis	week	2020-06-22 00:00:00	0.530000	252	81	16997	449	35706.130000	8240.972000	2296464.500000	83018.060000	0.710675	0.756757	0.802838	
6411	thesis	day	2020-05-23 00:00:00	0.520000	27	6	2622	63	5233.803000	376.993000	373378.440000	9574.721000	0.686588	0.818182	0.949775	

Figure T.29: An example of the *shade_alternate_rows()* decorator to style pandas DataFrames.

Appendix U

Gantt chart

Initially, when presenting the dissertation plan, this project was divided into seven milestones distributed as follows:

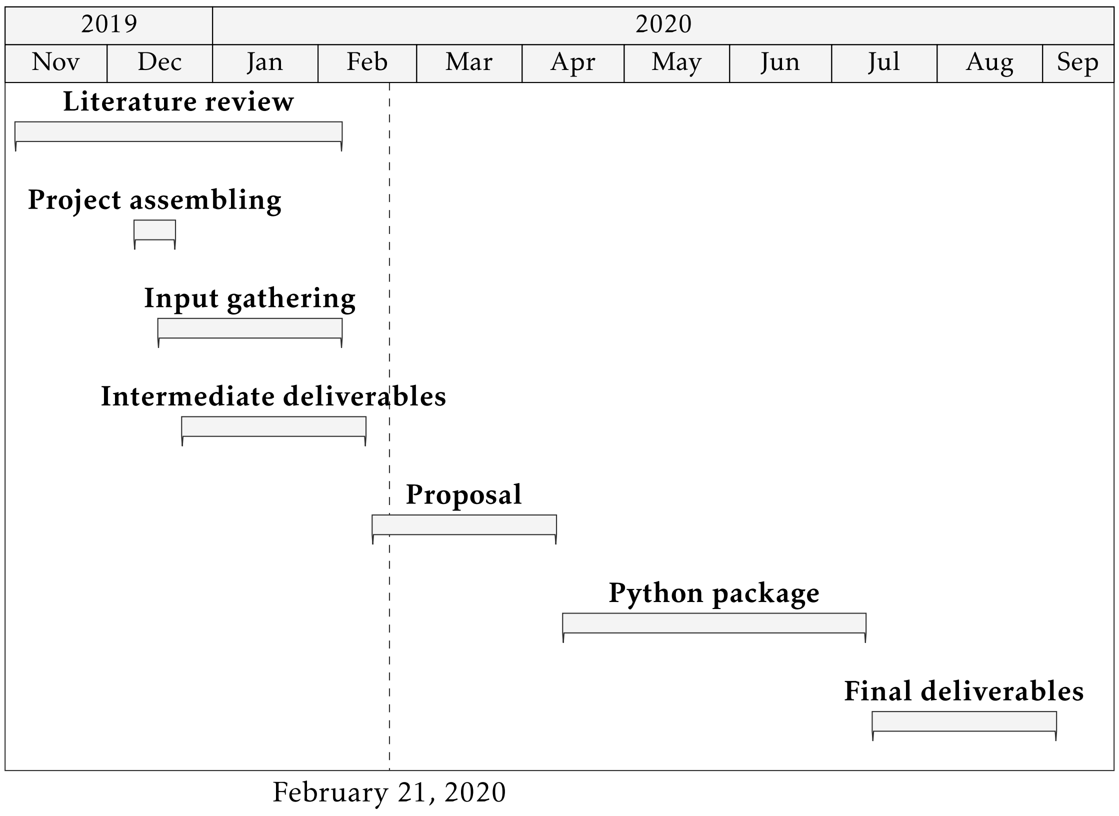


Figure U.1: Initial Gantt chart for the master's thesis.

In the end, the same milestones were approximately distributed as follows:

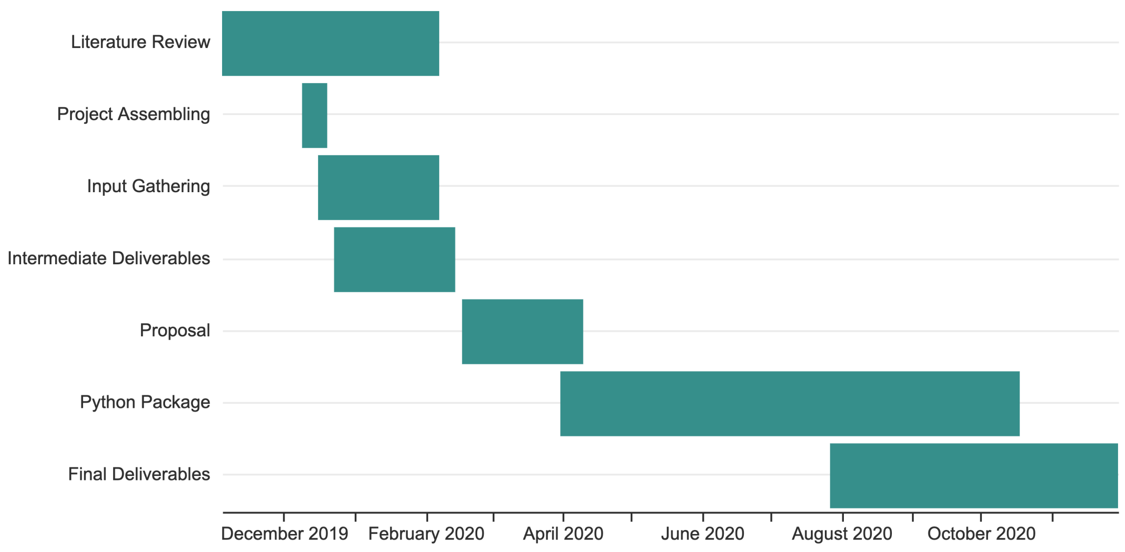


Figure U.2: Final timeline after project completion.

Appendix V

Mockup review conversation guidelines

Agenda:

1. Hello + Context (if necessary).
2. Present the considerations about the mockup with the help of the slide deck.

   Ask for questions.
3. Present the general structure of the mockup, with the help of the Table of
4. Contents, that the focus of this mockup is on the API, and mention that they can interrupt whenever they want.
5. Present each feature in the mockup (walkthrough).
6. Merge the presentation of each feature with questions about the possible utility, possible important customizations, and the current use.
7. Collect feedback on the following questions:

   Do you think the features presented cover your current workflow? Can you share an example, please?

   Do you think there should be some model-specific features?

   In terms of workflow, what do you think would be most important to you? An API or a standard notebook?

   In general, how many models/files do you analyze at one time? Is this analysis dependent on the most important previous conclusions? How is your procedure for loading files to a notebook?
8. Wrap-up summary + What haven't I asked you today that you think would be valuable for us to know? + Thanks.

Appendix W

Sticky header decorator for pandas DataFrames

Listing W.1: Snippet for the *sticky_header()* decorator to apply to a function to display a particular pandas DataFrame.

```python
from functools import wraps


def get_sticky_header_style(height):
    styles = [
        dict(
            selector="thead␣th",
            props=[("position", "sticky"), ("top", 0), ("background", "#FFFFFF")],
        ),
        dict(
            selector="", # `table` HTML element
            props=[
                ("display", "block"),
                ("overflow", "auto"),
                ("height", f"{height}px"),
            ],
        ),
    ]

    return styles


def sticky_header(height=300):
    def decorator_sticky_header(func):
        @wraps(func)
        def wrapper_sticky_header(*args, **kwargs):
            value = func(*args, **kwargs)
            styles = get_sticky_header_style(height)
            try:
                styler = value.style.set_table_styles(styles)
                return styler
            except AttributeError:
                previous_styles = value.table_styles
                return value.set_table_styles(
                    styles + previous_styles if previous_styles else styles
                )

        return wrapper_sticky_header

    return decorator_sticky_header
```