U.PORTO

FACULDADE DE ECONOMIA
UNIVERSIDADE DO PORTO

Building an automated MLOps pipeline and recommending an open-source stack to deploy a Machine Learning Application

William Inouye Almeida

Dissertation

Master in Modelling, Data Analysis and Decision Support Systems

Supervised by
Prof. Doutor João Gama
Prof. Doutor Bruno Veloso

2023

Acknowledgements

I would like to thank the institution FEP and all the MADSAD teachers that make this master's degree one of the best in Europe and reference in the field of Data Analytics. Especially to my advisor and co-advisor that accepted my thesis Prof. Doutor João Gama and Prof. Bruno Veloso and helped me when needed.

Resumo

O termo Machine Learning Operations (MLOps) tem proliferado desde o final de 2018 devido à dificuldade de implantar modelos de Machine Learning (ML) na produção e garantir o gerenciamento sustentável do ciclo de vida do ML. Esta dissertação aborda esses desafios a desenvolver um pipeline completo de CI/CD usando ferramentas de aprendizado de máquina de código aberto, guiadas pela metodologia CRISP-ML(Q) e pela estrutura do modelo de maturidade do Google. O pipeline implementado permite a implantação de um aplicativo de ML e o monitoramento contínuo do desempenho do modelo, acionando o retreinamento quando necessário. O estudo se concentra em uma tarefa de aprendizado supervisionado usando a API da Web do Spotify para classificar as faixas em listas de reprodução de humor, especificamente felizes e tristes. A seguir a metodologia CRISP-ML(Q), esta pesquisa atende aos padrões MLOps e garante a integração bem-sucedida dos sistemas ML na produção. Além disso, a pilha MLOps escolhida, abrangendo ferramentas como Git, DVC, GitHub Actions e MLflow, facilita o controle de versão de código e dados e a criação de modelos ML reproduzíveis. O requisito de escalabilidade é demonstrado usando Spotify Web API e Streamlit. Além disso, o aspecto de monitoramento de MLOps é abordado usando relatórios claros para avaliar a qualidade dos dados, o desvio de dados e o desempenho da classificação do modelo. Os resultados indicam que o pipeline desenvolvido atende aos requisitos do MLOps e mostra a explicabilidade e os aspectos responsáveis da IA ao empregar a biblioteca Shap para identificar as principais características nas previsões do modelo. No geral, esta dissertação atinge com sucesso seus objetivos pretendidos, entregando um projeto MLOps abrangente que combina as melhores práticas da indústria de ML com ferramentas de MLOps de código aberto.

Palavras-Chave: *MLOps, Machine Learning, Automação de pipeline, Ferramentas de código aberto.*

Abstract

The term Machine Learning Operations (MLOps) has been proliferating since late 2018 due to the difficulty of deploying Machine Learning (ML) models into production and ensuring sustainable management of the ML lifecycle. This dissertation addresses these challenges by developing a complete CI/CD pipeline using open-source machine learning tools, guided by the CRISP-ML(Q) methodology and Google's maturity model framework. The implemented pipeline allows deployment of an ML application and continuous monitoring of model performance, triggering retraining when needed. The study focuses on a supervised learning task using the Spotify Web API to sort tracks into mood playlists, specifically happy and sad. Following the CRISP-ML(Q) methodology, this research meets MLOps standards and ensures the successful integration of ML systems into production. Additionally, the chosen MLOps stack, spanning tools such as Git, DVC, GitHub Actions and MLflow, facilitates code and data versioning and building reproducible ML models. The scalability requirement is demonstrated by utilizing the Spotify Web API and Streamlit. In addition, the MLOps monitoring aspect is addressed using Evidently reports to assess the data quality, data drift and classification performance of the model. The results indicate that the developed pipeline meets the MLOps requirements and shows the explainability and responsible aspects of the AI when employing the Shap library to identify the main features in the model predictions. Overall, this dissertation successfully achieves its intended goals, delivering a comprehensive MLOps project that combines ML industry best practices with an open-source MLOps stack.

Keywords: *MLOps, Machine Learning, Pipeline Automation, Open-source tools.*

Glossary

AI	Artificial Intelligence
CI/CD	Continuous Integration/Continuous Delivery
CRISP-ML(Q)	Machine Learning applications with Quality assurance methodology
CT	Continuous Training
DAG	Directed Acyclic Graph
DEVOPS	Development and Operations
ML	Machine Learning
MLOPS	Machine Learning Operations

Table of Contents

List of Tables

List of Figures

1 Introduction

Machine Learning Operations (MLOps) adapt the Development and Operations (DevOps) methodology from software development, which consists of a rapid deployment cycle for Machine Learning (ML) systems (Mäkinen, 2021). The relatively new term MLOps started to gain relevance in late 2018 and is also understood as the standardization and automation of the complete ML lifecycle (Treveil et al., 2020).

One of the main issues emerging in the field of ML recently is the difficulty of deploying ML models in production. Only a tiny percentage of ML models can reach production (Symeonidis et al., 2022). Furthermore, even models currently in production are managed by data scientists as manual ML workflows (Kreuzberger et al., 2022).

Therefore, for ML professionals, the deployment phase to production, automating model retraining and monitoring are the biggest challenges because, unlike software development, ML models depend not only on the code but also on the extremely volatile data, and the maintenance is strenuous (Mäkinen, 2021; Ruf et al., 2021).

1.1 Problem Definition and Motivation

Since the ML Models in an enterprise can take anywhere from six to 18 months to reach production, and most models have a small lifetime, this delay between the starting an ML project and deploying the model it usually means that the production model is no longer conforming to real-world data (Liu et al., 2020; Salvucci, 2021).

For this reason, at the end of this master's in data analytics, it is essential to understand the end-to-end life cycle of the ML system, including what is needed to build a model, using all the steps of MLOps methodology from business and data understanding to ML model deployment, monitoring, and retraining.

Furthermore, many software solutions must be used at different stages of the ML lifecycle. These solutions have recently evolved into open-source and commercial choices. However, commercial platforms have major disadvantages, obviously the high cost; they are also not 100% transparent and lack the flexibility generated by modularity (Ruf et al., 2021).

Thus, in this dissertation only open-source tools are used to develop the ML model. Furthermore, the next subsection presents the complete solution to the defined problem.

1.2 Objectives

The four objectives of the dissertation are listed below. They summarize the research effort and results of the literature review and the development and operationalization of the ML model.

1. Recommend open-source MLOps tools based on ease of use and simplification of the MLOps stack.
2. Build an explainable ML classification model using the Spotify Web API and follow the CRISP-ML(Q) methodology.
3. Enhance the ML model with the MLOps CI/CD pipeline to enable ML application deployment.
4. Monitor ML model performance in production with trigger retraining when needed.

1.3 Dissertation Structure

The structure of this dissertation is organized as follows. This first section is an introduction that describes the problems, motivations, and objectives.

The second section presents a literature review with the main topics related to ML and MLOps relevant to the dissertation. The third section explains the data used and the methodology of the project. Moreover, this section also shows the stack of MLOps chosen to fulfil the first objective.

Section four will present all stages of development of the ML model following the MLOps methodology and show the results achieved, with objectives two, three and four fulfilled.

Finally, in the fifth and last section, conclusions are drawn, and suggestions for future work related to MLOps are presented.

2 Literature Review

This chapter presents the literature review on the main topics related to MLOps. It starts with the origin of MLOps methodology, explaining the relationships between DevOps and it. The following subsection is about the requirements for building an MLOps system and what is the maturity model that can be achieved. Next, the ML lifecycle based on the CRISP-ML(Q) framework is explained. In the fourth subsection, the leading open-source MLOps tools are presented with the functionalities they perform. The final subsection describes the main challenges that MLOps systems need to solve to survive over time.

2.1 Relationships between DevOps and MLOps

DevOps is a set of practices based on Agile software development principles that appear in 2008/2009. They were created to approximate the developers and operators, encouraging communication, knowledge sharing and team collaboration to deliver software faster and with better quality. The main pillars, process, principles, or even best practices of DevOps are called Continuous Integration (CI) and Continuous Delivery (CD) (Gift & Deza, 2021; Kreuzberger et al., 2022; Mäkinen, 2021; Salvucci, 2021).

CI is a practice that aims to reduce the software development process cycle to identify errors as quickly as possible and correct them, consequently improving quality. To achieve this goal, code needs to be integrated at rapid and frequent intervals, facilitating software development automation. CD is a process that aims to test and deploy code to quickly and reliably deliver software enhancements to end users (Gift & Deza, 2021; John et al., 2021; Praveen Gujjar & Kumar, 2022; Salvucci, 2021; Symeonidis et al., 2022).

Since DevOps principles in software development have been used in the industry for more than 10 years, industry experts and academics are transferring this knowledge to automate and operationalize ML systems, which are also software with the peculiarity of having a component of ML model. This process to automate ML process using DevOps best practices is often called MLOps. Both, have the goal of reducing development time until the software goes into production, and the team is responsible for the entire lifetime of the solution with a collaborative team composed of developers and operators (Gift & Deza, 2021; Kreuzberger et al., 2022; Salvucci, 2021).

Nevertheless, there are important differences between them. While DevOps focuses on testing the code, MLOps also needs to test the data, validate the model, and ensure model quality. Another key difference is related to types of monitoring, when an organization's strategy changes, this can lead to ML model degradation, hence covariate-shift and pre-shift monitoring may be required, instead than monitoring latency in software traditional development. Finally, the CI and CD pipelines need to be built on MLOps, ensuring that CI includes data and the ML models, and the CD contains the training pipeline and the final model prediction. In addition, MLOps have one more key practice called Continuous Training (CT) with the aim of ensuring that ML models have an automatic trigger to retrain them when needed to improve model performance (Garg et al., 2021; John et al., 2021; Salvucci, 2021; Symeonidis et al., 2022).

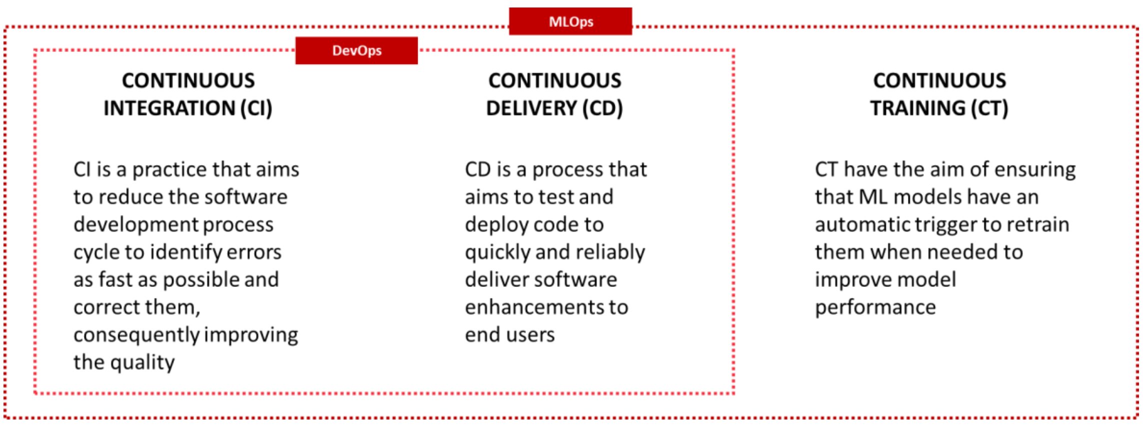

MLOps
DevOps
CONTINUOUS INTEGRATION (CI)
CI is a practice that aims to reduce the software development process cycle to identify errors as fast as possible and correct them, consequently improving the quality
CONTINUOUS DELIVERY (CD)
CD is a process that aims to test and deploy code to quickly and reliably deliver software enhancements to end users
CONTINUOUS TRAINING (CT)
CT have the aim of ensuring that ML models have an automatic trigger to retrain them when needed to improve model performance

Figure 1. Summary of DevOps and MLOps principles.

2.2 MLOps Requirements

In this subsection, a literature review of the most important state-of-the-art requirements to build a sustainable MLOps system using DevOps best practices adapted to the MLOps context was performed.

2.2.1 Versioning and Reproducibility

Data and model versioning are key challenges to building reliable ML models. Because of various reasons, training models have random weights, the data sources will eventually be updated, and the models will degrade, due to changing business decisions or data, over time. The main advantages of model versioning are enabling the rollback of a model if something unforeseen happens and allows different models to be used in different timeframes. On the other hand, data versioning is essential to follow the evolution of data over time. The union of both types of versioning make it possible to have model reproducibility, thus ensuring that the model can be shared with other teams and produce the exact results in a production environment (Lakshmanan et al., 2020; Ruf et al., 2021; Salvucci, 2021; Treveil et al., 2020).

2.2.2 Containerization and Orchestration

Containerization is a technology that uses a series of instructions to build a container called an image. This image encapsulates the code and its dependencies, allowing it to run on different computing platforms. Additionally, an image can reference multiple images, which can scale its usage. Also, container solution is a growing trend to be used as an alternative to virtual machines as various containers can run on just one operating system, which is more efficient in terms of computational power (Mäkinen, 2021; Salvucci, 2021; Treveil et al., 2020).

The key advantages of using containers are enabling the ML model to production without dependency on a single hardware and facilitating logging, monitoring, scaling and crash recovery, important aspects of MLOps architecture. Nevertheless, for large-scale applications, it is essential to have a container orchestration tool to make it simple to automate, deploy and communicate with other teams about containers (Garg et al., 2021; Silva, 2021).

Another essential type of orchestration in MLOps is the workflow, which uses directed acyclic graphs (DAGs) to represent the order of execution of steps considering relationships and dependencies. The main purpose of orchestration is to build the pipeline logic to determine which steps need to be performed and the expected results of them, for example it is possible to set a threshold to trigger the start of a pipeline if the accuracy of the ML model falls below this level or otherwise do not activate the pipeline (Kreuzberger et al., 2022; Lakshmanan et al., 2020).

2.2.3 Scalability

Scalability is a common challenge in ML workflows. This challenge can manifest itself in the stages of data collection and pre-processing, training, and service, and in the retraining model in the production phase. The dataset size is one of the essential requirements for choosing the right tool depending on the solution. Furthermore, the infrastructure necessary to process large amounts of data to train the ML models may be time-consuming and computationally expensive, so this needs to be considered, as it can be necessary, for example, to have a specific GPU for processing the ML workload (Lakshmanan et al., 2020; Treveil et al., 2020).

2.2.4 Monitoring

The monitoring step is crucial because ML models' performance degrades over time, and new data can generate the need to retrain the model in production, so if they do not have warnings or triggers, this can have a negative impact on business decisions. It is also a key factor to monitor the model metrics in production to detect the detachment between the model predictions and reality. In addition, there is a second level that the ML models need to be monitored, at a resource level, because in the production environment, it is essential to realize if the system is online, the speed of requests is being processed as expected, and the use of the CPU, RAM and network are as designed. (Salvucci, 2021; Sculley et al., 2015; Treveil et al., 2020). This second concern is typical of the DevOps topic, but the principle is the same for MLOps.

2.2.5 Explainability and Responsible AI

Assessing whether an ML model is ready for the production environment often uses metrics such as accuracy, precision, recall, and mean squared error. However, these metrics cannot provide the whole truth, as they do not tell why the ML model achieves the predictions, only measure how close the prediction is to the test set. The solution for this issue is a growing field in ML called interpretability or explainability of the models (Lakshmanan et al., 2020).

The field of explainability is intrinsically linked to the area of Responsible AI, which aims to find the best practices for building fairness in AI systems. This area has two main principles, and intentionality means that the AI projects must have unbiased data sources, which guarantees multiple checks and can be explained by humans. The second is accountability, which is a central control of all AI initiatives to understand which teams use what data sources to build which ML models, making it easier to comply with regulations and organize the insights into the business processes (Lakshmanan et al., 2020; Treveil et al., 2020).

Explainability can help with some of the big problems data scientists have in building and deploying ML models. Model debugging is one of them because it is essential to understand why the model performs better or worse in determined features to identify how to engineer new features, drop redundant ones and improve the overall model performance. Another problem that occurs during the monitoring phase after deployment is data drift, as it is important to explain how concept drift in features may impact the model outcomes and feature contributions. The last issues are model transparency and audit, as organizations seek transparency in the model prediction to make decisions with a certainty of effects to the end users and audit the data to comply with regulations and internal audit (Bhatt et al., 2020).

Explanation methods depend on three main factors: data modality, model type and prediction task (Klaise et al., 2020). The first refers to the complexity of the dataset. If it contains different types of data like images and text, they have more complexity because these types of data have different statistical properties. Second, some models are easier to explain, such as linear regression or decision trees. However, models like neural networks, known as black box models, are hard to explain. Finally, whether the prediction task is, classification or regression must be considered, as it affects the calculation of explanation methods.

The most used techniques of the explanation methods are partial dependence plots which identify the marginal impact of the features on the predicted outcome; subpopulation analyses, which explain how the model handles specific subpopulations; individual model predictions, normally using Shapley values, which show how the value of each feature contributes to a specific prediction; what if analysis, that is useful to explain the sensitivity of the prediction due to the inputs (Treveil et al., 2020).

2.2.6 MLOps Maturity Model

To implement the fully automated MLOps pipeline using the CI, CD and CT principles cited in subsection 2.1, there are some steps that organizations or ML professionals need to climb, called the MLOps maturity model.

Two of the main technology companies in the ML field, Google, and Microsoft, developed their own MLOps maturity model framework. It is explained both because the concepts will be important further in the thesis.

According to Google (2023), there are 3 levels of MLOps maturity. Level 0: manual process, level 1: ML pipeline automation and level 2: CI/CD pipeline automation.

At level 0: manual process, every process is manual, from data analysis to model validation, normally handled by data scientists using notebooks to experiment until they develop a viable model. Furthermore, the good MLOps practices cited in subsection 2.2 do not exist at this level, as data and model versioning, pipeline orchestration, model monitoring, explainability, CI and CD are completely ignored. This level is still very common in organizations but is only sufficient if the model is rarely changed or trained. For real world applications the model often fails due to changes in data and business decisions.

At level 1: ML pipeline automation, the main purpose is to develop a pipeline automation to achieve CT. At level 1 is possible to have faster experiments thanks to the pipeline orchestration that allows to prepare the whole pipeline towards production. Through the CT principle, the ML model is trained with new data in a production environment, using pipeline triggers. The symmetry among experimental and operational environments is also achieved using the same pipeline in pre-production and production environments. Furthermore, the pipeline construction needs to have reusable, composable, and potentially shareable components in the ML pipeline, and must be containerized to be reproducible. Howsoever is possible to continuous delivery the ML models to production.

Some additional components can be used at level 1 to enable CT, such as data and model validation, feature store, which is a centralized repository where features can be accessed for training, metadata management that stores information from each execution of the ML pipeline to debug errors and ML pipeline triggers which are automatic triggers to retrain ML models with new data according to defined criteria.

In level 2: CI/CD pipeline automation, there are 6 stages to achieve a fully automated CI/CD pipeline. The first is development and experimentation, where the ML professional tries orchestrating different algorithms and ML models. The output is the source code of the ML pipeline that goes to a source repository. The second stage is continuous pipeline integration, where the source code is built through various tests, and the outputs are pipeline components, such as packages and artifacts.

After that, in stage 3: pipeline continuous delivery, pipeline components from the CI stage are deployed to the target environment. The fourth stage is automated triggering, where the pipeline runs automatically in production due to a trigger or schedule. The trained model output from this step goes to the model registry.

The next stage is model continuous delivery, where the trained model is served as a prediction service, finally, in the monitoring stage, collects live data of the performance of the model. The final output is a trigger to return to the first stage or to run the pipeline in stage four.

According to Microsoft (2023), There are five levels of MLOps maturity. Level 0: No MLOps, level 1: DevOps but no MLOps, level 2: Automated Training, level 3: Automated Model Deployment and level 4: Full MLOps Automated Operations.

Levels 0, 2 and 4 are very similar to the Google maturity framework, however Microsoft added two intermediate levels. At level 1: DevOps but no MLOps, is also a manual process, but the main differences are that data is collected automatically and exist basic integration tests for the model. At level 3: Automated Model Deployment, the key differences from

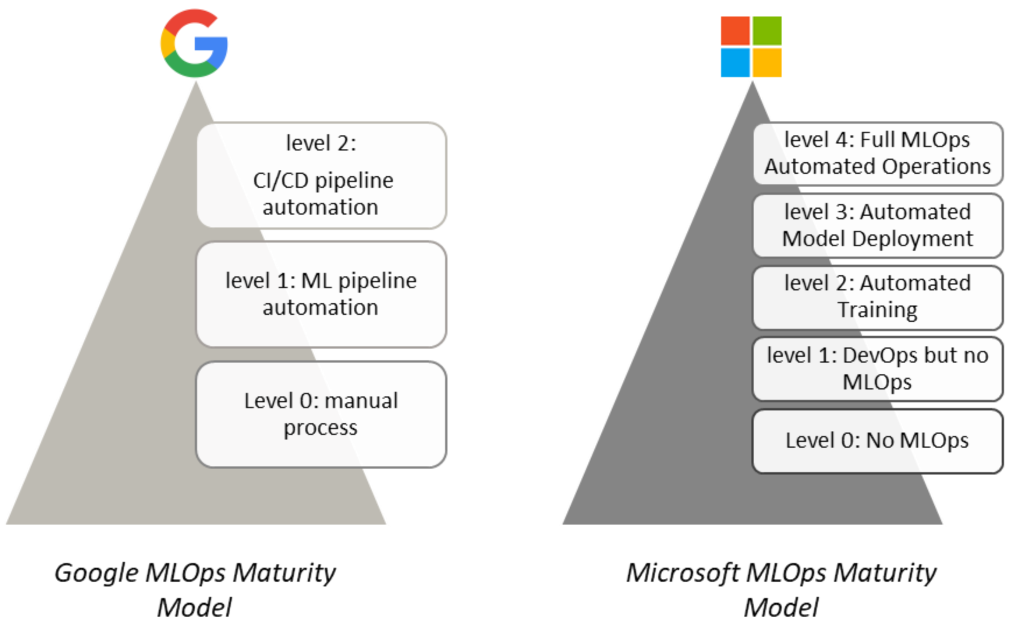

level 2:
CI/CD pipeline automation
level 1: ML pipeline automation
Level 0: manual process
Google MLOps Maturity Model
level 4: Full MLOps Automated Operations
level 3: Automated Model Deployment
level 2: Automated Training
level 1: DevOps but no MLOps
Level 0: No MLOps
Microsoft MLOps Maturity Model

Figure 2. Comparison of MLOps Maturity Models.

level 2 is that an automatic deployment is managed by CI/CD pipeline and each model release has unit and integration tests.

In this master's thesis, the concepts adopted follow the Google maturity model framework due to the simplicity and better detail provided in figure 3. In summary, the two frameworks are very similar, but Microsoft divides it into two more levels, which makes the application more complex, in this case it is a simple project that will be executed by only one person.

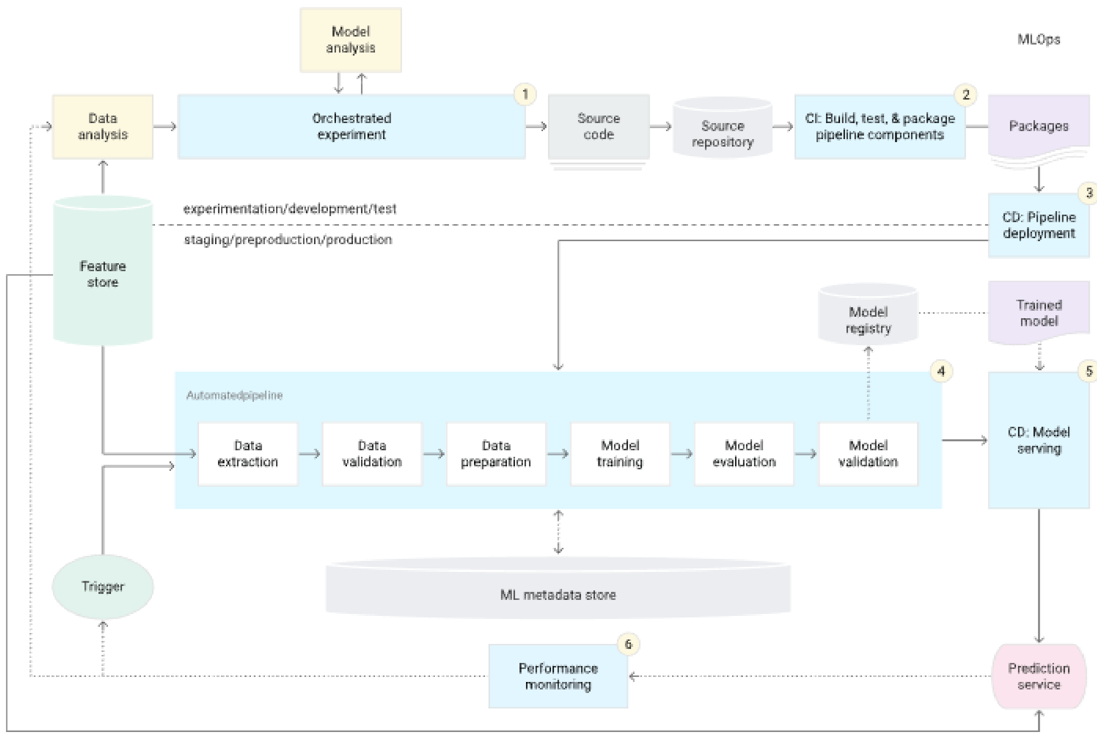


Figure 3. MLOps level 2: CI/CD automated pipeline (Google, 2023).

2.3 ML End-To-End Life Cycle

There was no standard ML lifecycle process model. For this reason, the Cross-Industry Standard Process model for Data Mining (CRISP-DM) is often used as an alternative. However, this methodology is not complete because of two reasons. The first is that ML models' performance degrades over time, so after deployment is necessary to have a monitoring and maintenance phase. The second is that CRISP-DM does not have a quality assurance methodology, which is essential to mitigate risks of each phase of the development and deployment of ML applications (Studer et al., 2020).

In this subsection, the phases of Cross-Industry Standard Process model for the development of Machine Learning applications with Quality assurance methodology (CRISP-ML(Q)) proposed by Studer et al. (2020) are detailed. Since this methodology was made to solve the issues mentioned above, this thesis adapts its phases to fulfill the objectives consistently.

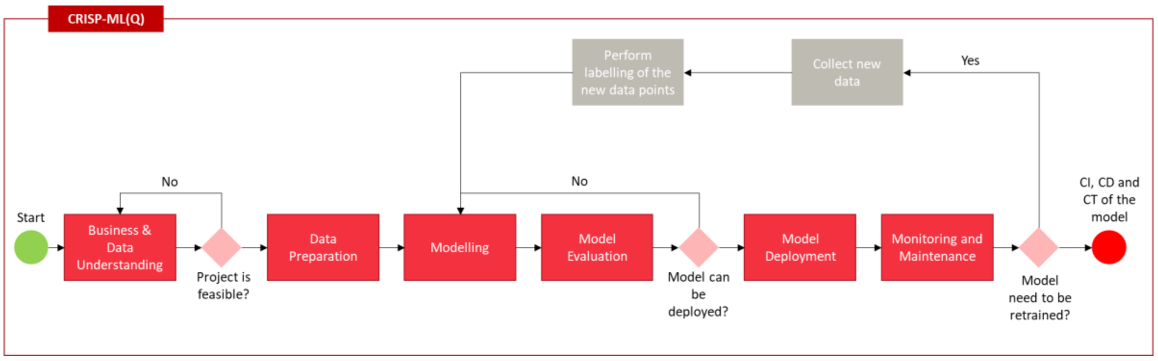

CRISP-ML(Q)
Perform labelling of the new data points
Collect new data
Yes
No
No
CI, CD and CT of the model
Start
Business & Data Understanding
Project is feasible?
Data Preparation
Modelling
Model Evaluation
Model can be deployed?
Model Deployment
Monitoring and Maintenance
Model need to be retrained?

Figure 4. CRISP-ML(Q) framework adapted from Studer et al. (2020).

2.3.1 Business & Data Understanding

The first step in developing ML applications is to ensure the project is viable. Nevertheless, before that, it is necessary to define the scope of the ML application and define the success criteria of the business and ML model.

So, what is usually done is to build a Proof of Concept (PoC) to understand the use of the ML model in a reduced scope. In addition, it is also relevant to verify the legal constraints and the requirements of the application. After feasibility confirmation, the data can be collected, and the data quality is checked. Finally, it is crucial to document all the aspects of the data, such as requirements, statistical properties etc. To serve as the basis for quality assurance in ML model development.

2.3.2 Data Preparation

The data preparation phase aims to set up a data set to the next phase, but this process is not linear. For example, if is error data is found in modelling, it is necessary to come back to adjust the data. This phase starts with selecting data, which includes feature selection, where the best features are selected through a filter, wrapper or embedded methods and bad quality data are discarded. Furthermore, it might be essential to deal with unbalanced classes using over-sampling or under-sampling strategies for data selection.

The next step is data cleaning to detect and correct errors in the data, which can include noise reduction, and elimination of missing values through data imputation techniques, such as imputing mean or median values, interpolated, or replacing by other special value.

After that, in the construction stage, it may be necessary, depending on the ML task, to perform feature engineering and data augmentation, using techniques like PCA, one-hot encoding and clustering. The final step is data standardization, where the input data formats, and the normalization process are unified to avoid the risk of bias to features on larger scales.

2.3.3 Modelling

The main objective in the modelling phase is to build an ML model or even several models that fit the business criteria and can meet the requirements and constraints established in the first phase. To build a modelling strategy, it is quintessential to define the quality measures of the model. Depending on the ML task, the six measures of ML models can have different weights, they are: performance, robustness, scalability, explainability, model complexity and resource demand.

Furthermore, in the modelling stage, there are model selection, model specialization, incorporating domain knowledge, and model training tasks. Some optional tasks that depend on the ML model are using unlabeled data and pre-trained models, model compression and ensemble methods to outperform individual models.

In this phase, to obtain a robust ML solution, it is crucial to ensure the reproducibility of the model. Two levels of reproducibility need to be ensured, the method and the results. It is important to emphasize that it is part of the method reproducibility to produce experimental documentation containing the model changes and their causes to improve the model quality measures.

2.3.4 Model Evaluation

The model evaluation mainly consists of validating the performance of the trained ML models on a test set, which is the pre-split portion of the original dataset, to be able to measure performance metrics. Furthermore, the robustness of the models must be addressed using noisy or erroneous input data. It is also a best practice to increase explainability to end users to provide trust and facilitate decision-making based on the models.

Finally, there is the decision, which can be automatic or manual, if the ML model satisfies the success criteria to be deployed. Otherwise, the process returns to the modelling phase or even ends in this phase.

2.3.5 Model Deployment

The model deployment phase starts by defining inference hardware. Here needs to be chosen the requirements of the hardware, and solving decisions such as the availability of CPU and GPU or whether it is more efficient cloud services or an embedded system. With that solved, it is essential to evaluate the models under production condition to understand if the production data is like the training data.

The next steps are to ensure that the users are comfortable with the final solution. This can be tested via a prototype. It is also important to minimize the risks of errors. One common alternative is to roll back the model to a previous version if something unforeseen happens.

Finally, in the deployment strategy task, it is essential to perform an incremental roll-out of the model, as it is possible that errors still exist. Thus, the cost of correcting them can be minimized.

2.3.6 Monitoring and Maintenance

After deploying the ML model, it is essential to monitor and maintain the models' performance. ML models' performance decreases over time because their evaluation is done based on train data, but real world data can change the model accuracy very quickly (Treveil et al., 2020). Additionally, model performance can be degraded by hardware performance and updates to the software stack.

Therefore, the best practice to avoid model degradation is continuous monitoring to decide whether the model needs to be retrained as quickly as possible. This leads to the next task, updating the model with new data, returning to the modelling phase. Adjusting the current model with the new data is recommended to save time. Finally, the automation of the update task allows achieving the CI, CD and CT of the model.

2.4 MLOps Main Open-Source Tools

Open-source MLOps tools are used by half of all IT companies worldwide, and that number is expected to grow to over 65% by 2023 as the open-source stack provides an easier, more reliable, and rapid delivery to manage end-to-end ML application lifecycle. However, no tool automates the entire ML lifecycle. They are specialists in different tasks and may have overlapping functionality. In addition, they often require an environment in Python and R languages (Hewage & Meedeniya, 2022; Ruf et al., 2021; Symeonidis et al., 2022).

Table 1, only open-source tools are filtered and compared in terms of functionality, adapted from the works of Hewage & Meedeniya (2022), Ruf et al. (2021) and Symeonidis et al. (2022). For the comparison of the MLOps tools, the features Data Versioning (DV), Experimentation (EX), Code Versioning (CV), Experiment Tracking (ET), Pipeline Orchestration (PO), Artifact Tracking (AT), Model Registry/Versioning (MRV), Model Deployment/Serving (MDS), and Model Monitoring (MM) were chosen.

	MLOps Open-Source Stack								
Tools	**DV**	**EX**	**CV**	**ET**	**PO**	**AT**	**MRV**	**MDS**	**MM**
Git			x						
Git LFS	x								
GitLab			x						
Dolt	x								
LakeFs	x								
DVC	x			x	x				
Jupyter Notebook		x							
Google Colab		x							
Tensor Board				x					
GitHub Actions					x				
Apache Airflow					x				
ZenML					x				
Kedro				x	x				
Polyaxon				x	x	x			
Kubeflow		x			x	x			
MLflow				x		x	x	x	
BentoML							x	x	
Clear ML				x	x	x	x	x	x
MLrun					x	x	x	x	x
ModelDB				x			x	x	x
TensorFlow Serving								x	
Streamlit								x	
Heroku								x	
Evidently									x
Prometheus									x
Grafana									x

Table 1. Main MLOps Open-Source Tools.

In Section 3, it is described which of the above MLOps open-source tools was chosen to build the ideal stack used in the rest of the development of the thesis.

2.5 MLOps Key Challenges

ML Systems have many technical challenges because the maintenance phase after deployment is often difficult and expensive over time. Furthermore, these systems have all the traditional software development issues plus the special concerns of the ML field. These problems can be difficult to notice as they do not occur at code level but at the system level (Sculley et al., 2015).

The technical problems can be classified as eroded boundaries due to the complex models, data dependencies costs, feedback loops, Anti-patterns in ML systems, configuration debt and changes in the external world. The first challenge's main example is called entanglement, which is related to the impossibility of independence of the features, entangling them. This is valid for hyperparameters, data selection, sampling methods etc. However, there are two strategies to mitigate it: isolate the models and serve ensembles or detect the changes in prediction behavior when they happen (Sculley et al., 2015).

The data dependencies problem emerges from the matter that ML code is only a tiny part of the ML system, and complex data dependencies can be built without a tool for tracking the unstable and underutilized dependencies. This issue is also related to data quality, as it is essential to understand the quality of the sources of data to train the ML models with consistency (Lakshmanan et al., 2020; Sculley et al., 2015).

Feedback loops are necessary to have a real-time updating of the ML systems, however, they can influence their behavior, which can be even more difficult to notice if it occurs gradually. This effect can happen directly, where the model influences the selection of its own future training data or indirectly (hidden), where two systems influence each other. To mitigate this, the best practices are to use randomization or isolate the parts of the data influenced by the ML model (Sculley et al., 2015).

Most ML systems are used just as an infrastructure and a tiny part to learn or prediction, and this can lead to import problems in design patterns. Two common reasons are known as glue code and pipeline jungles. The first means the use of generic packages code that, over time, freezes the ML system to the logic of specific packages, which can be mitigated using APIs that allows infrastructure reusability (Sculley et al., 2015).

The second is a kind of glue code for pipelines that, over time, becomes so complex that it needs expensive integration tests. To have efficient pipelines is essential to build compatible these three main pipelines: data manipulation, model creation and deployment. Also, it is crucial to use the same data to preprocess all of them (Sculley et al., 2015; Symeonidis et al., 2022).

Configuration debt in ML systems is a very common challenge because there are many configurable options, such as the features to select, specific algorithms settings and data selection. All these possibilities make the configurations hard to change, when necessary, which can be expensive and time-consuming (Sculley et al., 2015).

The ML system must deal with the inevitable changes in the real world, thus incurring maintenance costs. For this reason, it is essential to continuously monitor the accuracy of the models and the entire system. Data monitoring helps to find outliers and drift to train the model with correct data. On the other hand, model monitoring for high maturity systems monitors in addition to model accuracy, sustainability, robustness, fairness and explainability. Also, as the external world changes in real-time, it's important to build automated response systems to avoid manual work (Sculley et al., 2015; Symeonidis et al., 2022).

In addition to the technical challenges, there are also organizational ones. This type of challenge involves MLOps need cross-functional heterogenous teams to accomplish their goals, but it can lead to lower productivity in the long run. One of the causes is the lack of great experts in ML field, and another is that the organization's culture does not promote good communication practices between teams and can have multiple definitions of what a successful ML model is for each of them. For these reasons, it is quintessential to transform the culture into a product-oriented mindset to improve business outcomes (Kreuzberger et al., 2022; Lakshmanan et al., 2020; Sculley et al., 2015).

2.6 Literature GAP

The core MLOps topics were covered in this chapter, namely the prerequisites for performing MLOps, CRISP-ML(Q) the ML end-to-end life cycle approach, some of the key MLOps open-source tools, and the main technical issues that arise when the ML model is deployed.

The main GAP found in the literature review is a lack of straightforward real world end-to-end MLOps projects that can be done using the best practices found in the MLOps requirements subsection and following the steps suggested by CRISP-ML(Q) methodology. This is necessary to have a professional automated CI/CD pipeline to deploy ML models with consistency rather than just running the ML model in a notebook environment.

3 Data and Methodology

This section presents the data used in this work, including how the data was collected and the description of the variables. Also, the work methodology to achieve the objectives defined in section one is described. In addition, the MLOps stack chosen to develop, deploy and monitor the ML pipeline is presented.

3.1 Data

There are two common model deployment methods: model-as-a-service through Representational state transfer (REST) API endpoints, and embedded model (Ruf et al., 2021; Treveil et al., 2020).

These kinds of deployments allow easy interaction with input data, so it is possible to use this technology to extract the data of this project. The data was extracted from Spotify Web API, using REST principles. According to Spotify (2023), it provides metadata about tracks, artists, albums, and playlists from the Spotify Data Catalogue.

3.1.1 Collection

To perform data collection from Spotify API, the first step is to get the authorization credentials from the user's account of Spotify or create one if necessary. The next, is installing and using "Spotipy" Python library which provides an easier interface with the API interface.

One good way to extract the data is through playlists; choosing songs by track or artist is more efficient. The playlists extracted were the "Top Hits" by year made by Spotify, because it contains very popular songs from different years without duplicates tracks. Thus, the Top Hits from 2005 to 2022, and each with 100 songs, making a total dataset of 1800 different tracks and additional 100 from the most updated top 100 playlist to be used as new data.

3.1.2 Description

The variables extracted from Spotify Web API are described in table 2. The mood column was created to be the target variable which indicates 1 for the happy songs and 0 for the sad songs, based on the valence feature that will be excluded in further analysis.

#	Variables	Description
1	artist	Singer or band name
2	album	Album name
3	track_name	Song name
4	release_date	Release date of the song
5	popularity	Popularity of the song from 0 to 100
6	genres	Name of the first genre associated with the artist
7	sub-genres	Name of the second genre associated with the artist
8	explicit	Boolean value if the track has explicit lyrics
9	followers	Number of the followers of the artist
10	track_id	The unique ID of the track
11	danceability	Describes how suitable a track is for dancing from 0 to 1
12	energy	Represents a perceptual measure of intensity and activity from 0 to 1
13	key	Integers map to pitches using standard Pitch Class notation, if no key = -1
14	loudness	The overall loudness of a track in decibels (dB)
15	mode	Indicates the modality, major (1) or minor (0) of a track
16	speechiness	Detects the presence of spoken words in a track from 0 to 1
17	instrumentalness	Predicts whether a track contains no vocals from 0 to 1
18	liveness	Detects the probability of presence of an audience in the recording from 0 to 1
19	valence	Describes the musical positiveness conveyed by a track from 0 to 1
20	tempo	The overall estimated tempo of a track in beats per minute (BPM)
21	duration_ms	Duration of the song in milliseconds
22	time_signature	Notational convention to specify how many beats are in each bar from 3 to 7
23	mood	Indicates mood of the track 1 (happy), if the valence is greater or equal to 0.5 and 0 otherwise

Table 2. Dataset description adapted from Spotify (2023).

3.2 Methodology

The methodology will adapt the steps explained in subsection 2.3, based on CRISP-ML(Q) from Studer et al. (2020), the ML lifecycle framework.

1. **Business & Data Understanding:** in this phase, it is important to emphasize that

the goal of this ML task is to perform the techniques of supervised learning to classify the tracks in mood playlists, namely happy and sad playlists, using the target variable "mood". Also, an Exploratory Data Analysis (EDA) is performed to understand the behavior of the variables and analyze the data quality.

2. **Data Preparation:** In order to have data prepared to be modelled, it is crucial to perform data cleaning, feature engineering to select the variables, data augmentation and finally, the standardization of the variables, presented in subsection 3.1.2, for the model.

3. **Modelling:** As explained in subsection 2.3.3, the first step is to define the quality metrics of the model. After that the model is trained, in this case using the classification algorithms from supervised learning, and the last step is to improve the performance of the mode, if necessary, with AutoML or ensemble methods.

4. **Model Evaluation:** In this phase, the trained model has the performance evaluated, after the robustness is determined, and the model explainability can be improved by the techniques of subsection 2.2.5. Finally, the model needs to be ready to be deployed to advance to the next level.

5. **Model Deployment:** First the model is deployed; next the models are evaluated under production conditions, and to minimize error rollback points are built in the ML pipeline.

6. **Monitoring and Maintenance:** In the final phase, is monitored the performance of the model is monitored over time and create triggers to automatically retrain the models when needed to obtain the CI/CD pipeline automation, explained in subsection 2.2.6.

To finalize the methodology, the specific Spotify API was chosen because it is one of the most well-known open APIs used for training different machine learning models. Additionally, because it is an API rather than a simple csv file, it has more flexibility to scale the same pipeline, which is more efficient for MLOps project. Also, this simple ML classification assignment was chosen because it speeds up pipeline tests since it doesn't necessitate a lot of complex calculations like a deep learning challenge for example.

3.3 MLOps Open-Source Stack

The right choice of tools is essential to have an efficient MLOps system in each functionality and the connectivity between them. There is no silver bullet to accomplish this task. However, it is possible to follow some guidelines. It is often recommended to use fewer tools with high compatibility to be easier to manage, so the main challenge is to find the balance between flexibility and stack compatibility (Symeonidis et al., 2022).

	MLOps Open-Source Stack								
Tools	**DV**	**EX**	**CV**	**ET**	**PO**	**AT**	**MRV**	**MDS**	**MM**
Git			x						
DVC	x			x	x				
GitHub Actions					x				
Jupyter Notebook		x							
MLflow				x		x	x	x	
Streamlit								x	
Evidently									x

Table 3. Chosen MLOps Open-Source Stack.

Table 3 shows the filtered table presented in Section 2, with only the chosen MLOps tools. It followed the guidelines of choosing fewer tools as possible with easy connectivity between them, preserving all functionalities to have a complete ML pipeline that allows the management of the ML lifecycle.

In Figure 5, it is possible to visualize better the connections between the tools and the saving of tools made, mainly when choosing DVC and MLflow, which are flexible tools with multiple functionalities. Also, Git as code versioning tool and Jupyter notebook as experimentation environment were chosen because they are standards in ML filled. In addition, the language of this project is Python because it is the most common language for machine learning projects, providing numerous libraries and methods to perform ML tasks.

Furthermore, GitHub Actions was chosen because it is a tool inside GitHub repository that provides an easy way to create CI/CD pipelines. Streamlit also is convenient to integrate with a GitHub account and has a simple interface to deploy ML applications in cloud. Lastly, Evidently has predefined monitoring reports that can be used with few lines of code.

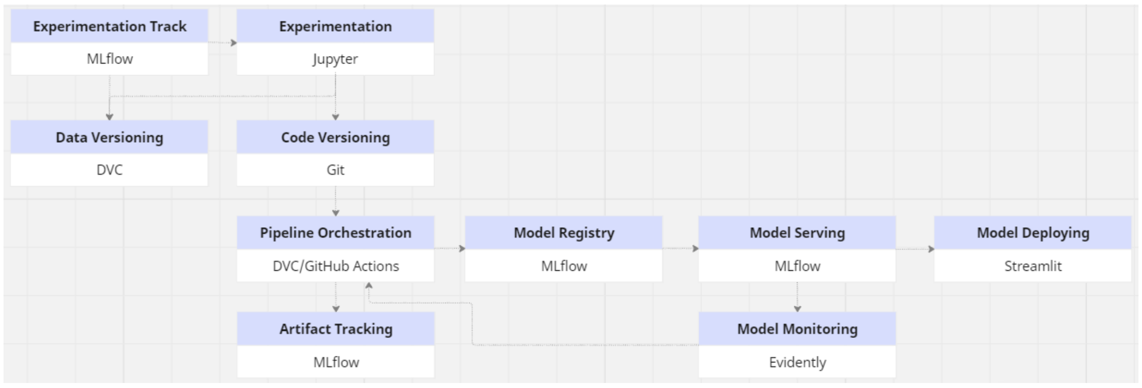


Figure 5. Framework of the chosen MLOps stack.

In the next chapter, an in-depth examination of the main issues explored in the literature review will be given. By answering the questions below, it is hoped to gain a better understanding of the subject and provide useful information. Each issue is thoroughly explored, evaluated, and debated, resulting in a complete and informative chapter.

1. How can the CRISP-ML(Q) technique be used to create a consistent ML project?
2. How and why to use every open-source technology listed in table 3?
3. How are the open-source tools linked to one another?
4. What are the best models for categorizing Spotify tracks as happy or sad?
5. How to create a CI/CD pipeline and deploy a machine learning application?
6. How should the ML model be efficiently monitored and retrained?
7. How should the best ML model results be explained using the MLOps requirement of explanability and responsible AI?

4 Results and discussion

The purpose of this chapter is the description the results of this dissertation, organized by the topics explained in the methodology based on CRISP-ML(Q).

Furthermore, the choices of the MLOps stack are detailed with the aim of better understand the functionalities of each tool and how they integrate, using Google's maturity model showed in subsection 2.2.6, to achieve the goals established in the first chapter.

4.1 Business & Data Understanding

In this phase was planned the complete development of the ML application based on the MLOps stack chosen and the framework of the Google maturity model. It also created the Python script to extract the data from Spotify API and finally performed the EDA from the features and target variables to get insights from the raw data.

4.1.1 Planning of the project

The planning of this project needed to consider the ML tools described in the previous chapter and the integration between them. Figure 6 shows a modified version of Google's maturity model with the chosen MLOps stack and the necessary remote repositories that will be used.

The orchestrated experiments will be done using MLflow Tracking component, which allows easy comparison of the model parameters and metrics experiments in a centralized UI. To help with the experimentation, the data and model analysis will be performed in Jupyter notebooks.

In MLOps projects, it is a very important aspect to have all the code versioned, for this will be used the Git version control tool together with GitHub as a remote repository to not lose any historical information during the code update steps of the project.

As mentioned in Chapter 2, the CI/CD pipeline automation is a fundamental principle that came from DevOps to MLOps and needs to be done to reach the most advanced level of MLOps. In this project, this step will be done using GitHub Actions inside the GitHub repository and, as a dependency, will demand the Python packages to be installed that will be available in a text file named "requirements.txt".

The CI/CD pipeline also will need to call the pipeline to work properly. The pipeline is going to be built some Python scripts files and, together with DVC, besides versioning the data of the project, have the functionality that enables to create DAGs, which makes the pipeline automated, simpler to maintain and easy to use in GitHub Actions.

All the data and metadata outputs generated by the pipeline will be stored in DagsHub remote repository connected with GitHub repository and in the MLflow server offered by DagsHub, which uses the Model Registry component. It is important to point out, as shown in figure 6, that changes in the feature store make the automated pipeline run again.

In the final steps will be necessary to serve the ML model using one more time MLflow components to deploy the model in Steamlit, which allows to easily build ML applications, to be used as a prediction service in Python language and make deployment in Streamlit public cloud connected directly to the GitHub repository. The last tool is Evidently which will comply with the performance monitoring functionality, which generates html reports for better understanding when the performance of the ML model degrades to manually trigger the pipeline to start the necessary steps in the pipeline.

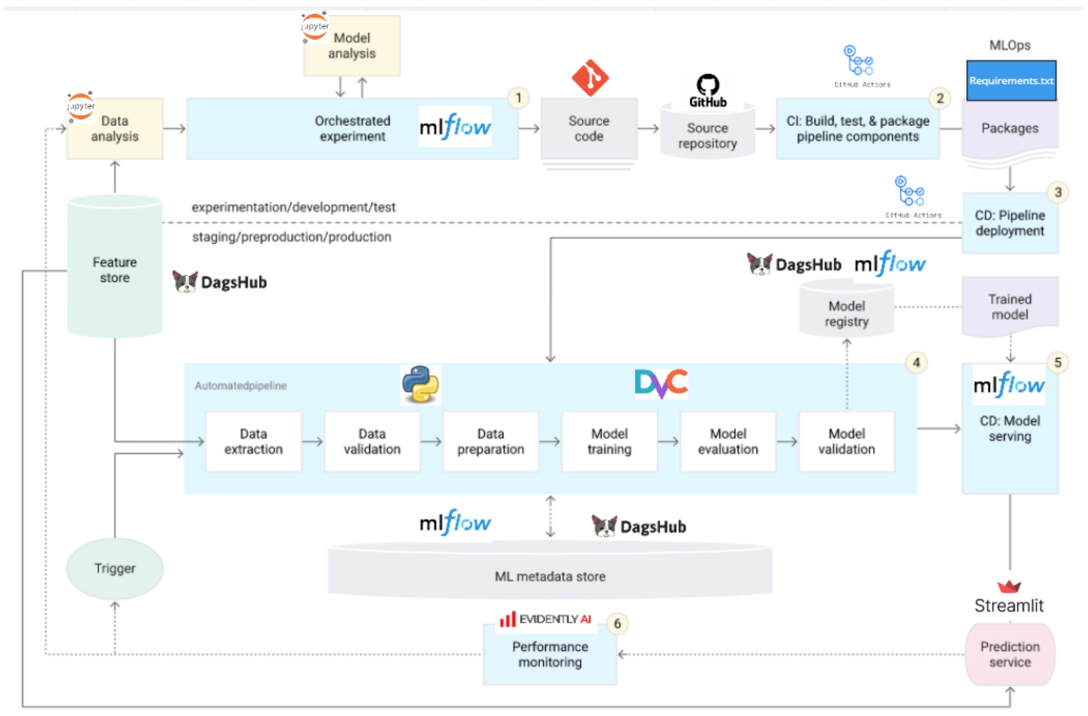


Figure 6. MLOps stack in adapted Google's maturity model framework (Google, 2023).

4.1.2 Data Extraction

The data extraction script was developed in six main steps.

1. Spotify account authentication through "Spotipy" library. To accomplish this task was necessary to create a ".env" local file to secure the login and password of my personal Spotify account because all the final code is pushed to GitHub public remote repository, which is visible to everyone.

2. Get the playlists links from Spotify application. This was a manual step necessary to get the top 100 tracks playlists between the years 2005 and 2022 and 100 additional tracks from top 100 songs playlist, which updated the songs every week, to serve as the new updates data to test the ML models over the time.

3. Development of two functions to build the raw data frame, in the first function is selected all the features and the target variables better described in subsection 3.1.2 from the API, next a loop is created to extract every variable from each track of one playlist and in the end is created a data frame. The second function was made for the necessity of a loop creation for multiple playlists and concatenating them in a unique data frame.

4. Execution of both functions, for the 2005 to 2022 top 100 tracks playlists, the second function is used to result in 1800 rows data frame. For the new songs, as it is only one playlist, the first function was enough to generate the 100 rows data frame.

5. Function to clean the genres and sub-genres features. This step was essential because the genres were extracted as an array with all the genres from each track. So, this function created the genres and sub-genres columns with the top two genres from the songs.

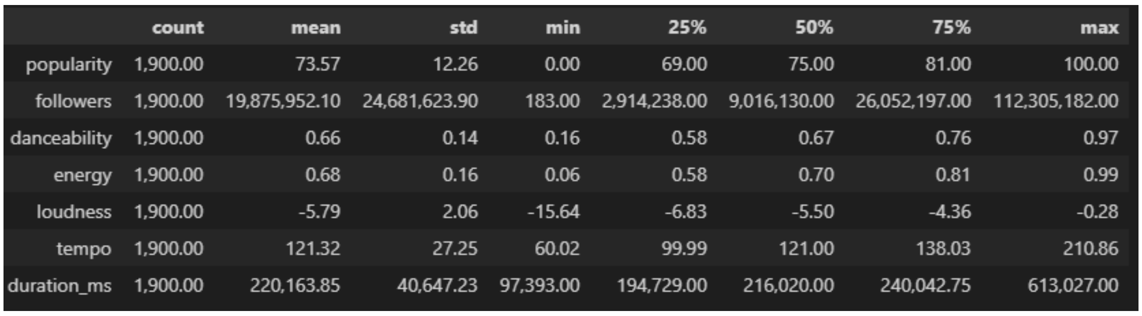
6. Export the data frames to csv files. This final step exports both data frames individually in two csv files to be used in the next phases of the project.

4.1.3 Exploratory Data Analysis

The Exploratory Data Analysis (EDA) consisted of analyzing the raw data generated in the data extraction phase. For this analysis, both csv files were imported and concatenated, which generated a data frame with 1900 tracks. The objective is to better understand the data to make the necessary treatments in the data preparation phase.

The first analysis was an overview of the content of the study dataset. It was discovered the top 20 artists and albums using the number of tracks and top 20 most followed artists, which can be seen in figures 28, 29 and 30 in appendix.

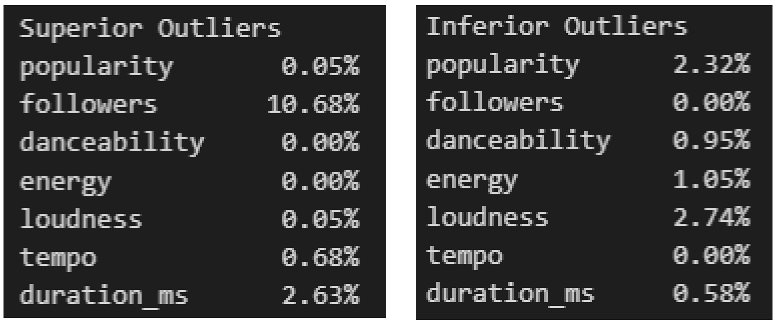
To provide a comprehensive analysis of the numerical features was made a simple statistical description. In the figure 7, it is easy to realize that "followers" and "duration_ms" have the greatness of values much higher than the other variables and "loudness" have only negative values. These insights will need to be addressed in the future of this project.



	count	mean	std	min	25%	50%	75%	max
popularity	1,900.00	73.57	12.26	0.00	69.00	75.00	81.00	100.00
followers	1,900.00	19,875,952.10	24,681,623.90	183.00	2,914,238.00	9,016,130.00	26,052,197.00	112,305,182.00
danceability	1,900.00	0.66	0.14	0.16	0.58	0.67	0.76	0.97
energy	1,900.00	0.68	0.16	0.06	0.58	0.70	0.81	0.99
loudness	1,900.00	-5.79	2.06	-15.64	-6.83	-5.50	-4.36	-0.28
tempo	1,900.00	121.32	27.25	60.02	99.99	121.00	138.03	210.86
duration_ms	1,900.00	220,163.85	40,647.23	97,393.00	194,729.00	216,020.00	240,042.75	613,027.00

Figure 7. Descriptive statistics of the numerical variables.

The next step concerning numerical features is outlier detection. This analysis consists of detecting the upper and lower outliers for each variable using the Interquartile Range (IQR) method, whose upper outliers are those higher than the third quartile plus 1.5 times the IQR and the lower outliers below the first quartile minus 1.5 times the IQR. In figure 8, the only feature that deserves some attention is the "followers" with more than 10% of outliers, but it is not possible to exclude outliers because they have a small number of artists with a greater number of followers and at the same time tracks, as can be seen in figures 28 and 30 in the appendix, so it is preferable to keep them. The other variables only have a small number of outliers, for this are not very important for deepening.



```
Superior Outliers
popularity        0.05%
followers        10.68%
danceability      0.00%
energy            0.00%
loudness          0.05%
tempo             0.68%
duration_ms       2.63%
```

```
Inferior Outliers
popularity       2.32%
followers        0.00%
danceability     0.95%
energy           1.05%
loudness         2.74%
tempo            0.00%
duration_ms      0.58%
```

Figure 8. Outliers' detection tables.

Another common analysis is the identification of missing values, but in this dataset, only the genres and sub-genres have approximately 1 % and 16 % of missing values respectively, which will be addressed in the data preparation part.

In order to improve the understanding of the relationship between the numerical variables and their relationship with the target, it was plotted Pearson's correlation matrix in figure 9.

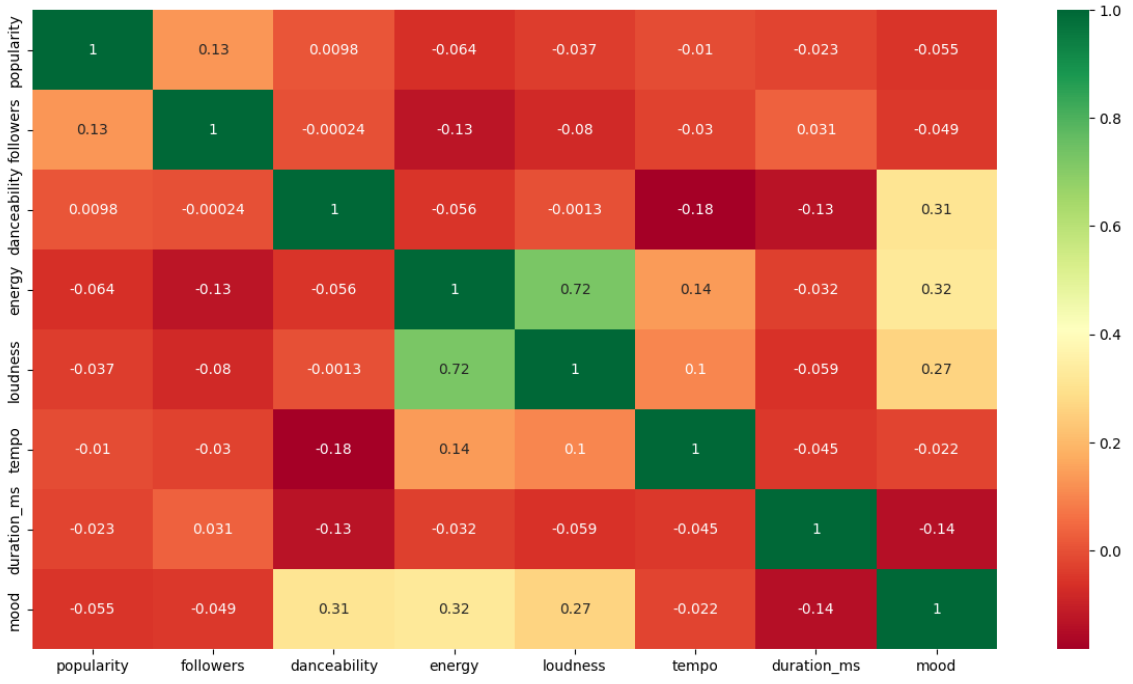


Figure 9. Pearson's correlation matrix of numerical features.

Most of the numerical variables do not have any relevant relationship with each other, except for the energy with loudness with a 0,72 positive index, what makes sense because usually loud music tends to be energetic. In addition, the features with the strongest relationship with the target are danceability, energy and loudness. However, even they are not very high.

Finally, the distribution of the numerical features and their behavior in relation to the target was examined. For this reason, was made the pair plot in figure 10 which orange represents music that tends to be happy and blue otherwise. It is possible to realize that the behavior of the three variables with the strongest positive relationships with the target, which were found in Pearson's correlation matrix, are also visible in the right tail of the distribution plots. The higher the values, the greater the tendency to be a happy song.

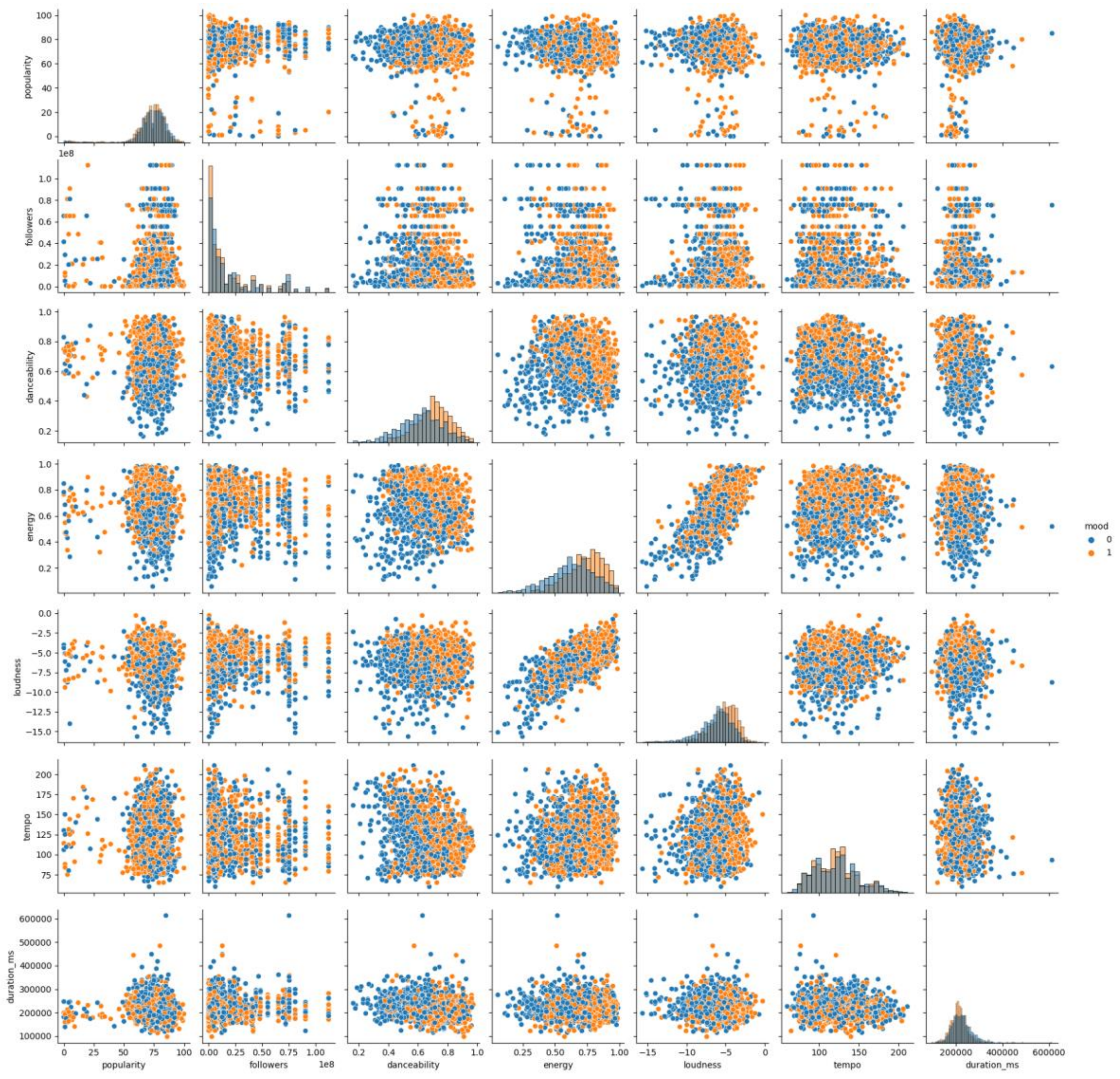


Figure 10. Pair plots of numerical features

One key aspect of EDA is to understand the target distribution to identify if there is a relevant imbalance between the classes. As it is shown in figure 11, the classes are close to being perfectly balanced, the mood equals "1" corresponds to 51,74 % and equals "0" to 48,26 % of the data. Therefore, the imbalance issue is not relevant to this project.

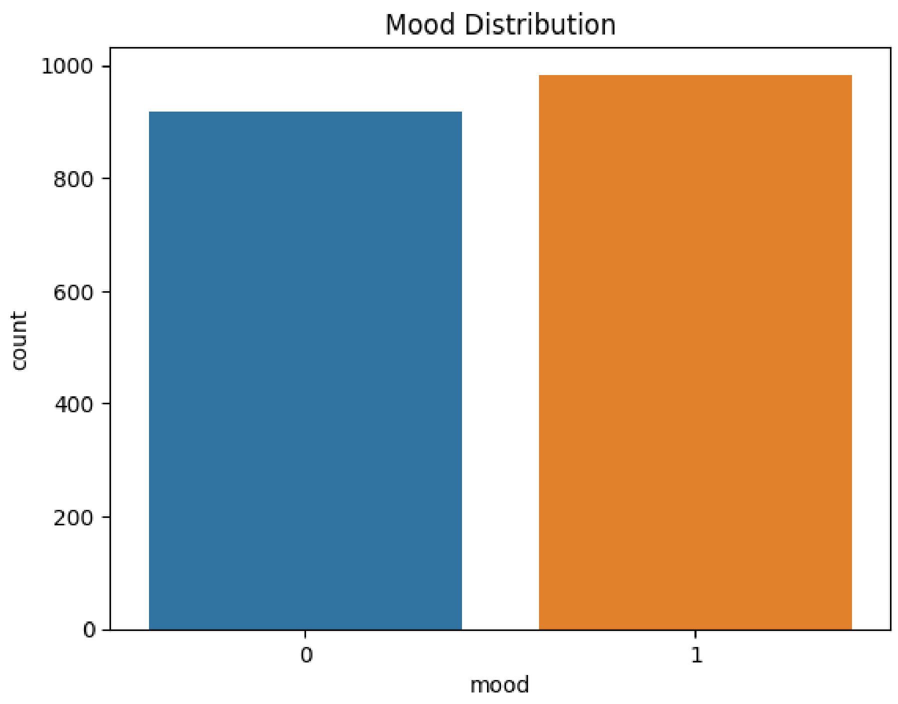


Figure 11. Target variable distribution.

Another fundamental aspect that requires examination is the relationship between the categorical features and the target. Three distributions are plotted in figure 12 to show that the features "speechiness", "liveness", and "instrumentalness" are constants or practically constants in the target value "0", which will not provide much information for building the classification ML models.

The three categorical that are not constants. It is important to understand if exists some relevant association between these variables and the target classes. With this aim, it was built three normalized crosstabs of the categorical features "mode", "explicit", and "time_signature" and the target.

After this, it was performed the chi-square test of independence, which can be seen in figure 13, "mode" feature has a p-value of more than 19 % which does not reject the null hypothesis indicating independence between them. On other hand, "explicit" and "time_signature" have a p-value of 0,23 % and 0,05 %, respectively much lower than the standard 5 % of the test, so it indicates that the null hypothesis is rejected, and there is a significant association between the variables, what can indicate that both variables provide useful information for predicting the target.

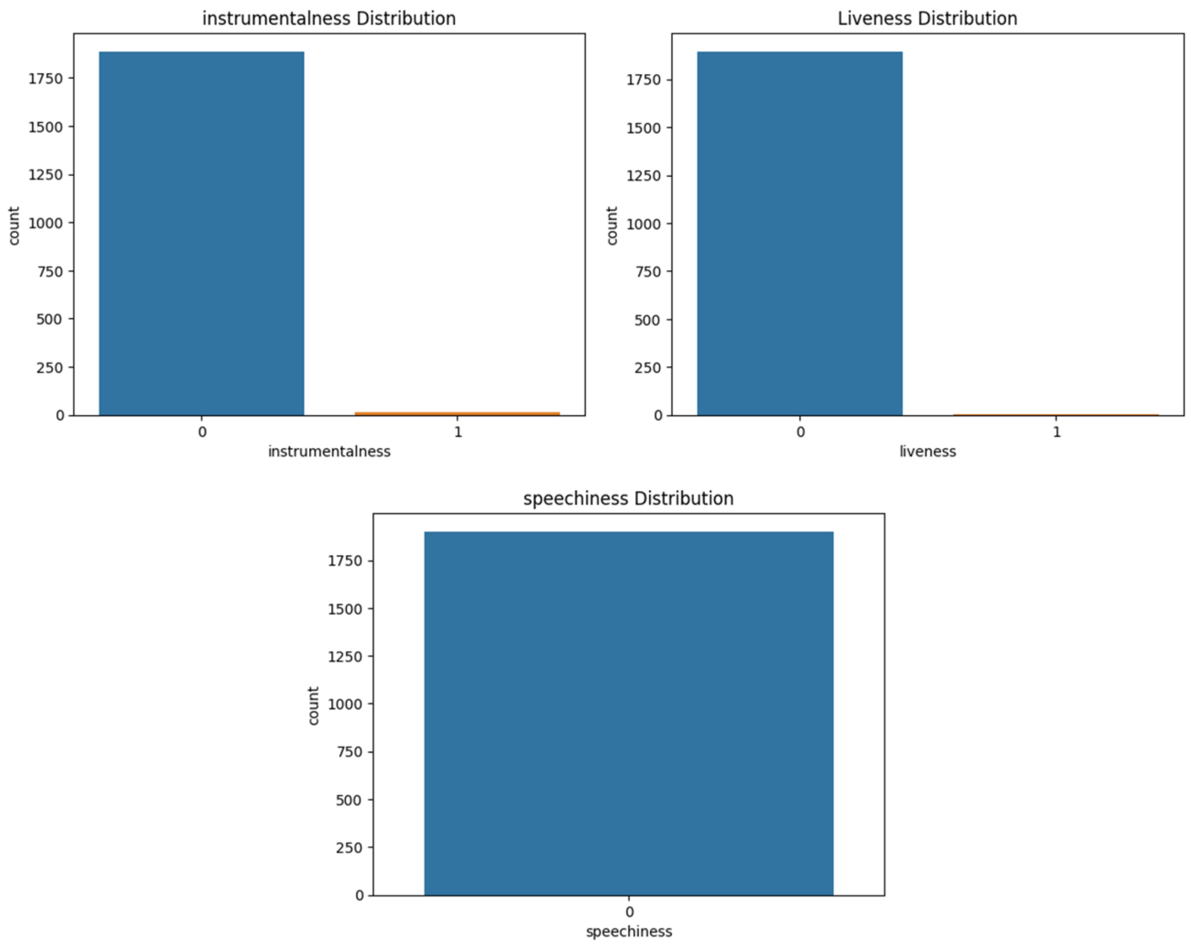
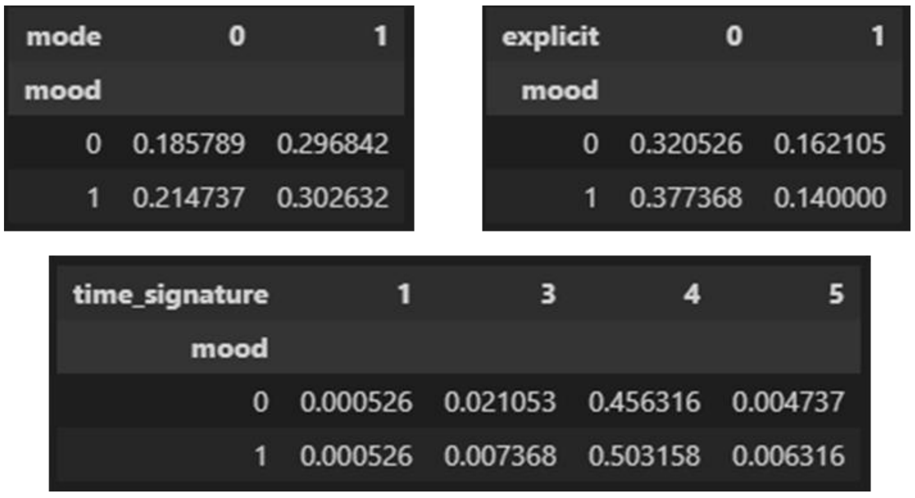


Figure 12. Distributions of categorical features.



mode	0	1
mood		
0	0.185789	0.296842
1	0.214737	0.302632

explicit	0	1
mood		
0	0.320526	0.162105
1	0.377368	0.140000

time_signature	1	3	4	5
mood				
0	0.000526	0.021053	0.456316	0.004737
1	0.000526	0.007368	0.503158	0.006316

Figure 13. Normalized contingency tables of the categorical features.

4.2 Data Preparation

Building upon the previous findings, this section explores the development of the data preparation Python script intending to generate preprocess data output.

The first step was to import both csv files generated from the data extraction script described in subsection 4.1.2. After that, the data types were improved to consume the minimum memory as possible. This is an important step to guarantee scalability of the model if the dataset increases over time.

In addition, with the discoveries from EDA phase, it was possible to understand that the genres and sub-genres columns need a treatment. The first transformation was replacing the missing values for zeros and using a label encoder to convert the names of the genres to numbers to make further calculations easier and avoid problems when building the ML models.

Another EDA finding was that the variables "speechiness", "liveness", "instrumentalness" can be considered as constants, so it is better to drop them from the dataset together with "valence" (it was converted to the target), "artist", "album", "track name", "release_date", "track_id" that are not relevant to the ML model, they are only for illustrative purposes.

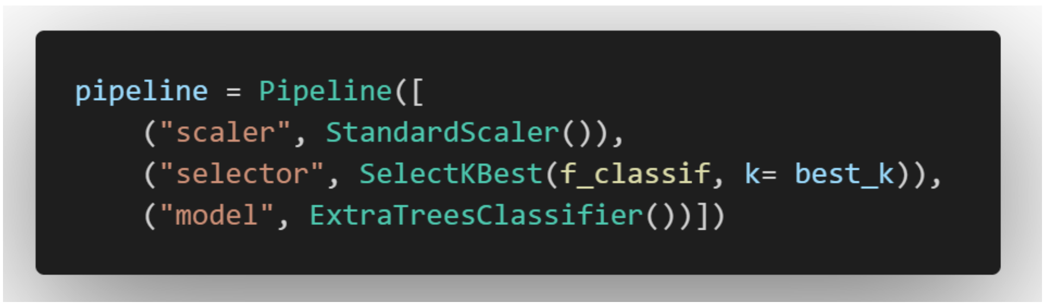
After the end of data cleaning, the last part consisted of exporting three csv files. One is composed of the 1800 tracks from the 2005 to 2022 playlist, and the second is the new music dataset from the top 100 tracks that is updated every week. They were separated to be used in the monitoring phase that will be better explained in the specific topic. The final file is concatenated from the two previous datasets that will serve as input for the next modelling phase.

4.3 Modelling

The objective of the modelling phase is to build a pipeline to train classification supervised models to predict the target variable "mood". To accomplish this goal was necessary to first import the complete preprocess data from the data preparation phase, split the data in training and test, next build a training pipeline, select the models which are going to experiment with, and save and register the model in a server repository to be used in the next phases.

The building of the training pipeline was considered one insight from the EDA, which discovered that the features have different scales and compared to other popular technique like Min Max scaler it is more suitable when features have non-uniform scales, so a standard scaler was used to avoid this issue. The first step performed was feature selection to understand the ideal number of features that can maximize the mean scores of the cross-validation five times, which tested the model in different samples.

After finding the ideal number of features, the train pipeline was composed of 1. standard scaler, 2. the ideal number of features and 3. the classification model. To improve even more, the model was necessary to do hyperparameter optimization, which was performed using random search cross-validation, which avoids excessive calculations and used the cross-validation mechanism to avoid overfitting.



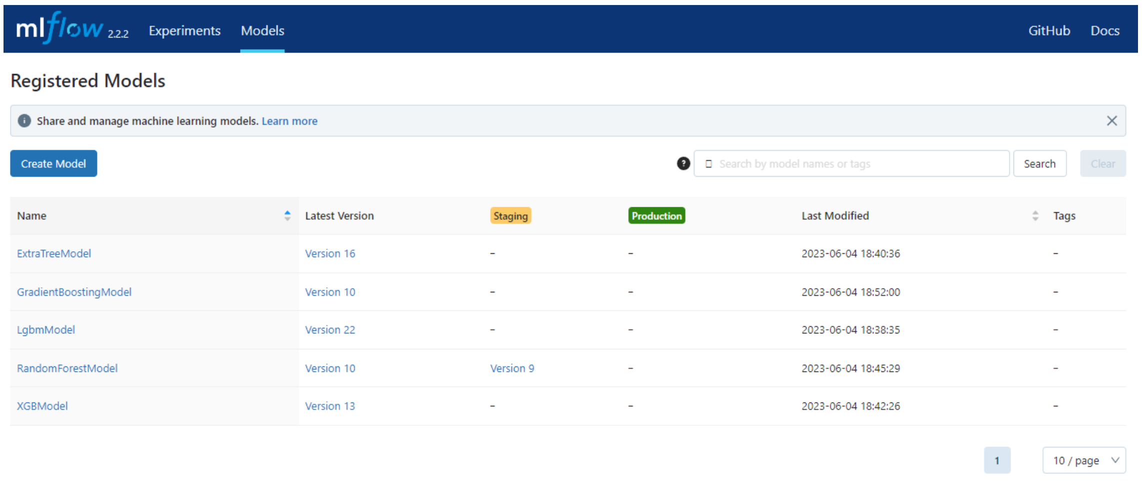
```
pipeline = Pipeline([
    ("scaler", StandardScaler()),
    ("selector", SelectKBest(f_classif, k= best_k)),
    ("model", ExtraTreesClassifier())])
```

Figure 14. Example of the train pipeline code from the Extra Trees model.

The final steps to train the model are fit and evaluate, which will be better explained in the next section. In addition, because this is an MLOps project, the MLflow tool was used to save the input features, evaluation metrics and hyperparameters in the MLflow linked to the DagsHub repository, which offers a free MLflow server. Finally, the model is saved and registered in the same server with all the dependencies to be reproducible in the deployment phase.

Figure 31 in appendix shows how MLflow store encapsulates a model with all packages required to train the model, the environment dependencies, and the model as a "Pickle" file.

One important aspect that needs to be discussed is the choice of models. It was performed in Jupiter notebook, several experiments using Python AutoML tools such as Pycaret and Tpot, indicating that the better-performing models are LightGBM, XGBoost, Random Forest, Extra Tree Classifier and Gradient Boosting. So, these five models were submitted to every step previously mentioned in this section.



Registered Models

Share and manage machine learning models. Learn more

Name	Latest Version	Staging	Production	Last Modified	Tags
ExtraTreeModel	Version 16	-	-	2023-06-04 18:40:36	-
GradientBoostingModel	Version 10	-	-	2023-06-04 18:52:00	-
LgbmModel	Version 22	-	-	2023-06-04 18:38:35	-
RandomForestModel	Version 10	Version 9	-	2023-06-04 18:45:29	-
XGBModel	Version 13	-	-	2023-06-04 18:42:26	-

Figure 15. Example of the MLflow UI server.

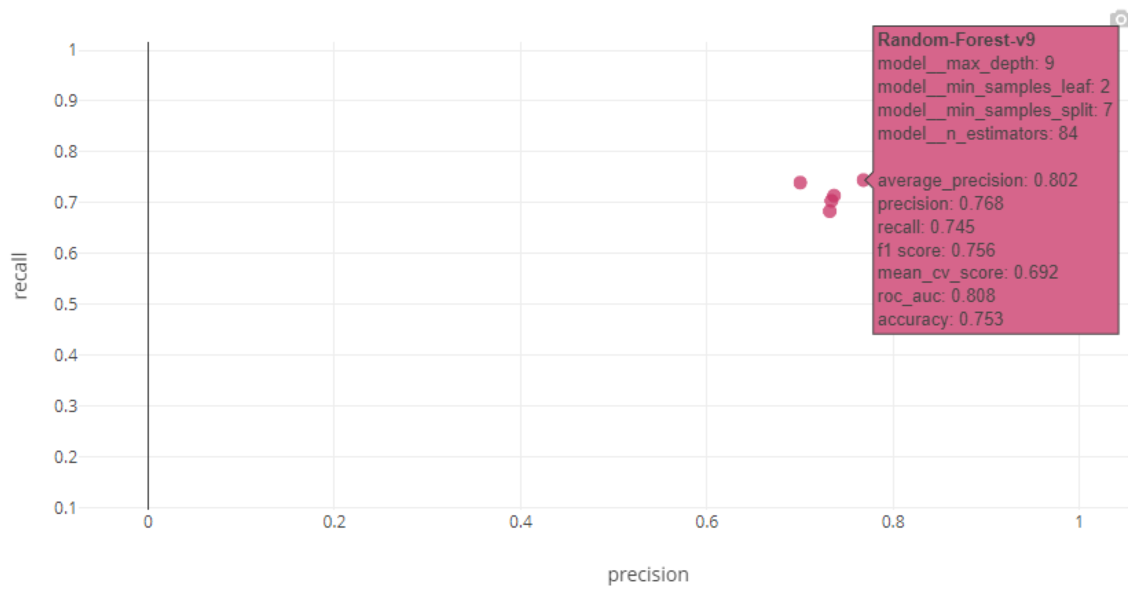
Figure 15 shows the MLflow UI server in the DagsHub, where it is possible to analyze the models trained in various experiment versions with their metrics and hyperparameter. After the evaluation phase, the registered model can be prioritized to advance to the staging and production stages to be deployed.

4.4 Model Evaluation

A function to return the main classification evaluation metrics was created in the model evaluation phase. It was calculated four metrics that used as input for the target test data and the target prediction: accuracy, precision, recall, f1 score. Additionally, the other two metrics were calculated that used target test data and the target prediction probabilities ROC AUC and average precision. The last metric was the mean of cross-validation score five times.

These metrics were incorporated into the model training script described in the last section, and the results were stored in the MLflow storage. To compare the five trained, it was possible to use the MLflow UI visuals available in figures 16 and 17.

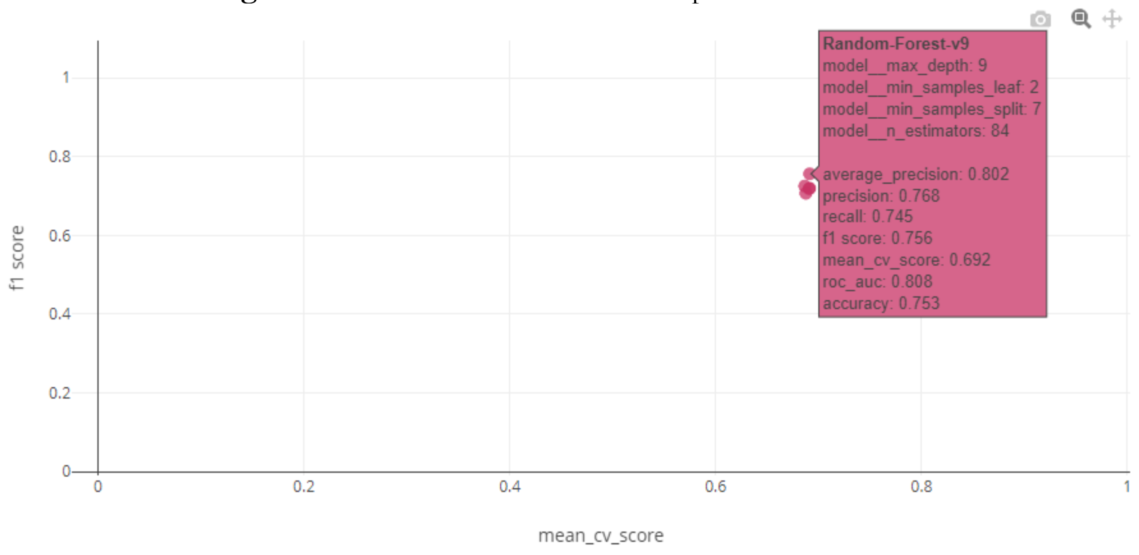
These figures show a scatter plot, comparing the five models according to precision and recall and the mean cross-validation score and f1 score. In both, it is possible to realize that the Random Forest model has the best results.



Random-Forest-v9
model__max_depth: 9
model__min_samples_leaf: 2
model__min_samples_split: 7
model__n_estimators: 84

average_precision: 0.802
precision: 0.768
recall: 0.745
f1 score: 0.756
mean_cv_score: 0.692
roc_auc: 0.808
accuracy: 0.753

Figure 16. Precision and recall scatter plot from MLflow.



Random-Forest-v9
model__max_depth: 9
model__min_samples_leaf: 2
model__min_samples_split: 7
model__n_estimators: 84

average_precision: 0.802
precision: 0.768
recall: 0.745
f1 score: 0.756
mean_cv_score: 0.692
roc_auc: 0.808
accuracy: 0.753

Figure 17. Mean CV and f1 scores scatter plot from MLflow.

The complete results of the evaluation metrics are available in table 4, in six out of seven metrics Random Forest achieves superior results, only losing the average precision for the Extra Tree Classifier. However, it is important to note that these results are only valid for these specific tracks dataset, and overtime, when the part of the dataset with new tracks will be updated, these metrics can change.

Metrics	Gradient Boosting	Random Forest	XGBoost	Extra Tree Classifier	LightGBM
accuracy	0.721	0.753	0.716	0.703	0.708
average precision	0.806	0.802	0.79	0.808	0.794
f1 score	0.725	0.756	0.719	0.72	0.707
mean cv score	0.687	0.692	0.691	0.692	0.688
precision	0.737	0.768	0.734	0.7	0.732
recall	0.714	0.745	0.704	0.74	0.684
ROC AUC	0.789	0.808	0.792	0.798	0.788

Table 4. Evaluation metrics extracted from MLflow.

In conclusion, the Random Forest with the hypermeters below is the best model for this experiment according to the metrics and will be deployed in the next section.

1. Max depth = 9,
2. Minimum samples leaf = 2,
3. Minimum samples split = 7,
4. Number of estimators = 84.

For the best Random Forest model, it was performed the model's explanation using Shap Python library, which contains the method Shapley Additive Explanations, as better discussed in subsection 2.2.5, explainability and responsible AI are fundamental aspects in any ML model to make it understandable and useful for a broader audience and not only be a black box that gives an output prediction that no one can trust.

In figure 18, the feature importance has been plotted using Shap, where it shows each track as a point distributed on each feature, the red indicates high values and blue low values of the features, and the positive x-axis indicates the contribution towards a happy song and the negative x axis in the direction of a sad song.

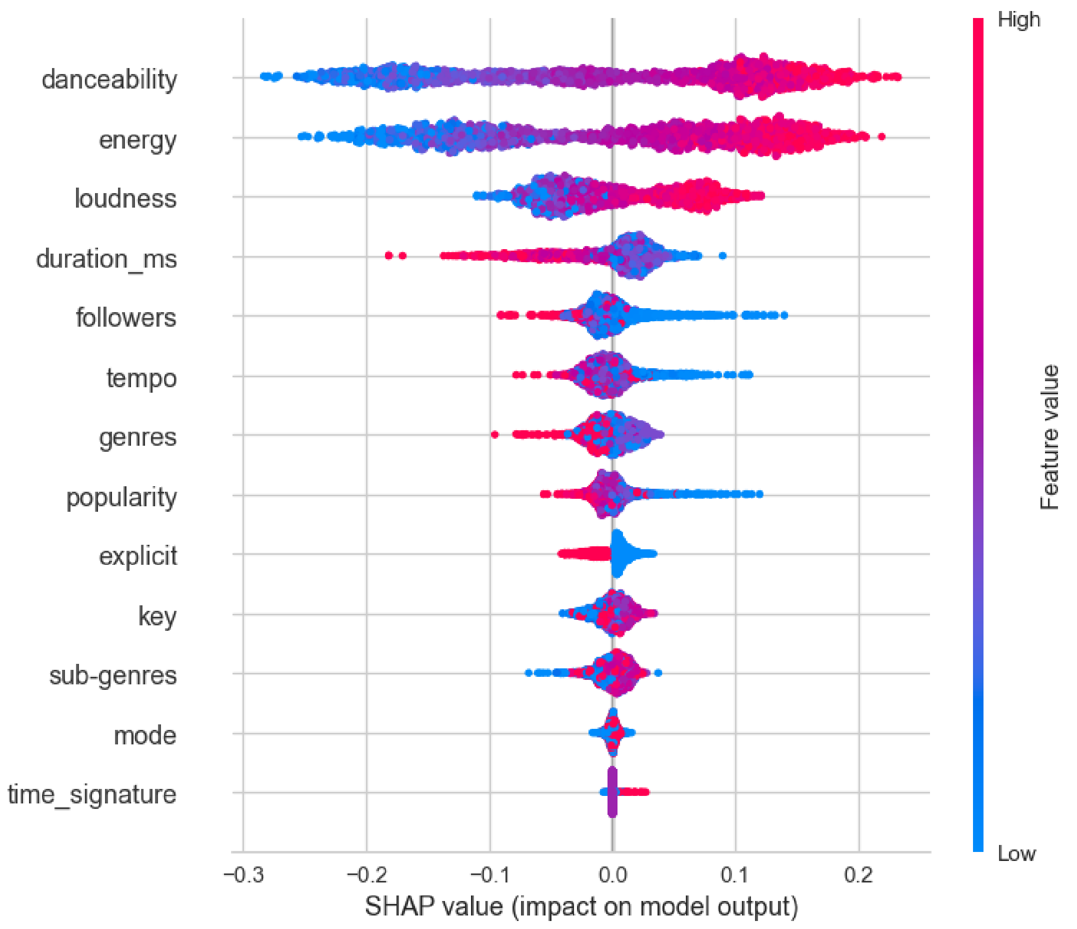


Figure 18. Shap feature importance summary plot.

Then, the features "danceability" and "energy" have the most significant impacts on predicting the target, and the higher values from both features are associated with happy songs and the lower values with sad songs. However, in the dependence plot in figure 19, between these two variables, even though they have similar impacts, many tracks with lower values of danceability have higher values of energy, indicating that they are not the same tracks with high energy and high danceability. In conclusion, both features are important to predict the target.

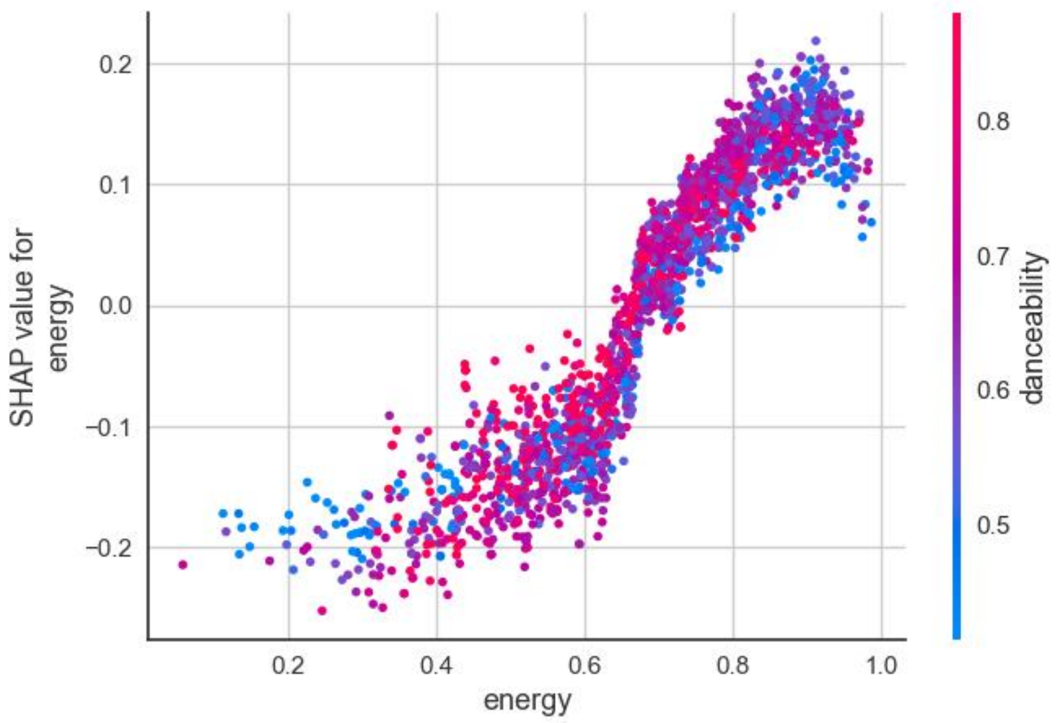


Figure 19. Shap dependency plot between energy and danceability.

The shap summary plot has two other features with interesting behaviors. The "duration_ms" has opposite impact in relation to the previous features. It shows that higher values of this feature have higher contributions to sad songs, and lower values tend to have lower contributions to happy songs.

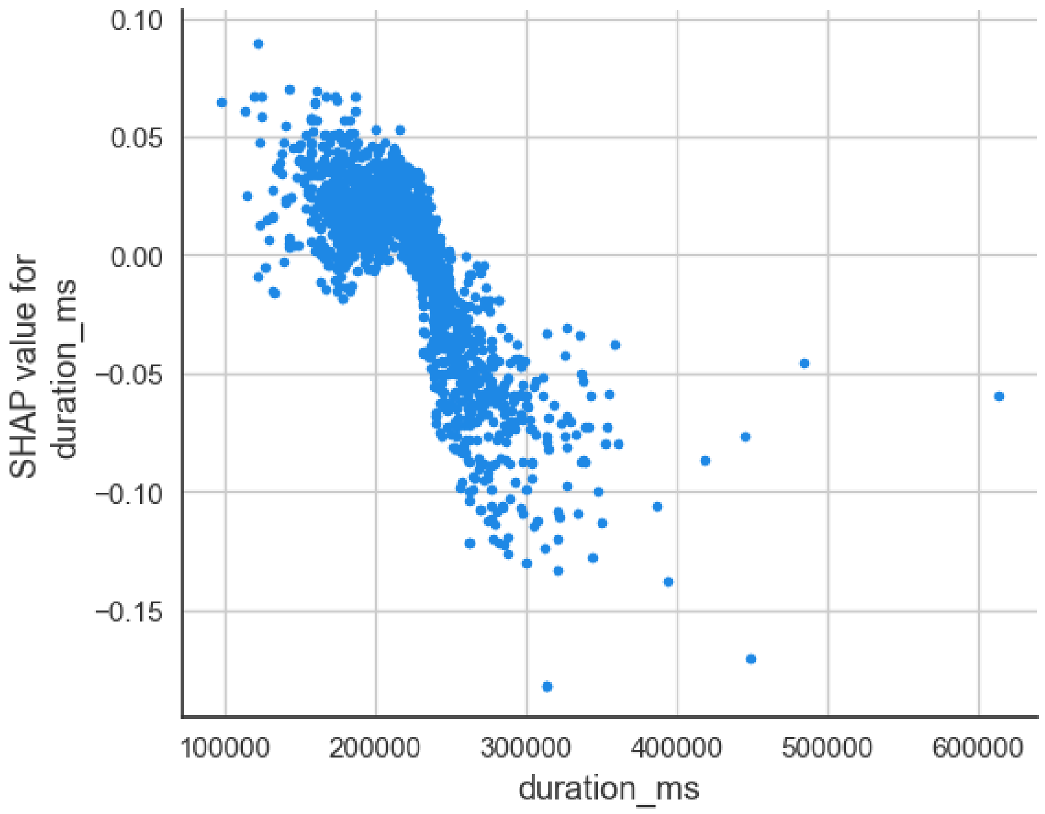


Figure 20. Shap scatter plot duration_ms.

The second feature is "popularity", which is the opposite of "duration_ms". It shows that higher values of this feature have lower contributions to sad songs, and lower values tend to have higher contributions to happy songs. The behaviors of these two features are clearer in the scatter plots where in the case of "duration_ms" in figure 20, the tracks with higher durations have greater negative impacts, and in "popularity" plot the tracks with lower popularity have greater positive impacts, it shown in figure 21.

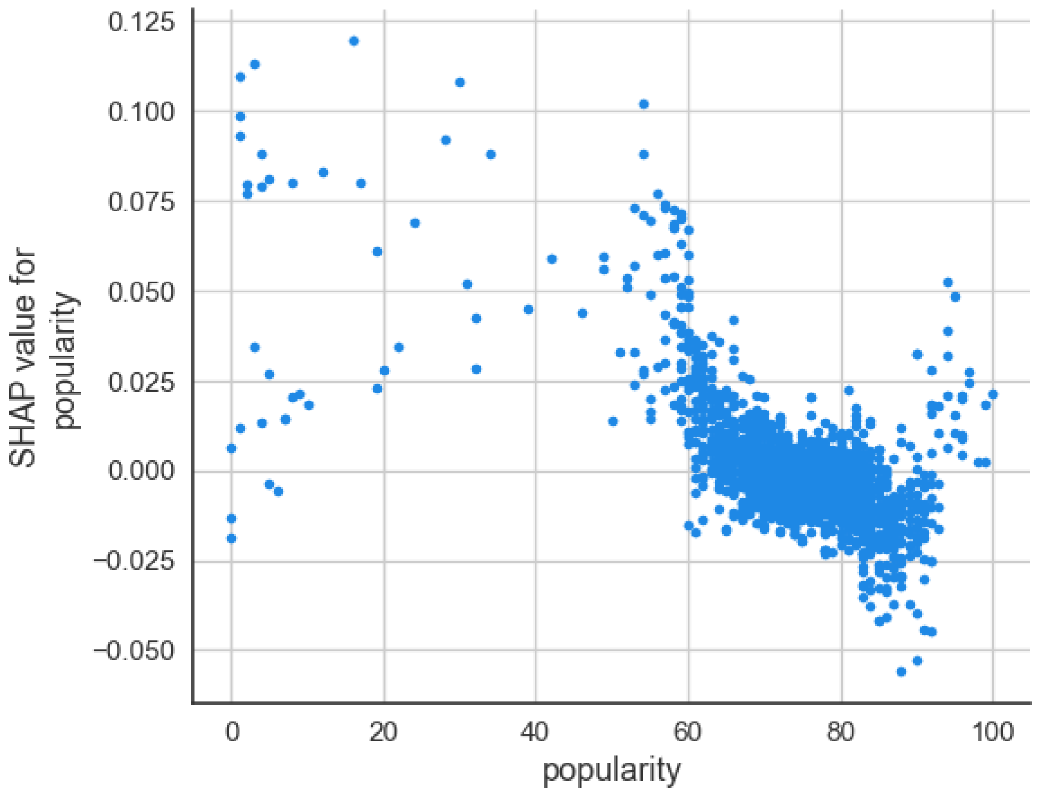


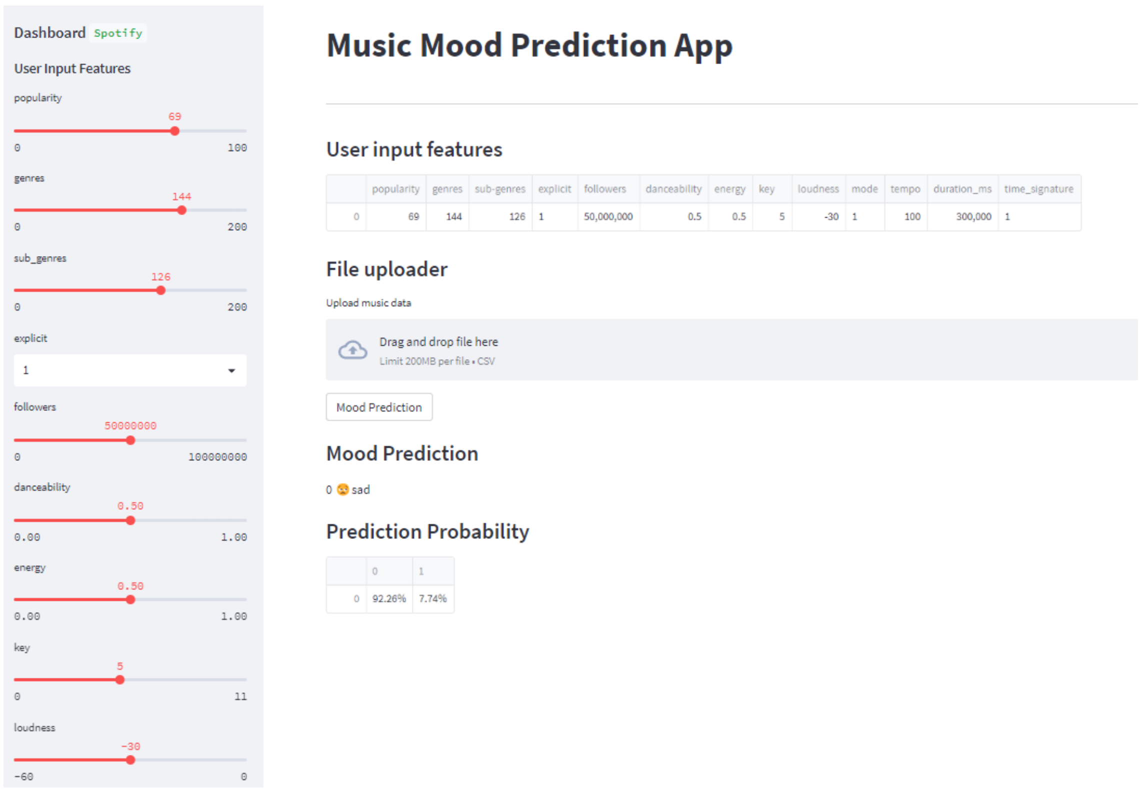
Figure 21. Shap scatter plot popularity.

4.5 Model Deployment

After evaluating the models and selecting the best one, it is time to deploy the model with its prediction power to an ML application using the Streamlit tool and public cloud to be used as a prediction service as mentioned in subsection 4.1.1.

To pursue this goal, the first step was building a prediction function to be called in the application script code. This function's input is the best model that is stored in MLflow server and the data in data frame or a csv file format. On the other hand, the outputs are going to be a list of predictions and one of the predictions probabilities to serve as the confidence of the predictions.

To continue this phase, building a Python script to run the Streamlit application was quintessential. It was called "Music Mood Prediction App" and got two main functionalities to make the predictions with the best model: 1. Manual input of the values of the features by the user; 2. Csv file uploader as an input of the features. The last step is to deploy the application to the Streamlit public cloud that was connected to the GitHub repository where the script is located.


Dashboard Spotify

User Input Features

popularity: 69 (0–100)

genres: 144 (0–200)

sub_genres: 126 (0–200)

explicit: 1

followers: 50000000 (0–100000000)

danceability: 0.50 (0.00–1.00)

energy: 0.50 (0.00–1.00)

key: 5 (0–11)

loudness: -30 (-60–0)

Music Mood Prediction App

User input features

	popularity	genres	sub-genres	explicit	followers	danceability	energy	key	loudness	mode	tempo	duration_ms	time_signature
0	69	144	126	1	50,000,000	0.5	0.5	5	-30	1	100	300,000	1

File uploader

Upload music data

Drag and drop file here
Limit 200MB per file • CSV

Mood Prediction

Mood Prediction

0 😢 sad

Prediction Probability

	0	1
0	92.26%	7.74%

Figure 22. Streamlit application: User input features example.

Figure 22 shows an example of the execution of manual user input functionality. In this case, with the features input of a song, that can be altered in the sidebar to test how the changes affect the prediction and probabilities in a straightforward manner. The best model predicts that a track with these feature values will be a sad song with a probability of 92,26 %.

Figure 32 in appendix illustrates the case of a Csv uploader that contains the features of ten tracks, which can be seen as a data frame. It works the same way as the first case but allows the upload of real songs from Spotify to test if they are happy or sad with the correspondent prediction probabilities.

4.6 Monitoring and Maintenance

4.6.1 Monitoring phase

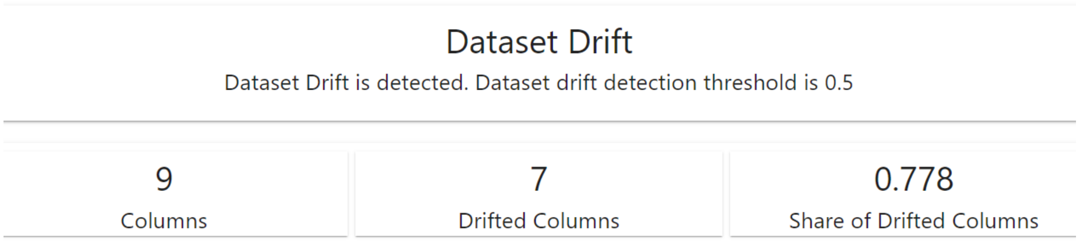
To monitor the data and the best classification model performance, five html reports were created using Evidently tool: 1. Data quality; 2. Data drift; 3. Target drift; 4. Test classification; 5. Classification performance.

The first step, as in the previous phases, was the creation of a Python script where it was received as inputs the preprocess data split into two files, one with the playlists from 2005 to 2022 and the other with the top 100 new tracks, both generated in the data preparation phase. In the code was necessary to map the features in numerical and categorical, specify the target variable and create a column with the predictions, using the predictions function from the deployment phase. Finally, it was possible to call Evidently library to create the reports having as standard the reference data as the dataset with old tracks and the current data as new tracks dataset.

The data quality report aims to provide a detailed statistical comparison between the two datasets and can also be used to improve the EDA analysis.

In figures 33 and 34 in appendix, both datasets are very similar, looking for the example of the danceability variable and the correlation between the numerical features.

The data drift and target reports are used to detect if the two datasets have significant changes concerning the input features, target, and prediction distributions. This can help to decide if the model needs to be retrained due to a degradation in the model performance over time. Evidently uses a standard algorithm of data drift for larger data, which are considered more than 1000 observations, for the calculation of the numerical features the Wasserstein distance and for the categorical Jensen-Shannon distance are used, both with confidence level of 0.95.


Dataset Drift

Dataset Drift is detected. Dataset drift detection threshold is 0.5

9	7	0.778
Columns	Drifted Columns	Share of Drifted Columns

Figure 23. Data Drift report summary.

Figure 23 shows that of the nine columns analyzed, seven had drifts. Only the target and the predictions do not have drift, so all the numerical variables presented in the datasets had some degree of drift. What may be distorted as the reference dataset has 1800 tracks and the current only 100. However, this behavior needs to be monitored over time to confirm that this is or is not a trend of the new data to retrain the model.

In figure 24, it can be observed the distribution of the variable "popularity", the one with the greatest drift. The clear trend is that new tracks have higher popularity than old ones.

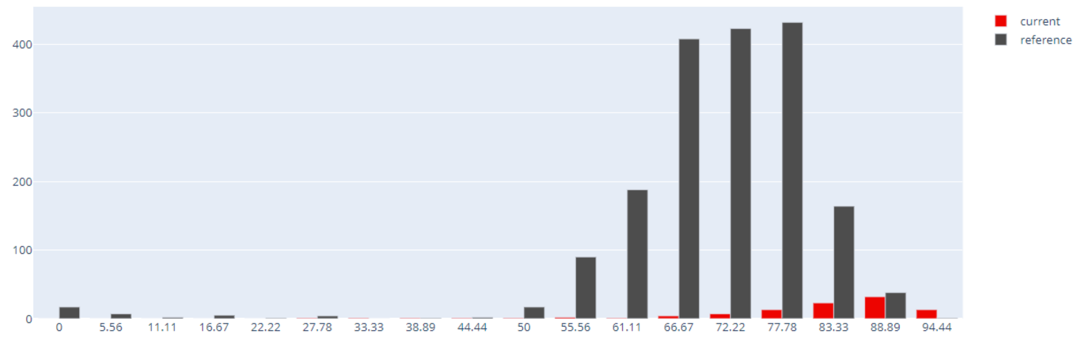
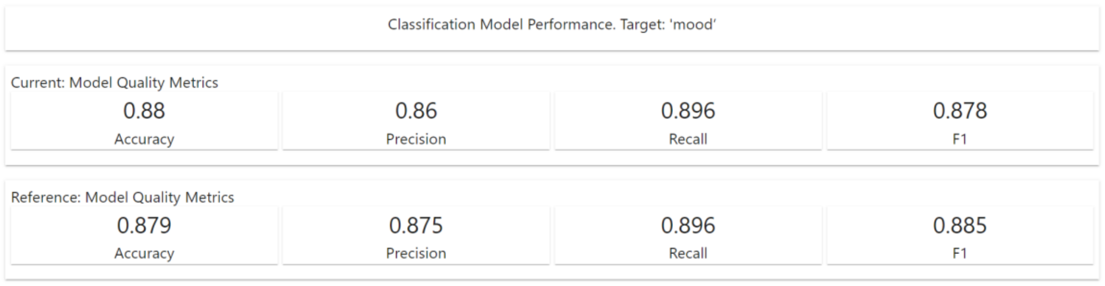


Figure 24. Data Drift report popularity distribution.

The other two reports test, measure and analyze the classification performance and should be used together to obtain the complete picture of the classification model.

Figure 35 in appendix shows the five tests presented in the test classification report that can serve as a summary of the classification report. The first test demonstrates no drift in the target variable using the threshold of 0,1. The following four tests measure the main quality metrics: precision, recall, f1 and accuracy score in both datasets to understand if they have significant differences, always using a threshold with confidence intervals. For this case, it was not received any warnings. However, it is important to analyze overtime with new tracks added to look up changes in behavior.


Classification Model Performance. Target: 'mood'

Current: Model Quality Metrics

0.88	0.86	0.896	0.878
Accuracy	Precision	Recall	F1

Reference: Model Quality Metrics

0.879	0.875	0.896	0.885
Accuracy	Precision	Recall	F1

Figure 25. Classification report: quality metrics.

The two figures 36 and 37 in appendix and the summary above were extracted from the classification performance report. After the evaluations provided from the test classification report, it is good to visualize in more detail what are the differences between the reference and current datasets. As expected, there are no great changes in the metrics between the datasets.

4.6.2 Maintenance phase

The purpose of the maintenance phase is to guarantee that the whole process is sustainable over time. To accomplish this goal, the pipeline must be automated with respect to the CI/CD principle that came from DevOps and the CT added from MLOps methodology.

To make the pipeline easier to manage, DVC functionality was used to create DAGs as mentioned in subsection 2.2.2. The construction of DAGs is done through a "Yaml" file where the name of the stages is specified, the command to execute the correspondent Python script built in the previous sections and dependencies and outputs.

Figure 26 is the illustration available in DagsHub, via integration when the DVC "Yaml" file is in the repository. It is shown that all code all managed by Git and all data are managed by DVC to maximize efficiency and ensure versioning. An important factor to note is the stages are extraction, preparation, training, and monitoring because the evaluation and deployment phases are imported in the same way as Python libraries when necessary, and the code of the Streamlit application only uses the deployed model.

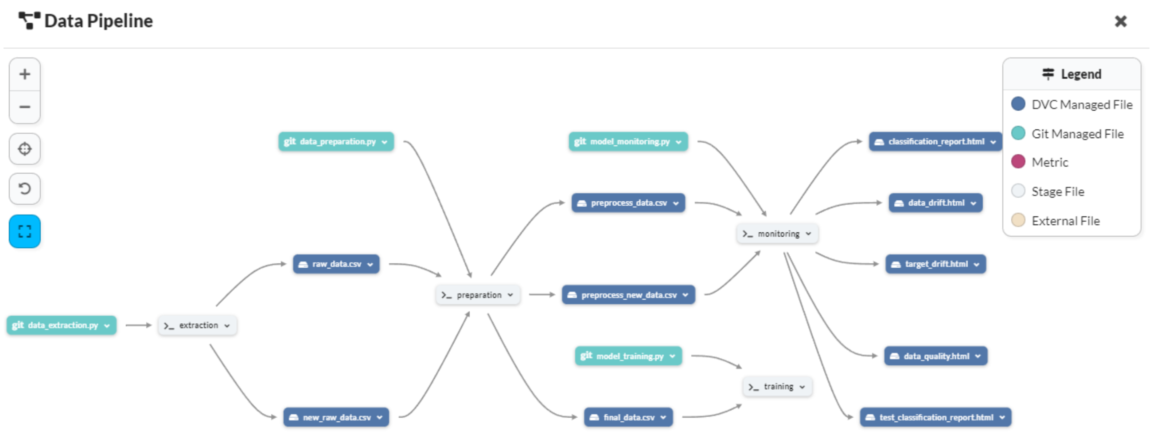
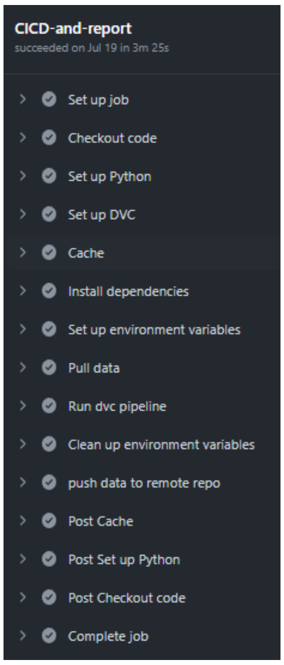


Figure 26. DVC pipeline illustration.

After building the DVC pipeline, it becomes possible to start it using only one command: DVC repro. Furthermore, this command only activates the stages that have changed. For example, if it is necessary to delete a feature in data preparation, the pipeline will not extract the data again, only from preparation forward, which ensures the minimum execution time in each cycle.



CICD-and-report
succeeded on Jul 19 in 3m 25s

- Set up job
- Checkout code
- Set up Python
- Set up DVC
- Cache
- Install dependencies
- Set up environment variables
- Pull data
- Run dvc pipeline
- Clean up environment variables
- push data to remote repo
- Post Cache
- Post Set up Python
- Post Checkout code
- Complete job

Figure 27. GitHub Actions CI/CD pipeline.

The last step to achieve the CI/CD pipeline is to automate using GitHub Actions to make it possible to execute the local pipeline in a remote environment. As shown in figure 27, it takes several steps configured in a "Yaml" file in GitHub. The first step is prior to the

"Yaml" file development, which is the registration of all local environment variables in GitHub secrets to be accessed by GitHub Actions.

Then, to start the "Yaml" file is necessary to establish a trigger to the pipeline. In this case was the execution of a push to GitHub but it could be a pull request or something else. Next, Ubuntu was selected as the operating system, and the variables stored in secrets were loaded. After this was performed, a checkout was to update the working directory, create a cache to store the memory of the pipeline steps and install Python, DVC and all the dependencies available in the "requirements.txt".

With these steps, the virtual environment is configured, and the pipeline data can be pulled from the DagsHub repository and the entire can be run with only the DVC repro command which makes this process as simple as possible. The last steps are to clean the environment variables and push the final data to DagsHub.

Finally, it is important to explain that to complete the continuous deployment part, the same push to GitHub activates the Streamlit public cloud connected to the repository, which automatically deploys the most updated code to the application.

4.7 Answering key research questions

In Chapter 4, all seven questions that were established at the end of Chapter 3 were answered. The first question was addressed throughout Chapter 4, which was guided by the CRISP-ML(Q) methodology, which allowed organizing the project and ensuring that each step was fulfilled.

The second issue was also addressed, as each open source tool was explained how and at what stage, as well as the reasons why they are used. The third was addressed mainly in subsection 4.1.1 and illustrated by figure 6. Next, sections 4.3 and 4.4 answered question 4, which showed how the best ML classification models for the problem were modeled and evaluated, as well as the comparison among them in table 4.

In section 4.5 and subsection 4.6.2, question 5 was explained, about what was the deployment of the ML application on Streamlit and how to maintain it using a CI/CD pipeline that was built on GitHub Actions. On the other hand, subsection 4.6.1 addressed question 6, answering how to properly monitor an ML model. Finally, the last question was answered in section 4.4, with the use of Shap it was possible to make the ML model algorithm more explainable to the audience.

5 Conclusions and future works

The present study successfully addressed the research objectives by developing an entire CI/CD pipeline using only open-source machine learning tools, which allowed the deployment of an ML application and monitoring of the production model to be retrained when necessary. To accomplish this, it was essential to follow the CRISP-ML(Q) steps and Google's maturity model framework to ensure that the MLOps project would be consistent with the best practices of the ML industry.

The CRISP-ML(Q) methodology was an important design guide, as in addition to providing the additional step of monitoring and maintenance over CRISP-DM, which is the quintessence for achieving MLOps standards, it maintains the traditional steps for achieving deployment of an ML Application. Moreover, Google's maturity model framework provided the architecture that helped choose the MLOps stack and the integration between the tools to achieve level 2: CI/CD automated pipeline.

The choice of the MLOps stack also fulfilled the requirements expressed in chapter 2. The tools Git and DVC ensured code and data versioning, respectively, of the entire pipeline and the repositories GitHub and DagsHub, together with the user interface of MLflow turn possible to have reproducible ML models packed with all necessary dependencies.

Another key requirement was the orchestration to achieve an automated pipeline. This was accomplished using DVC to build DAGs of each pipeline stage and then automated with GitHub Actions each time a push command is made from the local machine to the GitHub repository. In this specific project, it was unnecessary to use containers to store the dependencies. Just a text file was enough, but it could also have been used without problems.

The scalability requirement could be achieved by HTTP requests using Spotify web API. However, as one limitation of this dissertation due to only one CPU being available, it was downloading the data of 1900 tracks so as not to be a very time-consuming computation task on the data extraction and modelling phases. Also, Streamlit has a 1 GB limit to deploy as a public application, which means that scaling would not be an issue at this stage of development because the Python code file of the application is light. The heavy part was the stored data on DagsHub which has 10 GB of space and for the 1900 tracks it only used less than 1 MB of storage so there is plenty of headroom to scale.

The MLOps monitoring requirement was accomplished using Evidently reports, which allowed for identifying data quality, data and target drift and the model classification performance by comparing the reference data and the new data. This is to understand if the model metrics degraded over time and need to be retrained. As one of the limitations, the built trigger was manual, parsing the monitoring reports and then retraining the model and replacing the best model to deploy. However, it could have been done automatically, for example, setting a threshold of model accuracy and hooking it to the pipeline, but this development would not be trivial.

The last requirement concerned explainability and responsible AI. This step was performed in the evaluation stage using the best model: Random Forest. To complete this task the Shap library was used and discovered that the "danceability" and "energy" were the features with higher contributions to the target predictions.

Therefore, this thesis fulfilled the objectives proposed in the first chapter, executing the main steps of the CRISP-ML(Q) methodology and the MLOps requirements to obtain a consistent real world end-to-end MLOps project, in addition to providing a complete review of the MLOps tools with their functionalities and connections between them, an optimized MLOps stack was selected, and an automated CI/CD pipeline was delivered for continuous improvement of an ML application.

In future works, the possibility of increasing the number of tracks through the Spotify API and creating an automatic trigger has already been mentioned. Also, it would be interesting to test different MLOps stacks and compare efficiency, extract data through other APIs, and perform other ML tasks like regression, unsupervised learning, recommender systems or deep learning applications. Finally, to optimize the quality of the automated pipeline, tests can be created between major phases to ensure the quality of inputs and outputs.

References

Bhatt, U., Xiang, A., Sharma, S., Weller, A., Taly, A., Jia, Y., Ghosh, J., Puri, R., Moura, J. M. F., & Eckersley, P. (2020). Explainable machine learning in deployment. *FAT* 2020 - Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*, 648–657. https://doi.org/10.1145/3351095.3375624

Garg, S., Pundir, P., Rathee, G., Gupta, P. K., Garg, S., & Ahlawat, S. (2021). On Continuous Integration / Continuous Delivery for Automated Deployment of Machine Learning Models using MLOps. *Proceedings - 2021 IEEE 4th International Conference on Artificial Intelligence and Knowledge Engineering, AIKE 2021*, 25–28. https://doi.org/10.1109/AIKE52691.2021.00010

Gift, Noah., & Deza, Alfredo. (2021). *Practical MLOps: Operationalizing Machine Learning Models* (First).

Google. (2023). MLOps: Continuous delivery and automation pipelines in machine learning. https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning

Hewage, N., & Meedeniya, D. (2022). *Machine Learning Operations: A Survey on MLOps Tool Support*. https://doi.org/10.48550/arXiv.2202.10169

John, M. M., Olsson, H. H., & Bosch, J. (2021). Towards MLOps: A Framework and Maturity Model. *Proceedings - 2021 47th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2021*, 334–341. https://doi.org/10.1109/SEAA53835.2021.00050

Klaise, J., van Looveren, A., Cox, C., Vacanti, G., & Coca, A. (2020). *Monitoring and explainability of models in production*. http://arxiv.org/abs/2007.06299

Kreuzberger, D., Kühl, N., & Hirschl, S. (2022). *Machine Learning Operations (MLOps): Overview, Definition, and Architecture*.

Lakshmanan, V., Robinson, S., & Munn, M. (2020). *Machine Learning Design Patterns Solutions to Common Challenges in Data Preparation, Model Building, and MLOps* (R. Novack, C. Collins, B. Kelly, C. Roumeliotis, & H. Bauer Forsyth, Eds.; First Edition). O'Reilly Media, Inc.

Liu, Y., Ling, Z., Huo, B., Wang, B., Chen, T., & Mouine, E. (2020). Building A Platform for Machine Learning Operations from Open Source Frameworks. *IFAC-PapersOnLine*, *53*(5), 704–709. https://doi.org/10.1016/j.ifacol.2021.04.161

Mäkinen, S. (2021). *Designing an open-source cloud-native MLOps pipeline* [Master, University of Helsinki]. http://www.cs.helsinki.fi/

Microsoft (2023). Machine Learning operations maturity model. https://learn.microsoft.com/en-us/azure/architecture/example-scenario/mlops/mlops-maturity-model

Praveen Gujjar, J., & Kumar, V. N. (2022). Demystifying MLOps for continuous delivery of the product. *Asian Journal of Advances in Research*, *13*(3), 19–23.

Ruf, P., Madan, M., Reich, C., & Ould-Abdeslam, D. (2021). Demystifying mlops and presenting a recipe for the selection of open-source tools. *Applied Sciences (Switzerland)*, *11*(19). https://doi.org/10.3390/app11198861

Salvucci, E. (2021). *MLOps-Standardizing the Machine Learning Workflow Thesis on Big Data*. Università di Bologna.

Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., Chaudhary, V., Young, M., Crespo, J.-F., & Dennison, D. (2015). *Hidden Technical Debt in Machine Learning Systems*.

Silva, L. (2021). *Architectural redesign and evaluation of an open source MLOps platform: a case study of Apache Marvin-AI* [Master]. Universidade Federal de São Carlos.

Spotify (2023). Web API | Spotify for Developers. https://developer.spotify.com/documentation/web-api/

Studer, S., Bui, T. B., Drescher, C., Hanuschkin, A., Winkler, L., Peters, S., & Mueller, K.-R. (2020). *Towards CRISP-ML(Q): A Machine Learning Process Model with Quality Assurance Methodology*. http://arxiv.org/abs/2003.05155

Symeonidis, G., Nerantzis, E., Kazakis, A., & Papakostas, G. A. (2022). MLOps - Definitions, Tools and Challenges. *2022 IEEE 12th Annual Computing and Communication Workshop and Conference, CCWC 2022*, 453–460. https://doi.org/10.1109/CCWC54503.2022.9720902

Treveil, M., Omont, N., Stenac, C., Lefèvre, K., Phan, D., Zentici, J., Lavaoillotte, A., Miyazaki, M., & Heidmann, L. (2020). *Introducing MLOps: How to Scale Machine Learning in the Enterprise* (First Edition). O'Reilly Media, Inc.

Appendixes

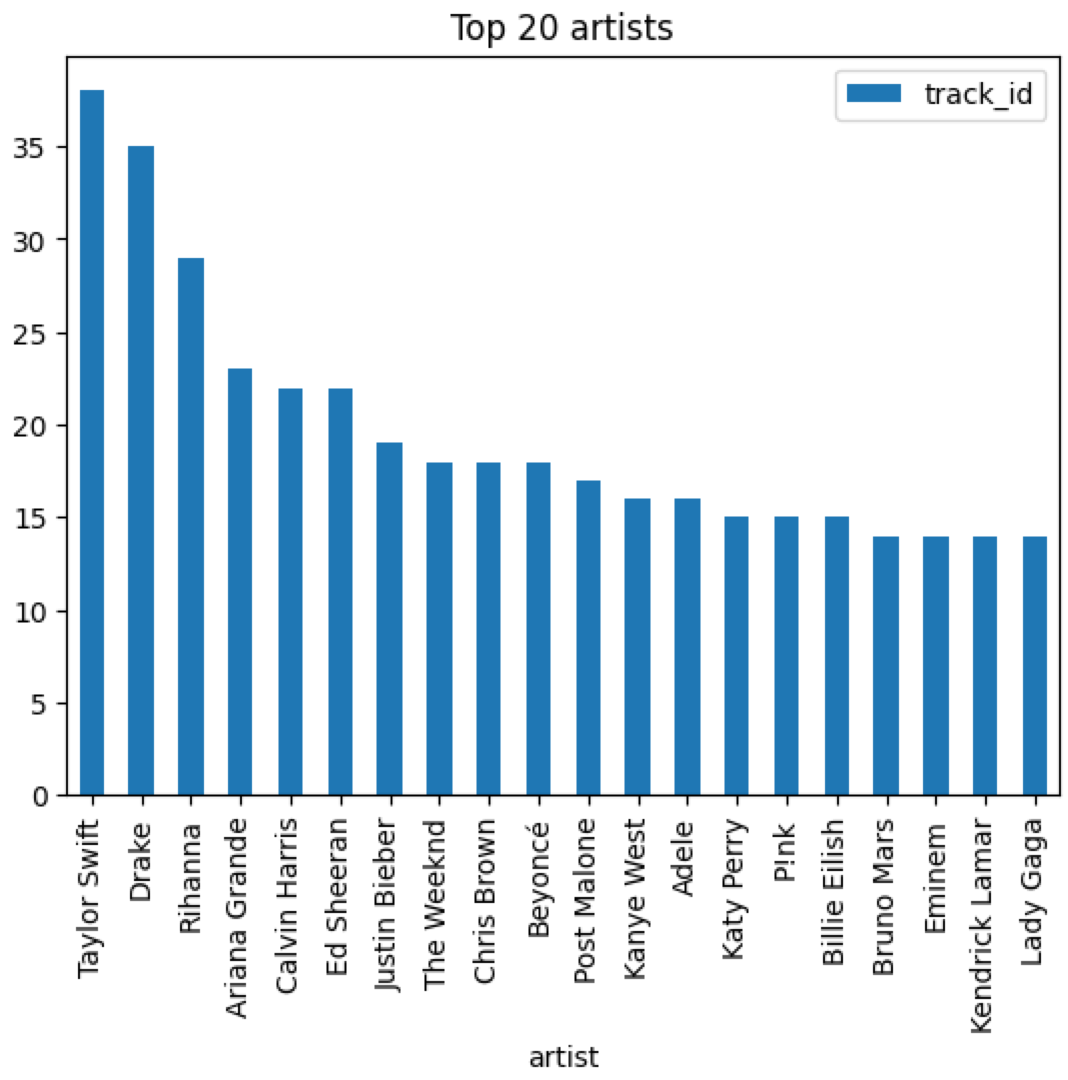


Figure 28. EDA: Top 20 artists.

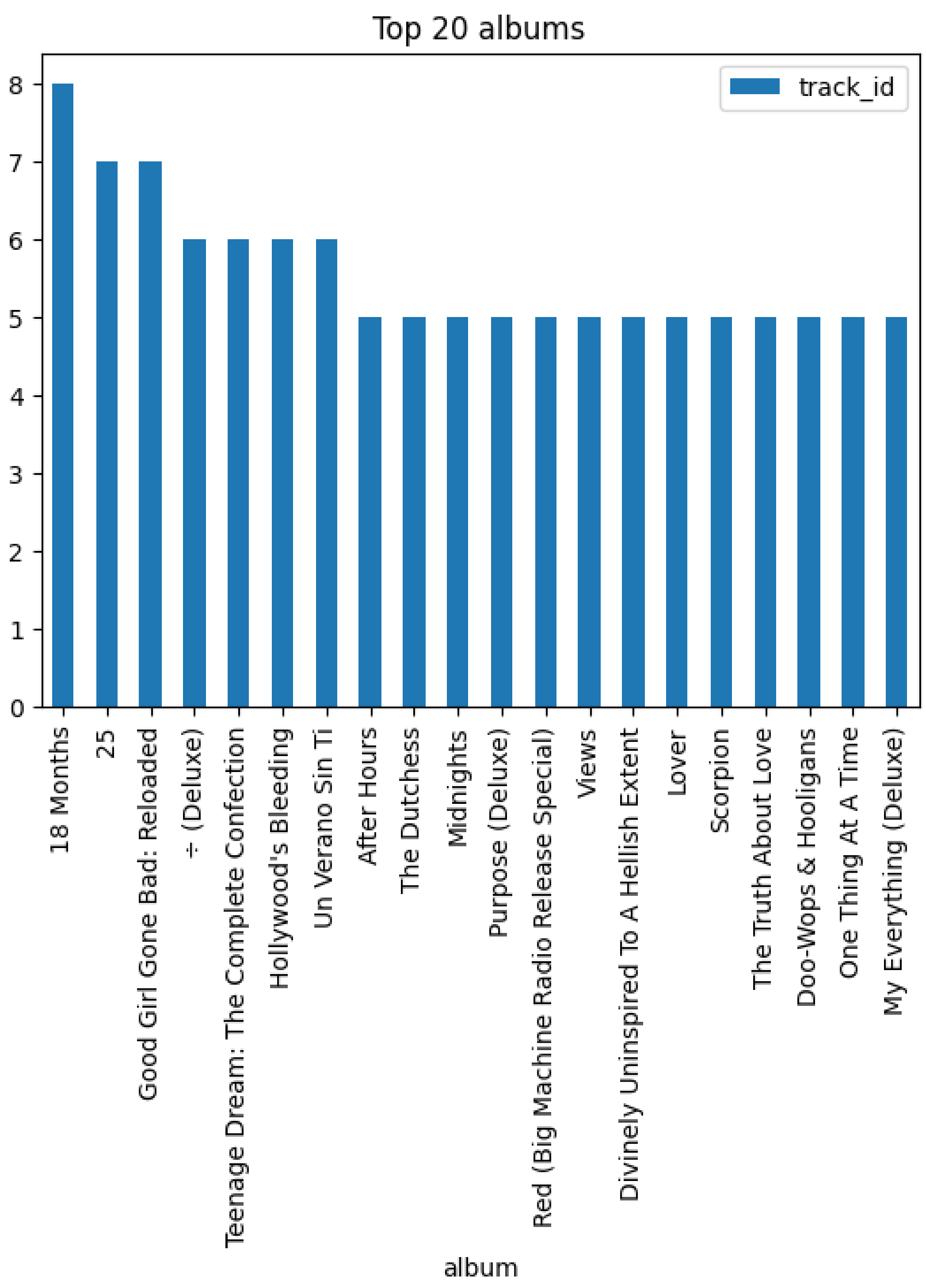


Figure 29. EDA: Top 20 albums.

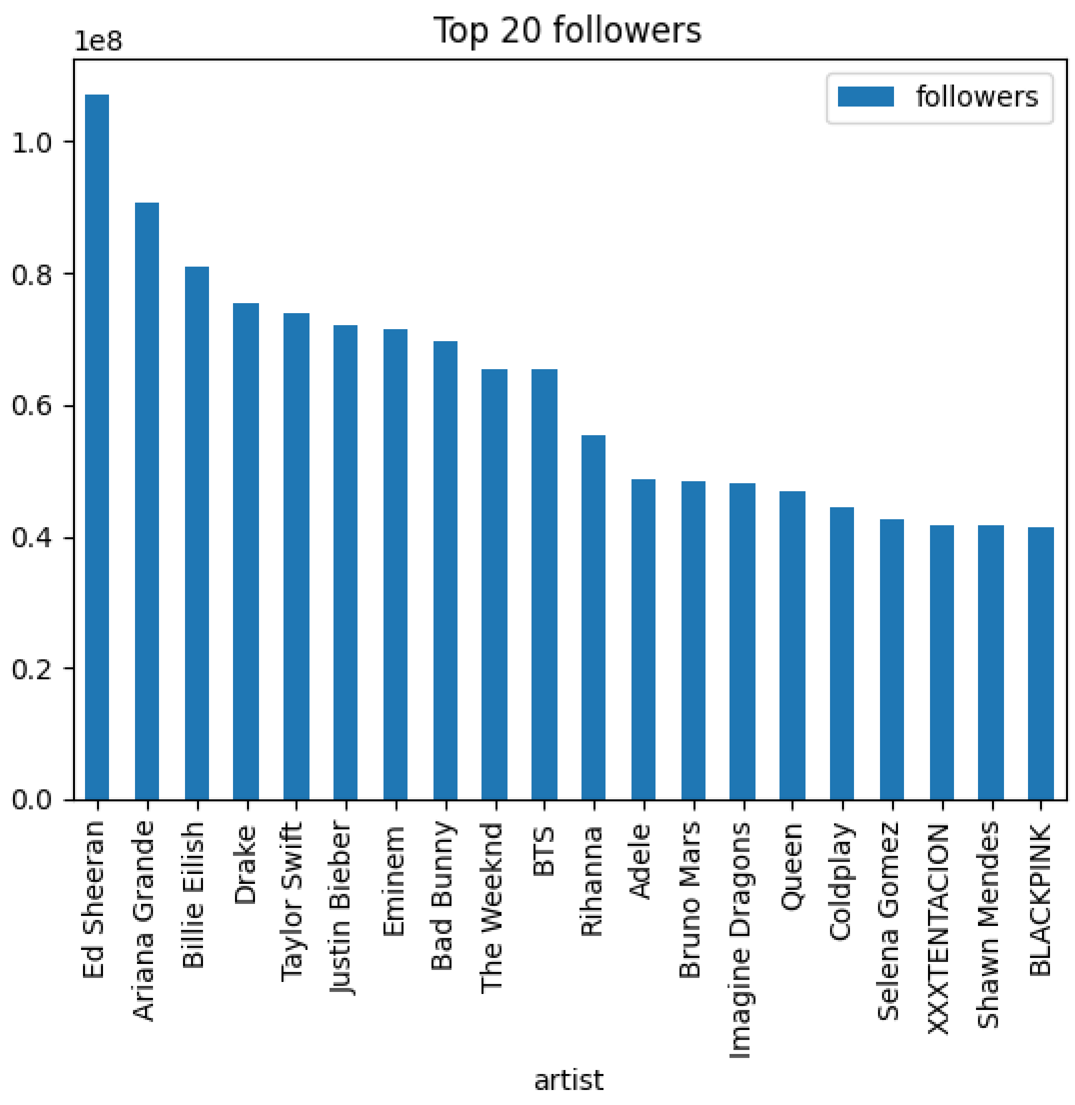
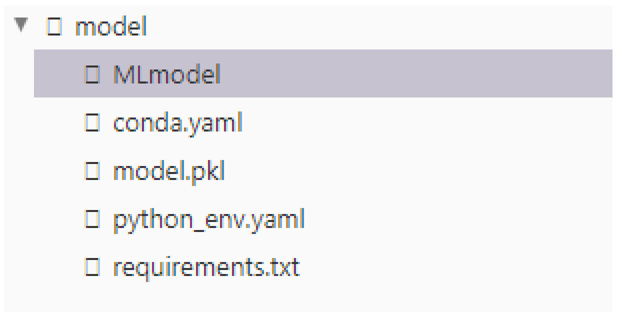
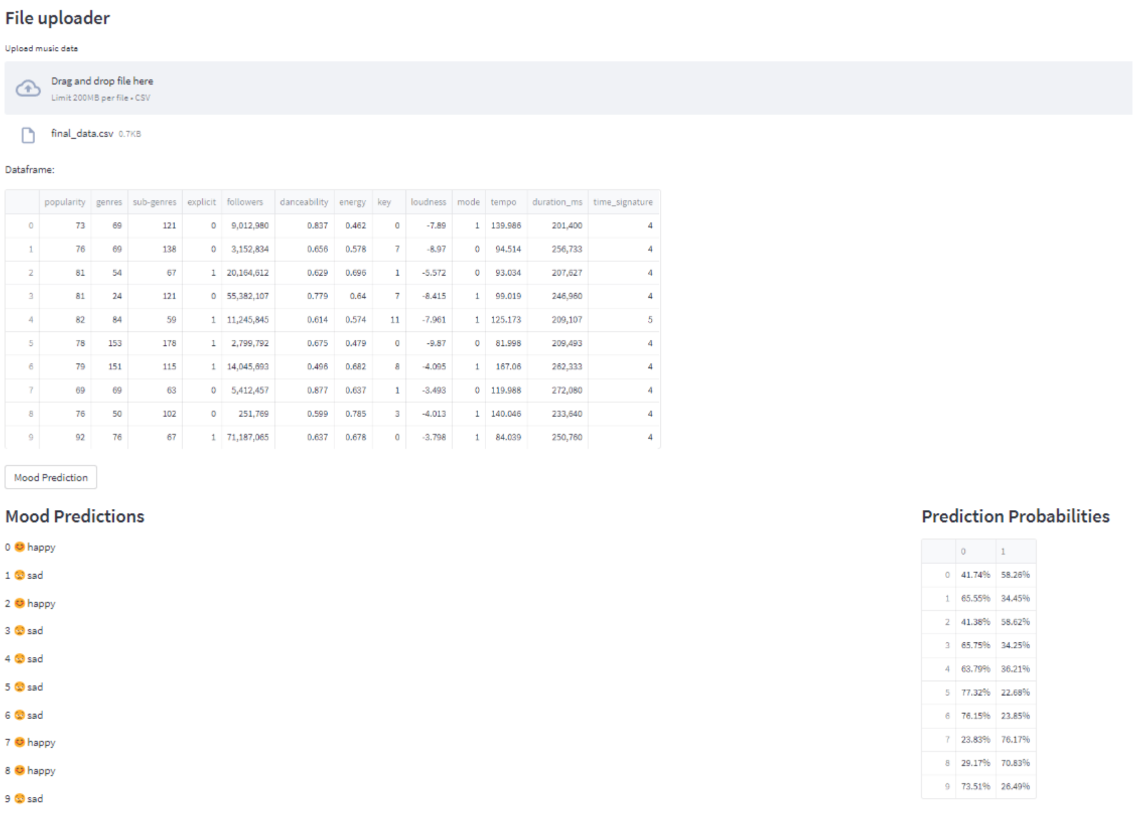


Figure 30. EDA: Top 20 followers.



```
▼ model
    MLmodel
    conda.yaml
    model.pkl
    python_env.yaml
    requirements.txt
```

Figure 31. Example of a model stored in the MLflow server.



File uploader

Upload music data

Drag and drop file here
Limit 200MB per file • CSV

final_data.csv 0.7KB

Dataframe:

	popularity	genres	sub-genres	explicit	followers	danceability	energy	key	loudness	mode	tempo	duration_ms	time_signature
0	73	69	121	0	9,012,980	0.837	0.462	0	-7.89	1	139.986	201,400	4
1	76	69	138	0	3,152,834	0.656	0.578	7	-8.97	0	94.514	256,733	4
2	81	54	67	1	20,164,612	0.629	0.696	1	-5.572	0	93.034	207,627	4
3	81	24	121	0	55,382,107	0.779	0.64	7	-8.415	1	99.019	246,960	4
4	82	84	59	1	11,245,845	0.614	0.574	11	-7.961	1	125.173	209,107	5
5	78	153	178	1	2,799,792	0.675	0.479	0	-9.87	0	81.998	209,493	4
6	79	151	115	1	14,045,693	0.496	0.682	8	-4.095	1	167.06	262,333	4
7	69	69	63	0	5,412,457	0.877	0.637	1	-3.493	0	119.988	272,080	4
8	76	50	102	0	251,769	0.599	0.785	3	-4.013	1	140.046	233,640	4
9	92	76	67	1	71,187,065	0.637	0.678	0	-3.798	1	84.039	250,760	4

Mood Prediction

Mood Predictions

0 😊 happy

1 😞 sad

2 😊 happy

3 😞 sad

4 😞 sad

5 😞 sad

6 😞 sad

7 😊 happy

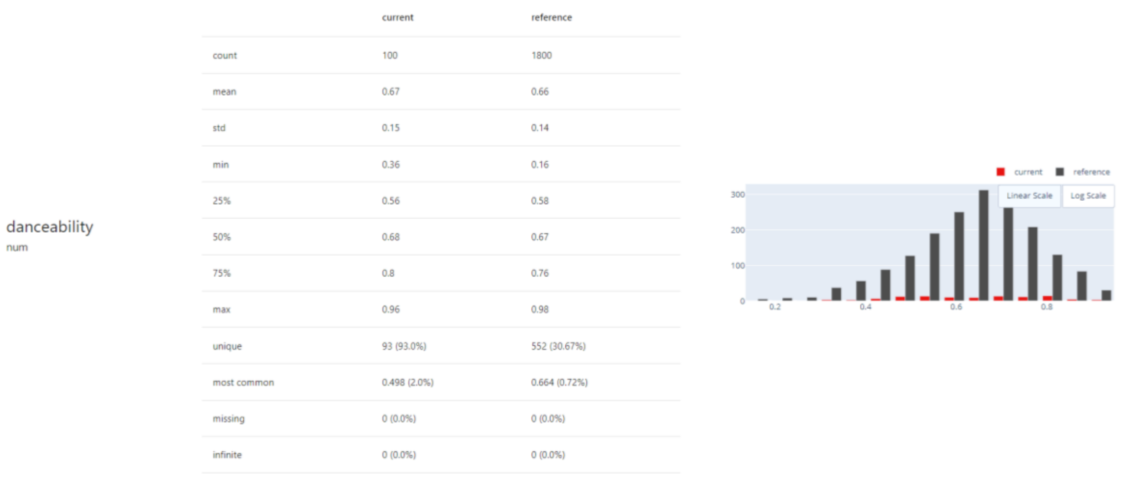
8 😊 happy

9 😞 sad

Prediction Probabilities

	0	1
0	41.74%	58.26%
1	65.55%	34.45%
2	41.38%	58.62%
3	65.75%	34.25%
4	63.79%	36.21%
5	77.32%	22.68%
6	76.15%	23.85%
7	23.83%	76.17%
8	29.17%	70.83%
9	73.51%	26.49%

Figure 32. Streamlit application: Csv uploader example.



danceability
num

	current	reference
count	100	1800
mean	0.67	0.66
std	0.15	0.14
min	0.36	0.16
25%	0.56	0.58
50%	0.68	0.67
75%	0.8	0.76
max	0.96	0.98
unique	93 (93.0%)	552 (30.67%)
most common	0.498 (2.0%)	0.664 (0.72%)
missing	0 (0.0%)	0 (0.0%)
infinite	0 (0.0%)	0 (0.0%)

Figure 33. Danceability variable statistical description example.

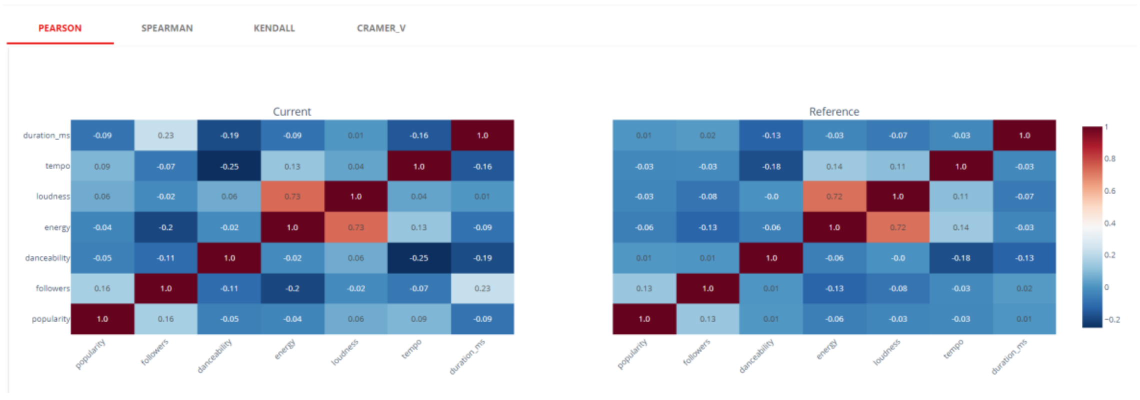
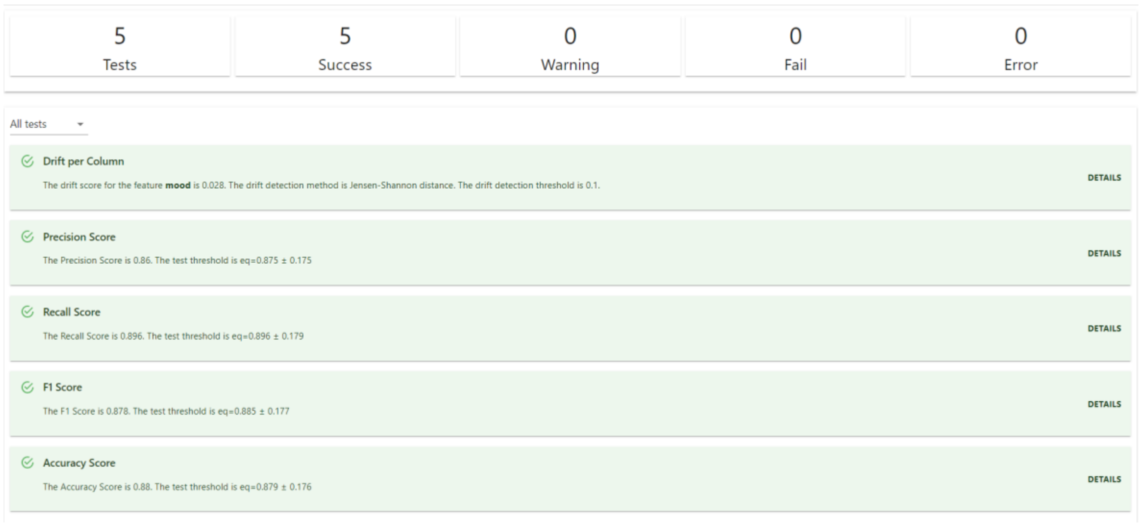


Figure 34. Pearson correlation matrix example.



5	5	0	0	0
Tests	Success	Warning	Fail	Error

All tests

✓ Drift per Column
The drift score for the feature **mood** is 0.028. The drift detection method is Jensen-Shannon distance. The drift detection threshold is 0.1.
DETAILS

✓ Precision Score
The Precision Score is 0.86. The test threshold is eq=0.875 ± 0.175
DETAILS

✓ Recall Score
The Recall Score is 0.896. The test threshold is eq=0.896 ± 0.179
DETAILS

✓ F1 Score
The F1 Score is 0.878. The test threshold is eq=0.885 ± 0.177
DETAILS

✓ Accuracy Score
The Accuracy Score is 0.88. The test threshold is eq=0.879 ± 0.176
DETAILS

Figure 35. Test classification report.

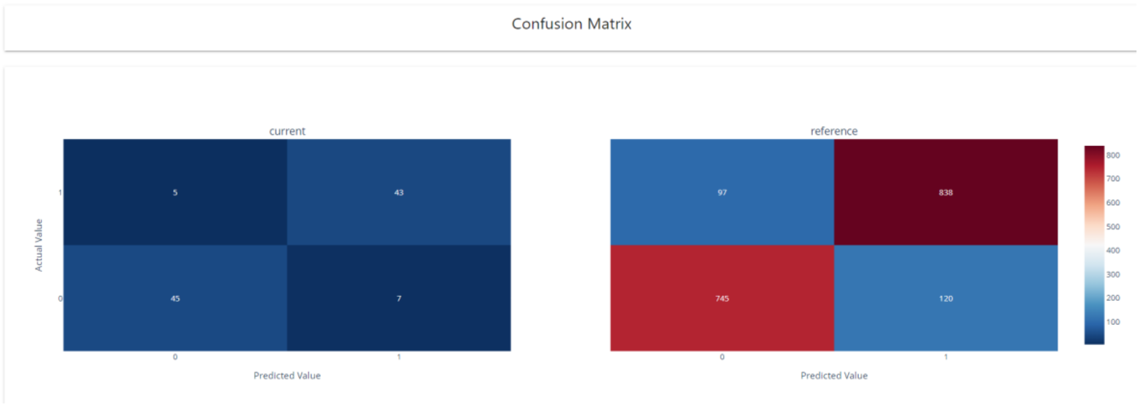


Figure 36. Classification report: confusion matrix.

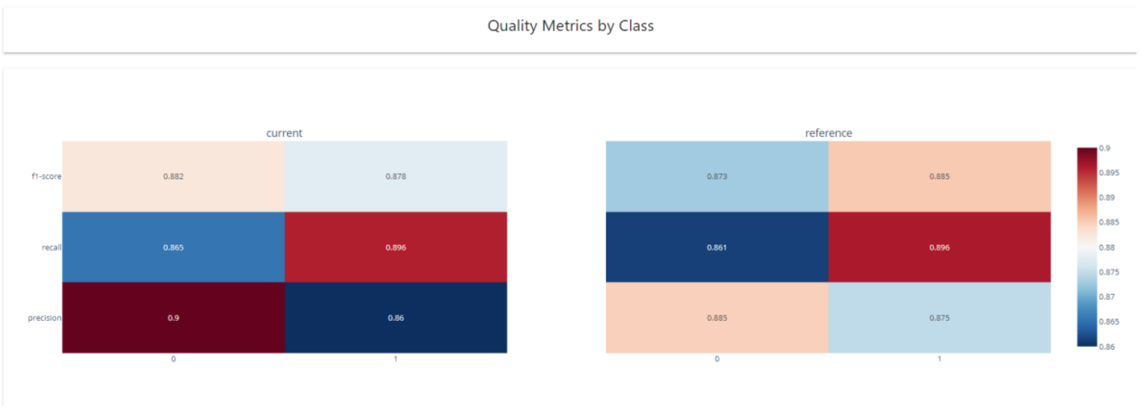


Figure 37. Classification report: quality metrics by class.