MASTER

Automatic data cleaning

Zhang, J.

Award date:
2018

Technische Universiteit
Eindhoven
University of Technology

Department of Mathematics and Computer Science
Data Mining Research Group

Automatic Data Cleaning

Master Thesis

Ji Zhang

Supervisor:
Dr. ir. Joaquin Vanschoren

Assessment Committee Members:
Dr. ir. Joaquin Vanschoren
Dr. ir. Nikolay Yakovets
Dr. ir. Michel Westenberg

Eindhoven, November 2018

Abstract

In the machine learning field, data quality is essential for developing an effective machine learning model. However, raw data always contain various kinds of data problems such as duplicated records, missing values and outliers, which may weaken the model power severely. Therefore, it is vital to clean data thoroughly before proceeding with the data analysis step. The process that cleans the potential problems in the data is called data cleaning. Unfortunately, although inevitable and primary as data cleaning is, it is also quite a tedious and time-consuming task. People do not want to repeat this process endlessly and hence expect a tool to help them clean data automatically.

In this thesis, a Python tool is developed in order to fulfill this expectation. This tool is able to identify the potential issues in the data and report results and recommendations such that users can clean data smoothly and effectively with its assistance. Compared with existing data cleaning tools, this tool is specially designed for addressing machine learning tasks and can find the optimal cleaning approach according to the characteristics of the given dataset. There are three aspects meaningfully automated in this thesis: automatic discovery of data types, automatic missing value handling, and automatic outlier detection.

Preface

First and foremost, I would like to thank my supervisor Dr. Joaquin Vanschoren for his dedication and guidance throughout this master thesis work. With his encouragement, I can find my passion for data science and have the freedom to pursue this thesis. Special thanks to Dr. Michel Westenberg and Dr. Nikolay Yakovets for their valuable suggestions on this thesis and assessing my project. Many thanks to my great colleague students for their kindness to offer me help at any time. It was really a joy working alongside them. In addition, I would like to thank my wonderful friends for listening, offering me advice, and especially for always being there for me. Finally, I would like to express my gratitude to my parents, Jianju and Jianxin, who have constantly inspired me to study as much as possible, and given me the opportunity to do so. Without all these people, I would not be here.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Data in the real world are acquired from a variety of sources. The raw data are usually inconsistent, inaccurate and incomplete, which we call dirty data. The analytical results from dirty data are not dependable since high-quality decisions are normally based on high-quality data. Consequently, the raw data cannot be used directly for performing analytical procedures and need to be cleaned beforehand. Data cleaning detects and removes the inconsistent, inaccurate and incomplete parts from data to improve the quality of data. This process prepares data for future analysis and is usually inevitable and essential.

In this thesis, we are concerned with the automation of data cleaning. A simple Python tool is developed to offer automated, data-driven support to help users clean data easily. In this chapter, we introduce the basic background and provide an overview of the whole project. In Section 1.1, we explain why the automation of data cleaning is desirable. Section 1.2 presents the main objective of the thesis. The results of this thesis are summarized in Section 1.3. The further outline of this thesis is described in Section 1.4.

1.1 Motivation

Data cleaning is a primary task of data science. Potential problems such as missing values and outliers in the raw dataset will bias the results of data analysis and need to be dealt with beforehand. However, the process of data cleaning is tedious and time-consuming, especially when the availability of information increases day by day. People usually prefer other more interesting tasks such as visualization or statistical computing instead of getting stuck in data cleaning. As troublesome as data cleaning is, every data scientist is aware that thorough, well-documented data cleaning is vital to the success of data analysis. Considering the inevitability and importance of data cleaning, data scientists are eager to find ways to automate this process. As a consequence, there is a great need of a powerful tool to help us effectively clean the raw datasets.

1.2 Thesis Objective

Datasets are an integral part of the machine learning field. The need for large amounts of data to train and run machine learning models makes the quality of datasets especially crucial in the machine learning field. For machine learning models to accurately learn, the datasets being used to train them must be trustworthy. Moreover, it is widely known that Python is popular in machine learning as it is elegant, flexible and straightforward. Therefore, there is a demand to develop an automatic data cleaning tool in Python for machine learning.

This thesis is aimed at *developing a Python tool which can offer automated, data-driven support to help users clean data effectively and smoothly.*

The objective of the Python tool can be formulated as follows: *Given a random raw dataset representing a machine learning problem, the Python tool is capable of automatically identifying* *the potential issues and reporting the results and recommendations to the end-user in an effective way.*

Considering the various dataset formats and machine learning tasks, for convenience and consistency, we design the tool aimed for supervised learning tasks and based on the parsed datasets from OpenML [65]. To be noticed, some datasets on OpenML may not be parsed, but we can utilize the feature in the Python API that automatically maps original data to tabular numerical data in that situation.

1.3 Results

The final version of the data cleaning tool is capable of presenting an overview report and cleaning common data problems. With the OpenML dataset ID as the input, the tool can show useful information about the given dataset, for example, the most important features and the data type of each feature. The tool is also good at dealing with common data problems. It first detects the data problems and presents them to the end-user using effective visualization techniques. Next, it recommends the proper technique to help the end-user clean data easily. Strictly speaking, the final tool is not only a data cleaning tool but also involves some other stages of data mining, for example, data understanding (data type discovery). To make it more clear, we summarize the capabilities of the tool as follows:

- Present an overview report of the given dataset
  - The most important features
  - Statistical information: mean, min, max and so on
  - **Data types of features**
- Clean common data problems in the raw dataset
  - Duplicated records
  - Inconsistent column names
  - **Outliers**
  - **Missing values**

Among the capabilities above, we highlight the three aspects we meaningfully automated: automatic discovery of data types, automatic missing value handling, and automatic outlier detection.

- Automatic discovery of data types
  - Discover common data types: boolean, float, integer, date and string
  - Discover statistical data types: real, positive-real, count and categorical
- Automatic missing value handling
  - Identify missing values
  - Visualize missing values
  - Clean missing values
- Automatic outlier detection
  - Identify both univariate and multivariate outliers
  - Visualize both univariate and multivariate outliers

These three aspects are the main focus of the thesis. The concrete approaches will be elaborated in Chapter 4.

1.4 Outline

In Chapter 2, the problems to be solved in this thesis are formulated and challenges for each problem are analyzed. Chapter 3 provides the background and describes the related work about data cleaning. Common data problems and corresponding cleaning techniques are examined. Chapter 4 demonstrates our approach to addressing common data problems and explains how we design the data cleaning tool to assist users in cleaning data. Chapter 5 summarizes the thesis and presents conclusions and possible improvements for future work.

Chapter 2

Problem Statement

Data cleaning can be performed at different levels of granularity. For example, some data cleaning tools may be dedicated in one specific data problem such as outlier detection, while some tools may cover a wide range of data problems. For another example, for the missing value issue, we can simply delete all the records containing missing values. However, this method may substantially reduce the information of the given dataset (imagine the extreme case that every record in the dataset contains missing values, and nothing would be left after deletion). We can also use advanced methods such as multiple imputation to fill in missing data. Besides, we can even take a step further by using different techniques according to the characteristics of the specific dataset.

The ultimate goal of this project is to develop a Python tool to help data scientists understand and clean raw data. To develop such a tool, we first have to investigate what are the most common data problems and the existing data cleaning techniques. Then we determine the core capabilities of our tool, that is the main data problems our tool focuses on addressing. Afterward, we integrate the state-of-the-art techniques into our tool to handle these issues. Last, we improve our tool a further step on this basis. To summarize, this work answers the following questions:

- *What are typical data problems in raw data?*
- *How do the state-of-the-art techniques deal with these data problems?*
- *How can we integrate these techniques into a data cleaning Python tool?*
- *How can we recommend the right techniques for the data at hand?*
- *Where can we improve the existing techniques?*

We are interested in how existing techniques deal with dirty data and wish to find where can we improve the current approaches. We wish to gain a deeper insight into state-of-the-art data cleaning techniques through the design and implementation of the data cleaning tool. After preliminary study, we determine the three key aspects we would like to address:

- Automatic discovery of data types
- Automatic missing value handling
- Automatic outlier detection

We further put forward the research questions next for each of the subtask above.

2.1 Automatic Discovery of Data Types

2.1.1 Problem Formulation

Automatic discovery of data types addresses the following two questions:

- *Which data types are important in the context of machine learning?*
- *How can we optimally detect the data types from a raw dataset?*

2.1.2 Challenges

There are many data types in computer science. However, not all the data types are useful for understanding machine learning problems. For instance, a feature known as 'object' is not as informative as a feature known as 'categorical'. Besides, since the datasets only contain a finite number of samples, it is difficult for us to distinguish whether a variable takes values from a finite set or infinite set. For example, it is complicated to know if a continuous variable can take values from the entire real line or only an interval of it. Moreover, we may need background information to determine the type of a variable. As an example, it is very difficult to distinguish between categorical and ordinal data since the presence of an order in the data only makes sense given a context. While colors in candies usually do not present an order, colors in the traffic lights clearly do. The limitations above contribute to the complexity and difficulty of data type discovery.

2.2 Automatic Missing Value Handling

2.2.1 Problem Formulation

The following research questions are explored for the automatic missing value handling

- *How are missing data usually encoded (0, 999, NAN or some other characters)?*
- *How can we visualize missing data effectively?*
- *How do existing approaches deal with missing data?*
- *How to recommend a proper technique to clean missing data for a given dataset?*

2.2.2 Challenges

First, missing data may be encoded as a variety of numbers or characters such as 0, 'nan' or '?'. The identification of missing values should consider all these possibilities. Second, there are a considerable amount of techniques available for dealing with missing values. It is challenging to find the optimal approach from all these options. Last, understanding the missing data is significant for a non-expert to select the proper approach. Hence we need to present the missing data to users in a straightforward manner.

2.3 Automatic Outlier Detection

2.3.1 Problem Formulation

We put forward the following questions for the automatic outlier detection part.

- *How do existing approaches detect outliers?*
- *How to recommend a proper outlier detection technique for a given dataset?*
- *How can we visualize outliers effectively?*

2.3.2 Challenges

The main challenge of outlier detection is that we have very limited information given a random dataset. To be more specific, we do not know the percentage of outliers in the dataset or which samples are outliers or inliners. Consequently, it is difficult to know which algorithm performs better on this dataset even though the outlier detection techniques have already been restricted in the unsupervised learning field. Moreover, we are usually more interested in multivariate outliers for machine learning problems. However, sometimes a dataset may contain hundreds of features which makes the visualization of outliers more difficult as we are visualizing high dimensional data.

Chapter 3

Literature Analysis

In this chapter, we provide the background knowledge and related work concerning automatic data cleaning. Section 3.1 state the concept of data cleaning. In Section 3.2, 3.3 and 3.4 we explore the major data problems in raw data and investigate the corresponding state-of-art cleaning techniques. We describe the visualization techniques which can be used to present data problems in Section 3.5. Related data cleaning tools are discussed in Section 3.6.

3.1 Data Cleaning

There is a massive amount of data created every single day. Machine learning can learn and make predictions on these data to make data valuable [48]. However, a major problem is that data in real life almost never come in a clean way [31] and poor data quality may severely affect the effectiveness of learning algorithms [17, 58]. Consequently, raw data need to be preprocessed before being able to proceed with training or running machine learning models as shown in Figure 3.1. Important and inevitable as data preprocessing is, this process is also tedious and troublesome. Data scientist usually spend more than half of analysis time on it [46], nevertheless non-expert users. As a result, data scientists are eager to find a tool to help them automate this process [33, 36, 51].

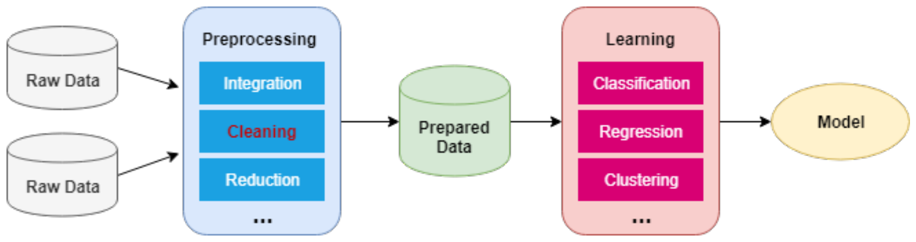


Figure 3.1: A brief machine learning process

There are many different tasks in data preprocessing such as data cleaning, data integration, and data transformation [29]. The task which aims at dealing with data problems is called data cleaning. Common data problems are missing values, outliers, inconsistent column names etc [37]. We briefly introduce the major problems as follows.

- Inconsistent columns names: Column names have inconsistent capitalizations.
- Duplicated records: Different or multiple records refer to one unique real-world entity or object in the dataset [73].
- Redundant features: Irrelevant features barely contribute to model constructions and may increase the training time and risk of overfitting [26].

- Inaccurate data types: The absence or inaccuracy of feature data types makes it difficult to understand the machine learning problem represented by the dataset.
- Missing values: No data value is stored for the feature in an instance [69]. Missing values are common and can have a significant effect on the conclusions that can be drawn from the data.
- Outliers: An outlier is an observation point that is distant from other observations which can cause severe problems in statistical analysis [25].

Data cleaning intends to clean data by filling in missing values, smoothing noisy data, identifying or removing outliers, and resolving inconsistencies [29]. This thesis seeks to develop a tool capable of addressing all the problems mentioned above. Among these problems, we mainly focus on inaccurate data types, missing values and outliers. In Section 3.2, 3.3 and 3.4, we elaborately demonstrate these three issues and examine the corresponding existing cleaning techniques.

3.2 Automatic Discovery of Data Type

Data types are significant for users to understand a random dataset. As an example, a dataset is usually presumed as representing a classification problem if the target feature is known as the categorical type. The information of feature data types helps the user gain a general idea of the meaning of the dataset. Moreover, features with different data types need to be processed differently for future analysis, for example, we perform one-hot encoding for categorical features and normalization for numerical features. Therefore, it would be a great advantage if we know the accurate feature data types beforehand.

3.2.1 Useful Data Types in Machine Learning

Different type systems support different kinds of data types as shown in Table 3.1. In practice of machine learning, we can quickly discuss and evaluate the preprocessing or encoding options with the reference of feature data types. For example, we may perform regression to impute missing values for a numerical feature and replace missing values by the most frequent value for a categorical feature.

Type Systems	**Data Types**
Statistics	real-valued, count, binary, categorical, ordinal, etc.
Pandas dtype	object, int64, float64, bool, datetime64, category, etc.
Python	str, int, float, bool, list, etc.
NumPy type	string_, unicode_, int_, int8, uint16, float_, float16, etc.
JSON schema	string, number, integer, object, boolean, null, etc.

Table 3.1: Data types in different type systems

Despite various data types, they are not equally important in the machine learning field. Experiments such as that conducted by Breiman [12] showed that statistical data types in a dataset provides particularly useful information in the context of machine learning. Here, we briefly describe six important statistical data types. As we know, the data type of a given feature can either be continuous or discrete. Continuous data can be further classified as real valued, positive valued or interval data whereas discrete data can be further classified as categorical, ordinal or count data.

- Continuous variables:
  1. Real-valued data, which takes values in the real number line.
  2. Positive real-valued data, which takes values in the positive real number line.

3. Interval data, which takes values in an interval of the real number line.

- Discrete variables:
    1. Categorical data, which takes values in a finite unordered set, e.g., $x_n^d \in \{'blue','red', 'black'\}$.
    2. Ordinal data, which takes values in a finite ordered set, e.g., $x_n^d \in \{'never','sometimes', 'often','usually','always'\}$.
    3. Count data, which takes values in the natural numbers, i.e., $x_n^d \in \{0, \cdots, \infty\}$.

The above are the six data types that matter in machine learning. In the next, we will investigate the approaches to distinguish between these types.

3.2.2 Data Type Discovery Techniques

There are many feasible approaches to discover data types from a raw dataset. Some approaches are simple, and may only require some statistics or heuristics. For example, to detect whether a feature is discrete or continuous, we can count the number of unique values that feature takes and compare it with the number of instances of that feature. Some approaches are more advanced or complex, which may require machine learning models to detect.

Heuristic Method

A Python package messytables [39] guesses data types by brute force guessing. Brute force guessing first takes a sample from a particular column and then tries to convert every single element in the sample into all possible data types. The number of successful conversions is counted per data type and then a majority vote determines the most probable data type for that column. The following data types are considered in messytables: String, Integer, Decimal, Bool and Date. This approach is flexible and easy to implement.

It is widely known that data can be stored in various formats such as CSV, XML, and JSON. Data are stored in different formats according to different rules. For example, XSD (XML Schema Definition) specifies how to formally describe the elements in an XML document [72]. Hence, we can infer the data types of elements utilizing the schema information. Schema Inference is a technique which is used to infer XSD after parsing the structure of any XML document, which gives some rules to discover data types of elements from XML document. Table 3.2 shows some of the schema inference rules to infer data types of attributes of elements from XML document.

Inferred Data Type	**Attributes Value for XML Element**
boolean	If it is true or false.
int	If it is integer value between -2147483648 to 2147483647.
float	If it is decimal value between -16777216 to 16777216.
byte	If it is integer value between -128 to 127.
string	If it is single or more than one Unicode format.

Table 3.2: Infer data types by schema inference [16]

Similar to messytables, the data types discovered by schema inference are not that useful for machine learning. An Integer feature is still too general and we are more interested in knowing the statistical types of a given feature. Even so, these approaches still provide a decent base on which we can further detect the data types using advanced methods.

Bayesian Method

Isabel Valera [64] proposes a Bayesian method to determine the statistical types of features. This proposed method is based on probabilistic modeling and exploits the following key ideas:

1. There exists a latent structure in the data that captures the statistical dependencies among the different objects and attributes in the dataset. Here, as in standard latent feature modeling, Valera assumes that this structure can be captured by a low-rank representation, such that conditioning on it, the likelihood model factorizes for both number of objects and attributes [64]. In other words, Valera proposes that a dataset can be represented by a matrix X which is the input of the probabilistic model and X can be factorized as two low-rank matrices Z and B as shown in Figure 3.2.

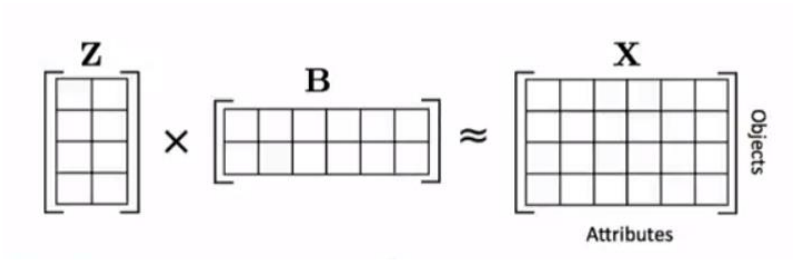


Figure 3.2: Low rank representation of a dataset

2. Each attribute is represented by an observation model which can be expressed as a mixture of likelihood functions, one per each considered data type, where the inferred weight associated to a likelihood function captures the probability of the attribute belonging to the corresponding data type [64]. Simply speaking, each attribute (feature) in the dataset x^d (d^{th} attribute) has a likelihood model which is a mixture of likelihood functions (each likelihood function represents a data type). For each likelihood function, a weight is assigned, and these weights sum up to one.

3. Then Valera derives an efficient Markov Chain Monte Carlo (MCMC) [8] inference algorithm to jointly infer both the low-rank representation and the weight of each likelihood model for each attribute in the observed data. The weights for each likelihood function are computed after the algorithm and the likelihood function with the highest weight will be considered as the data type of the given feature.

3.3 Automatic Missing Value Handling

Missing data is one of the common problems in practice as a result of manual data entry procedures, equipment errors, incorrect measurements, intentional missing and so on. A relatively few absent observations on some variables can dramatically shrink the sample size. As a result, the precision and efficiency of data analysis are harmed, statistical power weakens, and the parameter estimates may be biased due to differences between missing and complete data [40]. In machine learning, missing data will increase the misclassification error rate of classifiers [2]. Thus missing data need to be dealt with before training machine learning models.

3.3.1 Missing Data Mechanisms

In order to handle missing values effectively, the first step is to understand the data and try to figure out why the data is missing. Sometimes attrition is caused due to social or natural processes, for example, school graduation, dropout, and death. Skip pattern (the process of skipping over non-applicable questions depending upon the answer to a prior question) in the survey will also lead to missing data, for example, certain questions only asked to respondents who indicate they are married. A good understanding of data helps us determine the mechanism of missing data. Missing data mechanisms can be classified into three types [40, 55]:

- Missing Completely at Random (MCAR): There is no pattern in the missing data on any variable. For example, questionnaires get lost by chance during data collection.

- Missing at Random (MAR): Missing at random means that the propensity for a data point to be missing is not related to the missing data, but it is related to some of the observed data. As an example, suppose managers are more likely not to share income than staff, in which case the missingness in feature income is related to the feature profession.

- Missing not at Random (MNAR): The probability of a missing value depends on the variable that is missing. For example, respondents with high income may be less likely to report income.

Identifying the missing data mechanism is important for choosing the strategy to deal with missing data. For example, deletion is generally safe for MCAR while should be avoided for MAR and MNAR [5].

3.3.2 Missing Value Handling Techniques

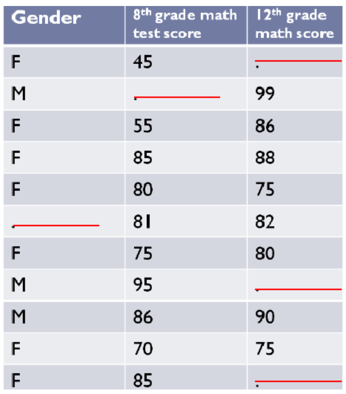
After determining the mechanism of the missing data, the next step is to decide the appropriate method to clean them. In this part, we introduce the state-of-the-art techniques for dealing with missing values.

Listwise Deletion

Listwise deletion [40] also known as complete case analysis, only analyzes cases with available data on each variable, as shown in Figure 3.3(a). Listwise deletion is very simple but and works well when missing mechanism is MCAR and sample size is large enough [47]. However, it reduces the statistical power and may lead to biased estimates especially for MAR and MNAR [52].

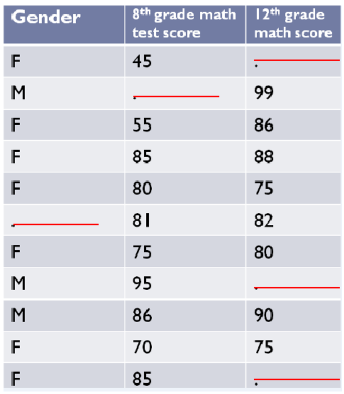
Pairwise Deletion

Different from listwise deletion, pairwise deletion also known as available case analysis, analyzes all cases in which the variables of interest are present, as shown in Figure 3.3(b). Compared with listwise deletion, pairwise deletion uses all the available information for analysis. For example, when exploring the correlation between two variables, we can use all the available cases of these two variables without considering the missingness of other variables. However, like listwise deletion, pairwise deletion only provides unbiased estimates in MCAR [52].



Gender	8th grade math test score	12th grade math score
F	45	
M		99
F	55	86
F	85	88
F	80	75
	81	82
F	75	80
M	95	
M	86	90
F	70	75
F	85	

(a) Listwise deletion [32]



Gender	8th grade math test score	12th grade math score
F	45	
M		99
F	55	86
F	85	88
F	80	75
	81	82
F	75	80
M	95	
M	86	90
F	70	75
F	85	

(b) Pairwise deletion [32]

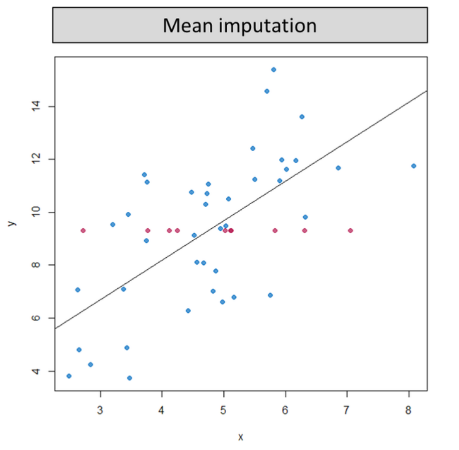
Figure 3.3: Listwise and pairwise deletion

Mean/Median/Mode Imputation

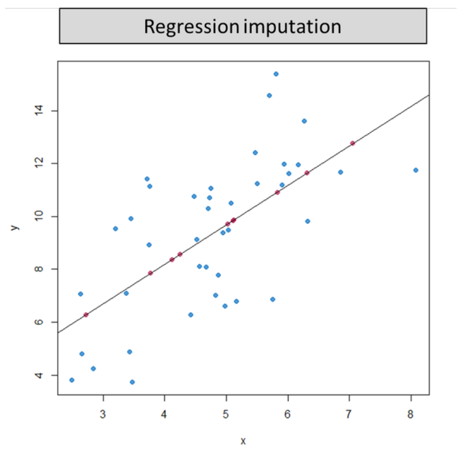
We can substitute missing values under a variable with statistical information such as mean, median or mode [27], as shown in Figure.3.4(a). This method uses all the data. However, it also underestimates the data variability since the true missing value may be far from mean, median or mode. Besides, it may also weaken the covariance and correlation estimates in the data due to the ignorance of the relationship and dependency between variables.

Regression Imputation

Regression Imputation replaces missing values with the predicted score from a regression equation, as shown in Figure.3.4(b). This method uses information from observed data but also presumes that the missing values fit the regression trend. Thus all the imputed values fit the regression model perfectly which leads to the overestimation of the correlation between variables. Stochastic regression [20] is put forward to address this problem. It adds random error to the predicted score, which supplies the uncertainty to the imputed values. Compared with simple regression, stochastic regression shows much less bias [20], but variance can still be underestimated since the random error may not be enough.



(a) Mean imputation shows the relation between x and y when the mean value is imputed for the missing values on y [19].



(b) Regression imputation assumes that the imputed values fall directly on a regression line with a nonzero slope, so it implies a correlation of 1 between the predictors and the missing outcome variable in the example [19].

Figure 3.4: Mean and regression imputation

Multiple Imputation

In order to reduce the bias generated from imputation, Rubin [56] proposed a method for averaging the outcomes across multiple imputed datasets. There are basically 3 steps in multivariate imputation. First, impute the missing data of the incomplete datasets m times ($m = 3$ in Figure 3.5). Note that imputed values are drawn from a distribution. This step results in m complete datasets. The second step is to analyze each of the m completed datasets. Mean, variance, and confidence interval of variables of concern are calculated [75]. Finally, we integrate the m analysis results into a final result.

Multiple imputation is the most sophisticated and most popular approach currently. The most widely-used multiple imputation approach is Multivariate Imputation by Chained Equation (MICE) [9] based on the MCMC algorithm [8]. MICE takes the regression idea further and take advantage of correlations between responses. To explain the idea of MICE, we give an example of

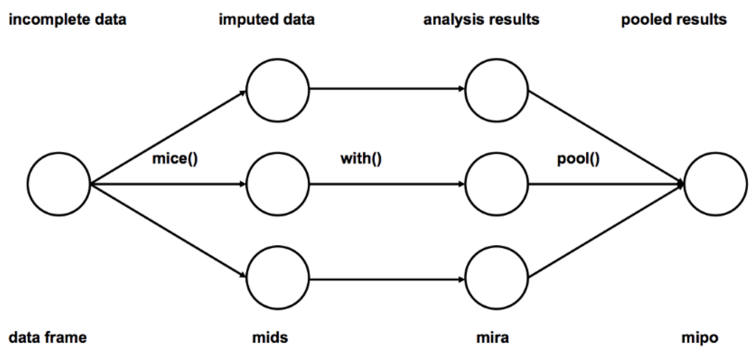


Figure 3.5: Multiple imputation process [56]

imputing the missing values for a simple dataset by MICE. Imagine we have three features in our dataset: profession, age and income, and each variable has some missing values. The MICE can be conducted through the following steps:

1. We first impute the missing values using a simple imputation method, for example, mean imputation.
2. We set the imputed missing values of variable profession back to missing.
3. We perform a linear regression to predict the missing values of profession by age and income using all the cases where profession are observed.
4. We impute the missing values of profession by the values obtained in step 3. And variable profession has no missingness at this point.
5. We repeat steps 2-4 for variable age.
6. We repeat steps 2-4 for variable income.
7. We repeat the entire process of iterating the three variables convergence.

Multiple imputation is aimed for MAR specially but it is found that it also produces valid estimate in MNAR [5].

Matrix Factorization

Matrix factorization is basically factorizing a large matrix into two smaller matrices called factors. Factors are multiplied to obtain the original matrix. There are many matrix factorization algorithms which Nonnegative MF and Multi Relational Matrix Factorization, which can be used to fill in missing data [11].

Matrix factorization is widely used to impute missing values in recommendations systems. We take music recommendations as an example. Table 3.3 shows a user-music rating matrix. Imagine we have 3 users u and 4 music m, we know that this matrix would be very sparse in real life as every user only listens to a small part of music in the music library.

	m_1	m_2	m_3	m_4
u_1		w_{12}^{um}		
u_2	w_{21}^{um}			
u_3			w_{32}^{um}	

Table 3.3: User-Music rating matrix R

Assume that they are only two music styles s_1, s_2 in the world, then we can factorize the matrix R to user-style preference matrix U and style-music percentage matrix V, as shown in Table 3.4.

	s_1	s_2
u_1	w_{11}^{us}	w_{12}^{us}
u_2	w_{21}^{us}	w_{22}^{us}
u_3	w_{31}^{us}	w_{32}^{us}

	m_1	m_2	m_3	m_4
s_1	w_{11}^{sm}	w_{12}^{sm}	w_{13}^{sm}	w_{14}^{sm}
s_2	w_{21}^{sm}	w_{22}^{sm}	w_{23}^{sm}	w_{24}^{sm}

Table 3.4: User-Style preference matrix U and Style-Music percentage matrix V

Hence if we can get matrix U and V, we can fill the missing values in R. U and V can be computed by solving the loss function with gradient descent. The loss function is defined by the distance between $\tilde{R} = UV^T$ and R:

$$arg \min_{U,V} = \mathcal{L}(R, UV^T) + \lambda(||U||_F^2 + ||V||_F^2)$$

where $\lambda(||U||_F^2 + ||V||_F^2)$ is the regularization to prevent from overfitting. And missing values can be estimated as shown in Figure 3.6.

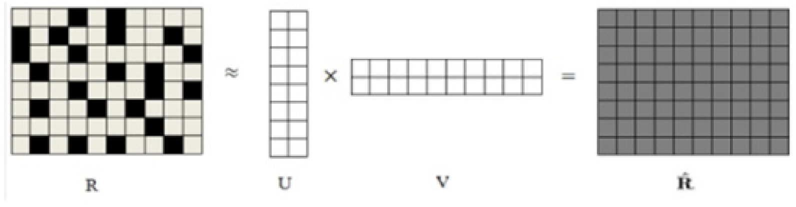


Figure 3.6: Matrix factorization

K Nearest Neighbor

There are other machine learning techniques such as XGBoost and Random Forest [62] for data imputation. K Nearest Neighbor (KNN) is the most widely used. In this method, k neighbors are selected based on the distance measure and their average is used as an imputation estimate. KNN can predict both discrete attributes (the most frequent value among the k nearest neighbors) and continuous attributes (the mean among the k nearest neighbors) [43]. The advantage of the KNN algorithm is that it is simple to understand and easy to implement. Unlike multiple imputation, the KNN basically asks for no parameter which gives it an edge in certain settings where the information of dataset are barely provided [34]. One of the obvious drawbacks of the KNN algorithm is that it becomes time-consuming when analyzing large datasets because it searches for similar instances through the entire dataset.

Summary

Collectively, there are many feasible approaches to deal with missing values. Different approaches apply to different situations. Deletion can only apply to MCAR without causing a big bias. Imputations using statistical information basically also only apply to MCAR as they are making up data without considering the correlation between variables. KNN, matrix factorization and MICE are widely used in MAR. MICE performs well in all missing mechanisms generally. There are also many other missing imputation techniques such as maximum likelihood [7] and missing indicator [24]. We do not elaborate them since they are either too complicated to be automated or can only be applied to MCAR.

3.4 Automatic Outlier Detection

In machine learning, the process of detecting anomalous instances within the datasets is known as outlier detection or anomaly detection. Even though modern classifiers are designed to be more robust to outliers, there are still many classifiers quite sensitive to outliers [1]. Hence, users need to be aware of the outliers in the dataset and select appropriate approaches to handle them before inputting data into training models.

3.4.1 Categorization of Outlier Detection

Outliers exist in both one-dimensional and multi-dimensional space. Detection of outliers in one-dimensional data depends on their distribution. The normal distribution is the most used when the distribution is not known [14]. Compared with one-dimensional outlier detection, multi-dimensional outlier detection is much more complicated. There are different setups of outlier detection depending on whether the labels are available, as shown in Figure 3.7. In this section, we introduce the three main types of outlier detection: supervised outlier detection, semi-supervised outlier detection, and unsupervised anomaly detection.

Supervised Outlier Detection

Supervised anomaly detection describes the setup where training datasets and test datasets are both fully labeled [23]. In this scenario, we know which data are outliers in the training datasets. This scenario is very similar to traditional supervised classification tasks. The difference is that classes in supervised anomaly detection are highly unbalanced.

Semi-supervised Outlier Detection

Semi-supervised anomaly detection also uses training and test datasets, whereas training data only consists of normal data without any outliers [23]. A model is learned from normal data and outliers can be detected as they deviate from this model.

Unsupervised Outlier Detection

Unsupervised anomaly detection is the most flexible setup which does not require any labels [23]. The idea is that unsupervised outlier detection techniques score the data solely based on the intrinsic properties of the dataset such as distance and density.

Summary

When given a random unseen raw dataset, we barely have any information about it. This means outliers are usually not known in advance. Consequently, the assumption that normal data and outliers are labeled correctly of supervised anomaly detection unsupervised can be rarely satisfied. Besides, as mentioned previously, data almost never come in a clean way, which also limits the use of semi-supervised anomaly detection. Overall, unsupervised anomaly detection algorithms seem to be the only reasonable choice for our data cleaning tool.

3.4.2 Outlier Detection Techniques

In this section, we take an insight into the most used unsupervised outlier detection algorithms as well as the one-dimensional outlier detection standard deviation method which can be used to serve our data cleaning tool.

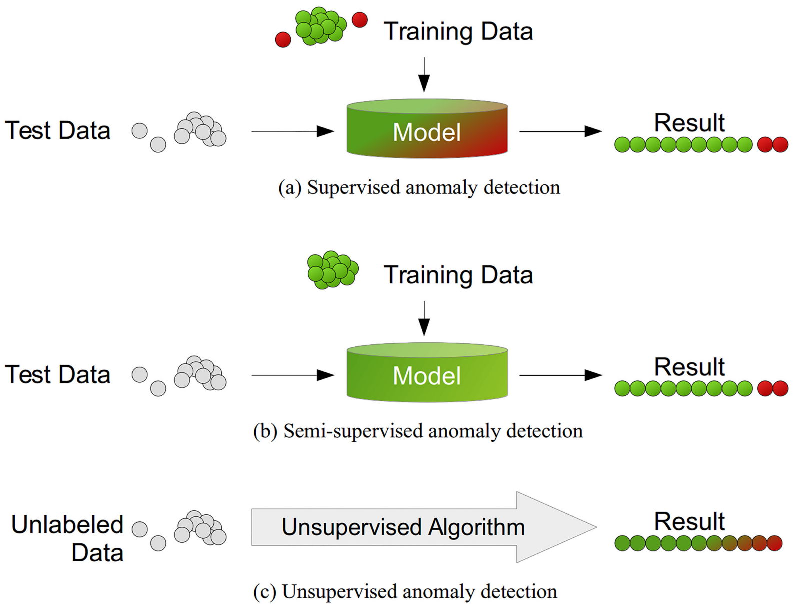


Figure 3.7: Outlier detection modes depending on the availability of labels in the dataset [23]

Standard Deviation

Standard deviation is a metric of variance, indicating how much the individual data points are spread out from the mean. For this outlier detection method, the mean and standard deviation of the residuals are calculated and compared. If a value is a certain number of standard deviations away from the mean, that data point is identified as an outlier. The default value is 3. As we can see from Figure 3.8, dark blue is less than one standard deviation from the mean. For the normal distribution, this accounts for about 68% of data, while two standard deviations from the mean (medium and dark blue) account for about 95%. The three standard deviations (light, medium, and dark blue) account for about 88.7%. Data outside the three standard deviations are considered as outliers. However, to be noticed, standard deviations can fail to detect outliers if the outliers are extreme. Because the extreme outliers increase the standard deviation. The more extreme the outlier, the more the standard deviation is affected [71]

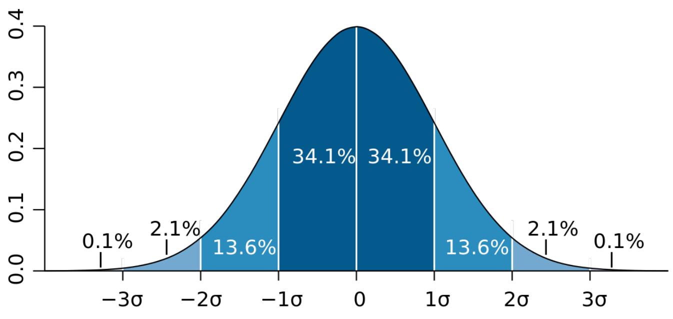
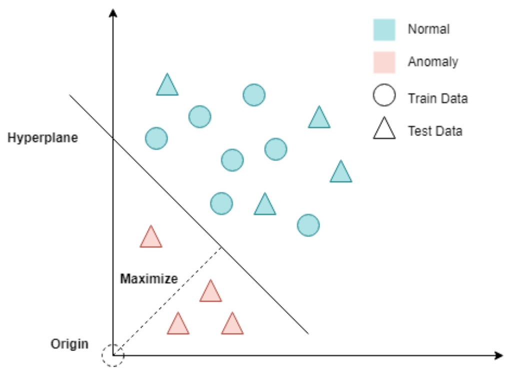


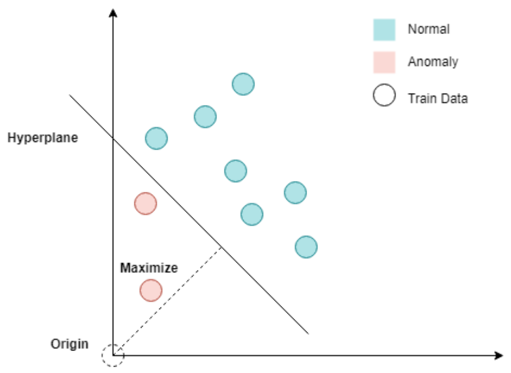
Figure 3.8: Standard Deviation [71]

One-class Support Vector Machine

One-class support vector machine (OCSVM) by Scholkopf [57] intends to separate all the data from the origin in the feature space F (Feature space refers to the n dimensions where features live [48]) by a hyperplane and maximizes the distance from this hyperplane to the origin [67], as shown in Figure 3.9(a). Technically speaking, this OCSVM put forward Scholkopf is heavily used as a semi-supervised method where training data needs to be anomaly-free. To make OCSVM applicable for unsupervised scenario, an enhanced OCSVM is proposed [6]. A parameter v is introduced to indicate the fraction of outliers in the dataset, which allows some data on the other side of the hyperplane, as shown in Figure 3.9(b). And each instance in the dataset is scored by a normalized distance to the determined hyperplane [23]. The basic idea is that outliers contribute less to the hyperplane than normal instances. Due to the importance of the parameter v this method is also called v-SVM.



(a) One-class SVM



(b) Enhanced One-class SVM

Figure 3.9: One-class Support Vector Machine

Local Outlier Factor

Local outlier factor (LOF) is the most well-known local anomaly detection algorithm and also introduced the idea of local anomalies first [13]. Today, its idea is carried out in many nearest-neighbor based algorithms. The LOF algorithm computes the local density deviation of a given data point with respect to its neighbors. The following steps show how to calculate the local density deviation of a data point o:

1. Compute the k-distance of data point o: $dist_k(o)$ = distance between o and its k^{th} nearest neighbor.
2. For each data point, compute set of points in k-distance $N_k(o)$.
3. Compute reachability distance for each data point o with respect to data point o', as shown in Figure 3.10.
$$reach\_dist_k(o', o) = \max\{k\text{-distance}(o), d(o', o)\}$$
4. Compute local reachability density (lrd):
$$lrd_k(o) = \frac{|N_k(o)|}{\sum_{o' \in N_k(o)} reach\_dist_k(o', o)}$$
5. Finally, compute Local outlier factor score:
$$LOF_k(o) = \frac{\sum_{o' \in N_k(o)} \frac{lrd(o')}{lrd(o)}}{|N_k(o)|}$$

The local density deviation depends on how isolated the data point is with respect to the surrounding neighborhood. More precisely, locality is given by k-nearest neighbors, whose distance is used to estimate the local density. The samples with substantially lower local density will result in larger LOF score, and are considered as outliers.

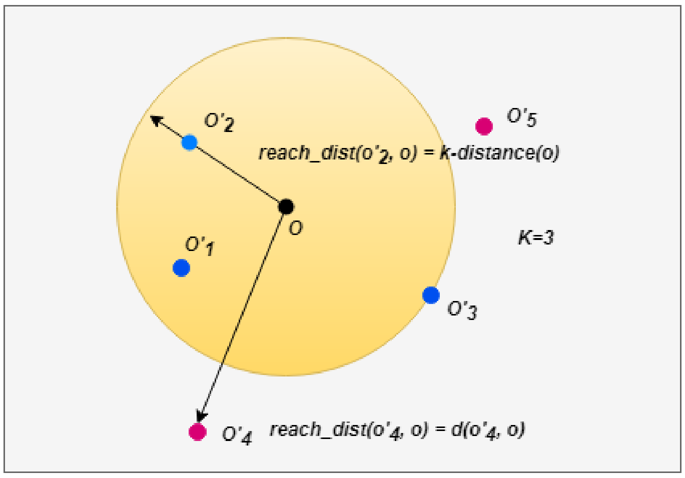

O'2
O'5
reach_dist(o'2, o) = k-distance(o)
O
K=3
O'1
O'3
O'4
reach_dist(o'4, o) = d(o'4, o)

Figure 3.10: Compute reachability distance (k=3)

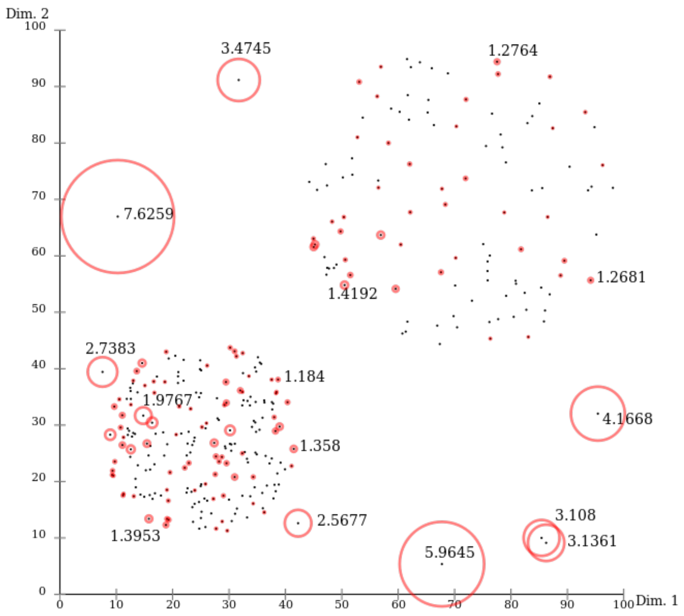


Figure 3.11: Local outlier factor [68]

Isolation Forest

Liu [41] proposed an unsupervised outlier detection algorithm isolation forest (iForest) on the basis of decision trees. IForest partitions data by first randomly selecting a feature and then selecting a random split value between the minimum and maximum value of the selected feature. These partitions can be represented as a tree structure. The idea is that outliers are less frequent than normal data and are different from them in terms of values. Hence, they lie further away from normal data in the feature space. Consequently, outliers are easier to be separate from the rest of the data and closer to the root of the tree, as shown in Figure 3.12. A score is derived based on the path length, i.e., the number of edges a data point must pass in the tree going from the root to the terminal node. The score s is defined as follows:

$$s(x, n) = 2^{-\frac{E(h(x))}{c(n)}}$$

where $h(x)$ is the path length of observation x, $c(n)$ is the average path length of unsuccessful search in a binary search tree, n is the number of external nodes. It is worth noticing that this method has a known weakness when the anomalous points are tightly clustered [41].

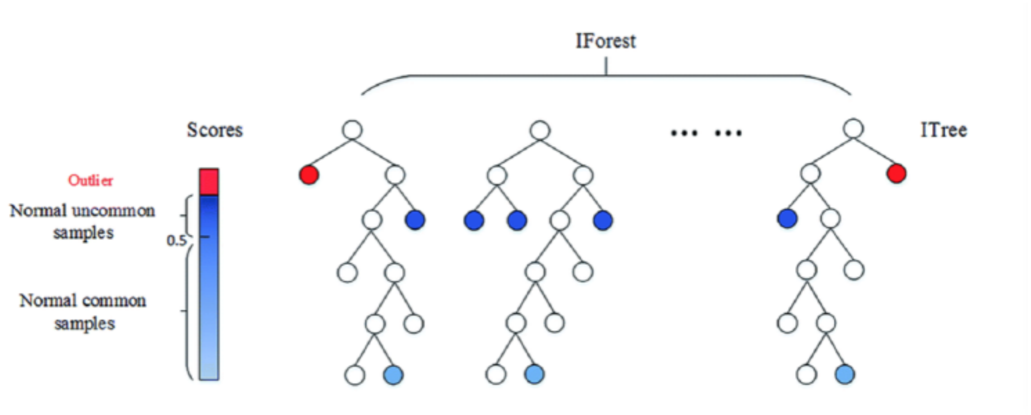


Figure 3.12: Isolation Forest [15]

3.4.3 Dealing with Outliers

It is definitely not a good idea to directly remove the outliers as not all the outliers are synonyms for bad data. In general, outliers can either be a mistake in the data or a true outlier. The first type, a mistake in the data, could be as simple as typing 5000 rather than 50.00, resulting a big bias for the analysis process afterward. The second type, a true outlier, would be something like the population of China in the world population dataset, which is so different from the population of other countries but is true data. The following are some approaches to deal with outliers [22]:

- Drop the outlier records: remove the outliers completely from the dataset to keep that data from affecting the analysis.

- Assign a new value: If an outlier seems to be a mistake, we can treat it as a missing value and impute a new value.

- Transformation: A different approach to true outliers could be to try creating a transformation of the data rather than using the data itself. For example, convert data to a percentile version or perform log transformation as shown in Figure 3.13.

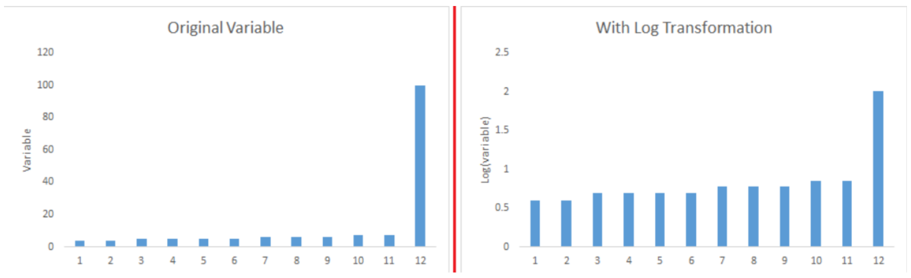


Figure 3.13: Deal with outliers by log transformation

3.5 Visualization Techniques

The basic purpose of visual representation is to efficiently interpret what is insight, as easy as possible [35]. However, with various visualization techniques, it may be confusing to know which one is appropriate to use in order to convey maximum possible understanding. A primary task of our data cleaning tool is to present the data in visualizations to help users understand the unseen dataset effectively. In this section, we introduce the common visualization techniques covered in this thesis and demonstrate the different situations each technique can be used.

3.5.1 Bar Chart

The bar chart is the most common known data visualization technique. The rectangular bars represent data and their lengths are proportional to the values they represent. There exist both vertical and horizontal bar charts. Figure 3.14 is a typical example of a vertical bar chart, sometimes called a line graph or histograms. Bar chart is suitable for tasks to compare and look up. It can be used when visualizing one quantitative value attribute and one categorical key attribute.

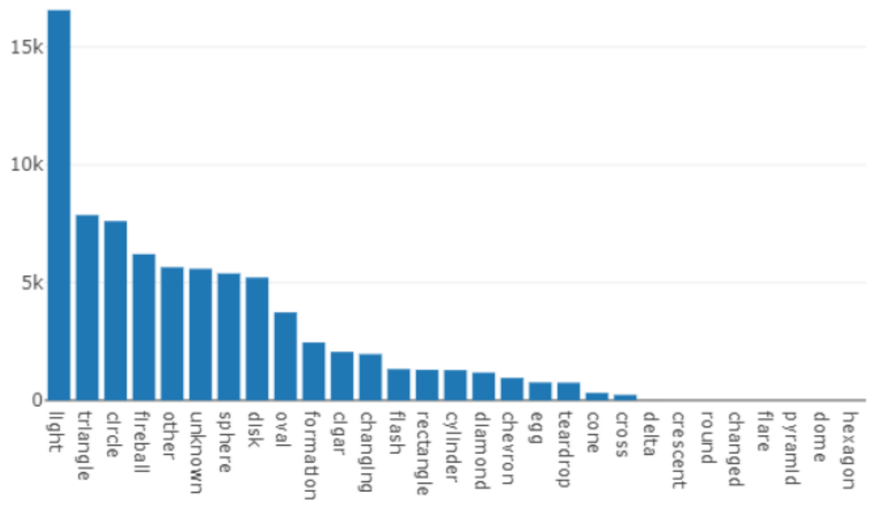


Figure 3.14: Bar chart

3.5.2 Box Plot

A box plot is another visualization technique for graphing numerical data. The box plot consists of 4 quartiles in which 25% of the samples is in each quartile. To illustrate this with an example take Figure 3.15. A box plot is mainly used for finding distribution and outliers are distinctively shown through it. As we can see in Figure 3.15, the values of these outliers are too much off in comparison with the greatest bulk of the data and are labeled as outliers.

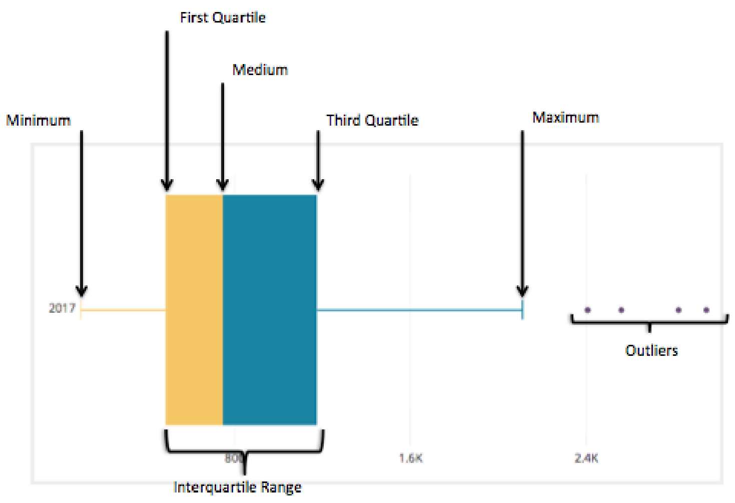


Figure 3.15: Box plot

3.5.3 Pie Chart

A pie chart is a circular statistical graphic which is divided into slices to illustrate numerical proportion. In a pie chart, the arc length of each slice (and consequently its central angle and area), is proportional to the quantity it represents, as shown in Figure 3.16. Pie chart shows the part-whole relationship and can be used to visualize one quantitative attribute and one categorical attribute.

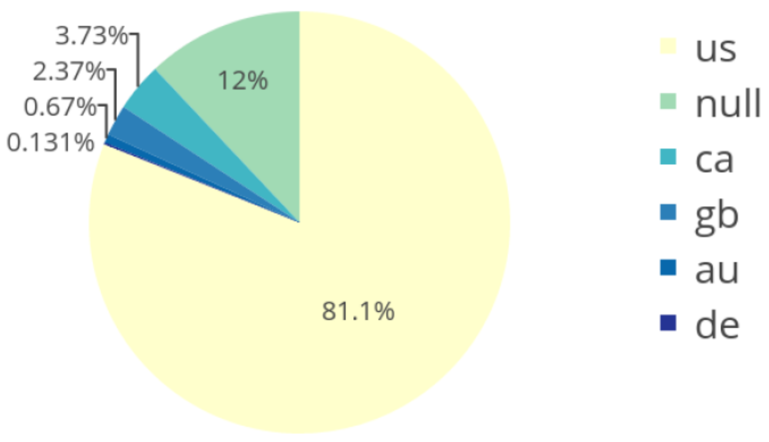


Figure 3.16: Pie chart

3.5.4 Scatter plot

A scatter plot is a type of mathematical diagram to display values for typically two variables for a set of data. The purpose is to identify the type of relationship (if any) between two quantitative variables. Scatter plot is good at tasks of finding correlations, trends or distribution. It can be used when visualizing multi-key table as shown in Figure 3.17.

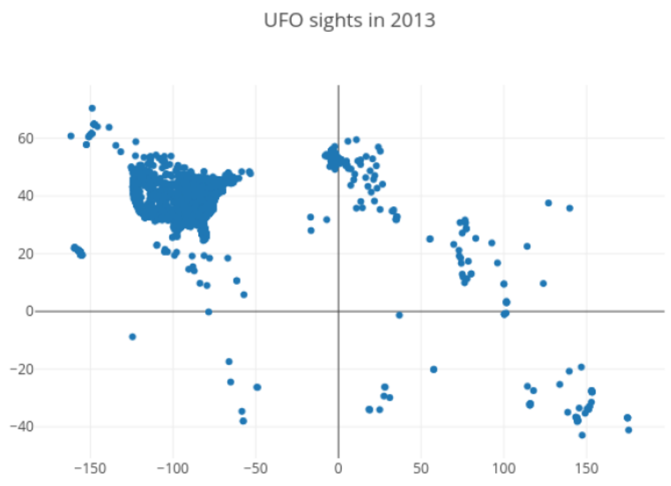


Figure 3.17: Scatter plot

3.5.5 Heatmap

A heatmap [74] is a graphical representation of data where the individual values contained in a matrix are represented as colors, as shown in Figure 3.18. It is good choice to adopt heatmap when dealing with tasks like finding clusters or summarizing. It can be used when there are two categorical key attributes and one quantitative attribute to visualize.

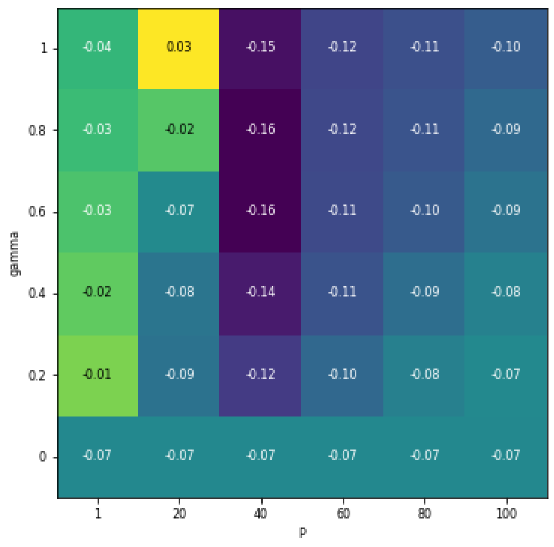


Figure 3.18: Heatmap

3.5.6 Parallel Coordinates Plot

The parallel coordinates plot (PCP) [30] is a common way of visualizing high-dimensional geometry and analyzing multivariate data. In a parallel coordinates plot, each variable is given its own axis and all the axes are placed in parallel to each other. Values are plotted as a series of lines that connected across all the axes. This means that each line is a collection of points placed on each axis, that have all been connected together. In this method, the trends, outliers, correlation, and extremes values could be easily identified. If clusters of similar lines are found, it might suggest the high chance to find a correlation. While similar crossing points might suggest a negative correlation. As for the outliers, there should be isolated lines or lines with different slopes than its neighbors. An example of parallel coordinates plot is given in Figure 3.19.

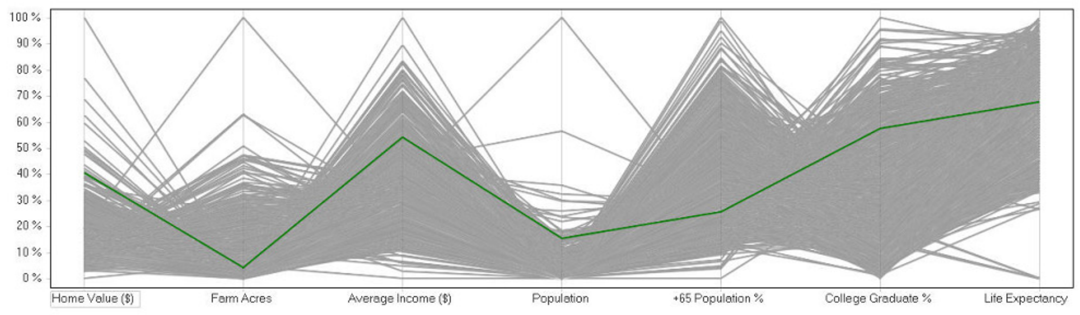


Figure 3.19: Parallel Coordinates Plot [35]

3.6 Related Work

There are a lot of great tools available for big data analytics such as Python, SAS, and Tableau. These tools can be used for cleaning data themselves or providing a basis for developing more fancy data cleaning tools. In this section, we first examine some commonly known data analytical tools and then take an insight into some existing modern data cleaning tools.

3.6.1 Data Analytical Tools

Data Analytical Tools can be generally categorized into three groups: programming languages (R, Python, SAS), statistical solutions (SPSS, STATA) and visualization tools (Tableau, D3). Users can choose one or some of them based on their programming background and specific usage.

SAS: SAS (Statistical Analysis System) is a powerful software which provides capabilities of accessing, transformation and reporting data by using its flexible, extensible and web-based interface [61]. It is highly adopted by industries and well-known for being good at handling large datasets.

SPSS: As a strong competitor of SAS, SPSS (Statistical Package for the Social Sciences) [49] is used by various kinds of researchers for complex statistical data analysis. Compared with SAS, SPSS is much easier to learn but very slow in handling large datasets.

Both SAS and SPSS are professional at dealing with dirty data. They are capable of dealing with common data problems such as missing values, outliers, and duplicated records. We can even find very advanced and complicated approaches such as multiple imputation and maximum likelihood for dealing with missing values [60]. However, the problem is neither of these two is free, and the visualization capabilities are purely functional. Moreover, they preprocess data on a very general level, not specifically for machine learning tasks.

R: R [63] is the open source counterpart of SAS. It is primarily focused on statistical computation and graphical representations. R is equipped with many comprehensive statistical analysis packages, which makes it popular among data scientists. Besides, libraries like ggplot2 make R competitive in data visualization.

Python: Python [45] is an interpreted high-level programming language and is developed with a strong focus on applications. Python is famous for its rich useful libraries, for example, NumPy, Pandas [44], Matplotlib, and scikit-learn [50]. NumPy and Pandas make it easy to operate on structured data. Visualization can be realized by Seaborn and Matplotlib. Moreover, Python has easy access to powerful machine learning packages such as scikit-learn, which makes Python especially popular in the machine learning field.

R and Python can make a stronger team together. There are packages for R that allow running Python code (reticulate, rPython), and there are Python modules which allow running R code (rpy2). Many data cleaning tools are designed based on these two programming languages. We will discuss them in the next section. SAS, SPSS, R, and Python can all be utilized to visualize data. But apart from these, there are softwares which are specifically focusing on visualization. Here, we briefly introduce two of them: Plotly and Tableau.

Plotly: Plotly is a data visualization library for python, R and javascript. Plotly provides a various of visualization techniques to present data in a fancy and interactive manner. Users can basically find any existing visualization techniques in Plotly which makes it possible to understand data from different perspectives.

Tableau: Tableau is a powerful software which provides a fast and intelligent way to analyze data visually and interactively. It can present any kind of data in the most perfect manner. And it can also be used for data preparation, but mostly for data integration and data reshape. Users can easily combine and transform data by moving and clicking the mouse. Tableau also provides simple functions for data cleaning, such as unify column names and remove one-dimensional outliers. But generally, it is aimed for data visualization.

Even though these tools can not satisfy our need to automatically clean data for machine learning tasks, they do provide many advanced approaches to clean or visualize data, which we can learn from when designing our tool.

3.6.2 Data Cleaning Tools

There are already a lot of tools which are capable of providing user support for different stages of data analysis, including data cleaning. In this section, we take an overview of these data cleaning tools such that we can find where to utilize and improve.

Pandas: Pandas is a powerful Python package which offers fast, flexible, and expressive data structures designed to make working with relational or labeled data both easy and intuitive [44]. Pandas are good at tasks such as data reshape and data transformation. For data cleaning, Pandas can be used to deal with problems such as duplicated records, inconsistent column names, and missing values. However, these capabilities seem a little bit plain and fundamental. For example, for imputing missing values, Pandas only provides three methods: use last valid observation, use next valid observation, and use a specific value to fill gap. There exists a lack of more advanced approaches. Pandas can also be used to detect data types. Data types supported by Pandas are float, int, bool, datetime64[ns], timedelta[ns], category and object. As mentioned in 3.2, we care more about statistical data types in machine learning. Hence data types need to be further discovered.

scikit-learn: Scikit-learn [50] is a free and effective machine learning library for Python. Compared with Pandas, Scikit-learn provides more powerful preprocessing functions, especially on data transformation. Also, scikit-learn can fill in missing values. The SimpleImputer class provides basic strategies for imputing missing values: mean, mode, median and specified value, which are still very simple. However, to be noticed, Scikit-learn features various classification, regression and clustering algorithms. Consequently, we can clean data utilizing these advanced algorithms. For example, we can detect outliers by taking advantage of the anomaly detection algorithms such as isolation forest, local outlier factor and one class support vector machine provided by scikit-learn. The problem is scikit-learn does not offer user assistance to choose appropriate algorithm and leaves room for us to improve on this aspect.

Weka: Weka [28] is an open source tool for data mining and allows users to apply preprocessing algorithms. However, it does not provide a guidance for the user in terms of algorithms selection.

Moreover, Weka is aimed at transforming data such that data can fit to the data mining algorithms. This simple transformation does not take improving the performance of algorithms into account. Overall, Weka focus more on the data mining step and the preprocessing part is more or less neglected.

dataMaid: The up-to-date R-package dataMaid is created by data scientists to assist the data cleaning process. It is aimed at helping the investigators to identify potential problems in the dataset [51]. Compared with other data cleaning tools mentioned in this part, dataMaid is the closest to the our thesis objective. The main capability of dataMaid is autogenerating data overview documents that summarize each variable in a dataset and flags typical problems, depending on the data type of the variable [51]. We can see that dataMaid focuses more on helping users understand the random dataset. For detecting and cleaning data problem, there leaves room for improvement.

We summarize the main characteristics or deficiencies these data cleaning tools may have as follows:

- They may only provide simple and plain data cleaning approaches.
- They may focus on addressing a single data problem.
- User assistance is not provided for approach selection.
- Most of them are not aimed at machine learning tasks.

Therefore, we can try to improve on these aspects when designing our data cleaning tool.

Chapter 4

Methodologies and Results

In this thesis, we developed a Python tool to offer automated, data-driven support to assist users clean data effectively. This tool can recommend the appropriate cleaning approaches according to the specific characteristics of the given dataset. The tool aims at improving the data quality such that better machine learning models can be trained. We address a wide range of data problems using existing approaches. But to be clear, the main focuses are automatic data type discover, automatic missing value handling and automatic outlier detection. In this chapter, we demonstrate the methodologies for designing this tool and present the results we achieved.

4.1 Automatic Discovery of Data Types

From the literature analysis, we already know that statistical data types (real-valued, positive real-valued, interval, categorical, ordinal, count) of features are more important in the machine learning field, as explained in Section 3.2. Hence the ultimate goal of this function is to distinguish between the statistical data types of features. To achieve this goal, we propose a solution which combines the simple logic approach and the Bayesian method. The datasets to be discovered are acquired from OpenML [65]. It is noticeable that, even though we limit our tool to supervised learning tasks, for this function, our approach can be applied to any dataset from OpenML.

We already know that the Bayesian method has proved to be able to discover statistical data types accurately [64]. Thus we decide to integrate the work of Valera [64] into our tool. However, apart from the raw dataset itself, the Bayesian model also needs some extra dataset information as the input such as MetaTypes which indicate whether a feature is continuous or discrete. Consequently, our solution is to divide the task into two steps. In the first step, we discover the basic data types (integer, float) of a feature using the simple logic approach. Then in step 2, we extract the dataset information required by the Bayesian model and apply the Bayesian method to discover the statistical data types. To provide an overview, we describe the workflow in Figure 4.1.

We first detect the basic data types of the features by applying some simple logic rules as follows:

- Bool: exactly 2 unique values in a feature
- Date: max 10 characters, contains '-' or '/' symbol
- Integer: try convert to integer with the *int*() function
- Float: try convert to integer and check for . symbol
- String: everything that remains

Then on this basis, we further determine whether a feature is discrete or continuous by checking the number of unique values:

- Discrete: limited number of unique values
- Continuous: unlimited number of unique values

Finally, we apply the Bayesian model to distinguish between the statistical data types. We make it more specific next.

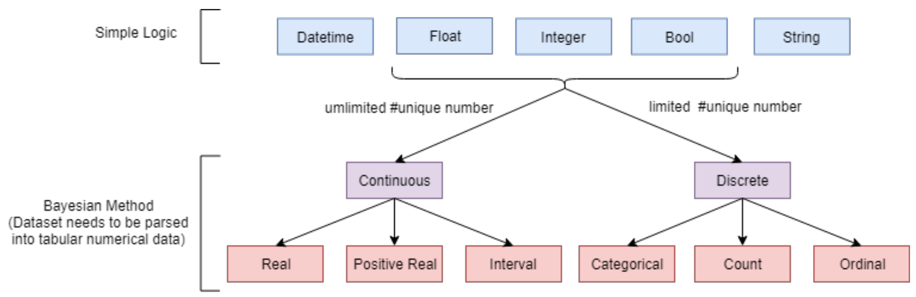


Figure 4.1: Workflow of discovering the statistical data types

4.1.1 Discover Basic Data Types

There are already many libraries such as Pandas which can be used to detect a general data type for each feature automatically. Unfortunately, missing values and outliers in the raw dataset will affect the judgment of Pandas. In practice, we can see many features are classified as 'object' by Pandas. And we can never promise that there are no problems in the dataset since data tend to come dirty. Therefore, we need an approach which is more robust to missing values and outliers.

Inspired by Brute Force Guessing and Anjelo's work [42], we check the type of every element in the feature and the type of the entire feature is determined through a majority vote. As an example, imagine there are ten elements in a feature, and nine elements are detected as the float type and one as the string type. In this case, the data type of this feature will be considered as float. It is worth noting that some datasets may be massive, in which case it would take a very long time to detect the type of every single element. Hence, every time we only take a sample of the given feature (10 percent in our tool) and perform Brute Force Guessing on the sample. The sample and majority vote mechanism weaken the effect of the missing values and outliers to some extent. The data types to be inferred in this step are date, bool, integer, float, and string. Next, we elaborate on how to detect every one of these types.

Starting from the most basic type bool, we first compute the number of unique values in the sample before checking the type of every elements. If there are exactly two unique values, the feature will be assigned the bool type directly. As we can see, there is a priority on these data types. If a feature is determined as bool, we do not further detect if it is float or string.

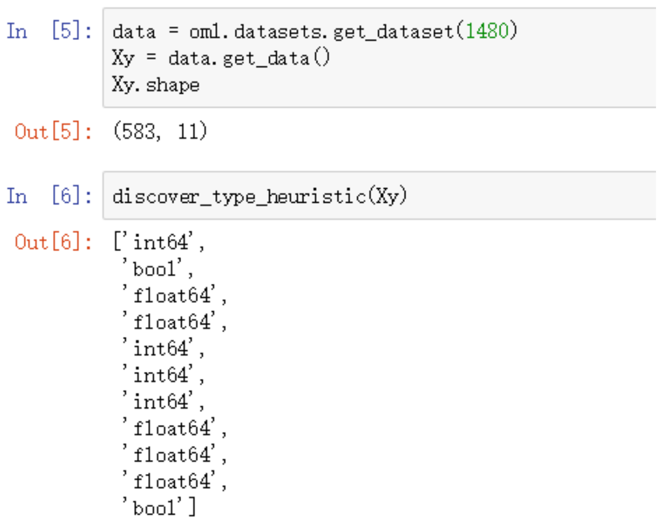
The next data type to be inferred is date. Dates can be encoded in various notations. To be regarded as a date, the elements need to satisfy the following two rules:

- A value only has a maximum length of ten characters.
- A value belongs to one of the four patterns: $xx/xx/xxxx$, $xxxx/xx/xx$, $xx - xx - xxxx$ and $xxxx - xx - xx$ where x stands for a number.

Floats and integers are both numeric types and are easy to be detected. We can simply try to convert the element to a integer with the Python built-in function *int*(). If this conversion succeeds, we know that this should be a numeric type. We continue to check the occurrence of the floating point to distinguish between integers and floats.

Finally, when an element is not either a bool, a date, an integer or a float, it will be assigned the string type.

As mentioned previously, the datasets acquired from OpenML have already been parsed into tabular numerical data mostly, hence actually only float, integer and and bool are used, as shown in Figure 4.2.

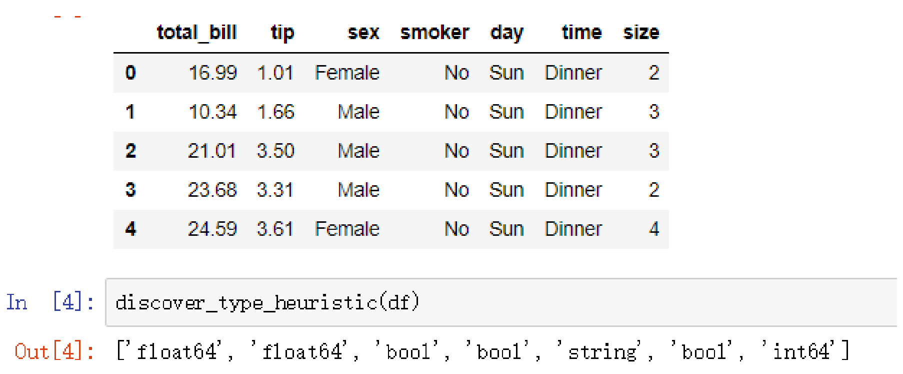


```
In [5]: data = oml.datasets.get_dataset(1480)
        Xy = data.get_data()
        Xy.shape

Out[5]: (583, 11)

In [6]: discover_type_heuristic(Xy)

Out[6]: ['int64',
         'bool',
         'float64',
         'float64',
         'int64',
         'int64',
         'int64',
         'float64',
         'float64',
         'float64',
         'bool']
```

Figure 4.2: Discover basic data types for a parsed OpenML dataset

However, this heuristic implementation makes it possible to detect data types for unparsed datasets as well. Figure 4.3 shows we detect the data types of an unparsed csv format dataset using the heuristic method.

Based on the results of these basic data types, we can further detect the statistical data types.



	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

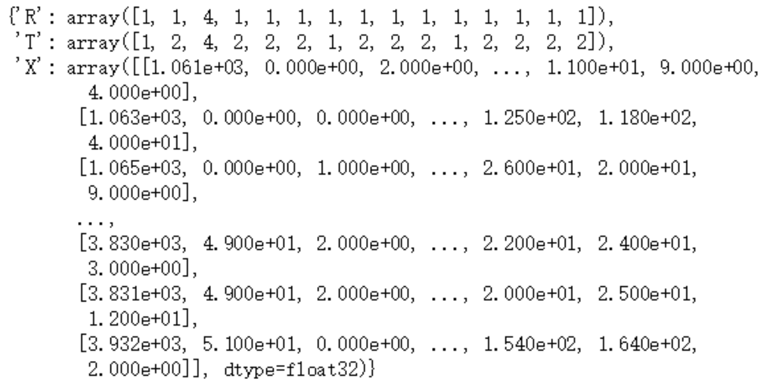
```
In [4]: discover_type_heuristic(df)

Out[4]: ['float64', 'float64', 'bool', 'bool', 'string', 'bool', 'int64']
```

Figure 4.3: Discover basic data types for an unparsed csv format dataset

4.1.2 Discover Statistical Data Types

We implement the Bayesian method based on Valera's work. The Bayesian model asks for three kinds of information as shown in Figure 4.4:

- X: Parsed dataset
- T: MetaTypes of each feature
- R: Cardinality



```
{'R': array([1, 1, 4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]),
 'T': array([1, 2, 4, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 2]),
 'X': array([[1.061e+03, 0.000e+00, 2.000e+00, ..., 1.100e+01, 9.000e+00,
         4.000e+00],
        [1.063e+03, 0.000e+00, 0.000e+00, ..., 1.250e+02, 1.180e+02,
         4.000e+01],
        [1.065e+03, 0.000e+00, 1.000e+00, ..., 2.600e+01, 2.000e+01,
         9.000e+00],
        ...,
        [3.830e+03, 4.900e+01, 2.000e+00, ..., 2.200e+01, 2.400e+01,
         3.000e+00],
        [3.831e+03, 4.900e+01, 2.000e+00, ..., 2.000e+01, 2.500e+01,
         1.200e+01],
        [3.932e+03, 5.100e+01, 0.000e+00, ..., 1.540e+02, 1.640e+02,
         2.000e+00]], dtype=float32)}
```

Figure 4.4: Input of the Bayesian model

For the parsed dataset, as we know, raw datasets usually contain much redundant information. Some information such as name, date, notes may not be that useful in the data analysis. Hence this kind of information can be dropped in the analytical process. Besides, many discrete variables may be encoded in text form, for example, "never", "sometimes", "usually". In order to train the machine learning model, these values need to be transformed to the numerical type. Fortunately, we can easily acquire the parsed datasets from OpenML and input them to the Bayesian model.

MetaType basically is used to indicate whether a feature is continuous or discrete. There are four types of MetaTypes:

- MetaType 1: positive real-valued, real-valued, interval
- MetaType 2: real-valued, interval
- MetaType 3: binary
- MetaType 4: categorical, ordinal, count

There are some overlaps between MetaTypes, for example, MetaType 1 covers MetaType 2. This means MetaType 1 is more general than MetaType 2 and more possible data types will be considered. The MetaType aims to limit the number of data types to guess. As mentioned in Section 3.2, each feature gets a likelihood model which is a mixture of likelihood functions. The likelihood function varies according to the data type. Hence if we know a feature is MetaType 2, the likelihood model of this feature will be represented as the mixture of real-valued likelihood function and interval likelihood function, which makes the model more efficient than infer from all the statistical data types. Hence, it is safe to assign MetaType 1 to a feature which is actually MetaType 2, but not vice versa. The important thing is to distinguish between MetaType 1, 2 and MetaType 3, 4. It is obvious that MetaType 1, 2 are continuous and Metatype 3 4 are discrete. Hence, the primary task is to determine whether a feature is continuous or discrete.

We already have the basic data types of the features: bool, integer and float (for numerical data). Bool is certainly the discrete type. However, it is a little tricky for integers and floats. At the beginning, we directly classified integers as the discrete type and floats as the continuous type. However, the detection accuracy turns out to be so disappointing that sometimes the data type predictions are even completely wrong for OpenML datasets. The problem is that many integers are encoded as floats in the parsed dataset, for example, 1.0, 2.0. We can fix this by calling the Python built-in function *is_integer* to check if a float is actually an integer.

However, this is still not enough. We notice that sometimes, a float variable may only take a very limited number of unique values. For example, it may only take values from 0.5, 1.5 and 2.5. So in this case, this float variable should be considered as the discrete type. On the other hand, it is not safe to classify integers as discrete either. The reason is that sometimes the integer variables we observed are actually the result of truncation. As an example, the variable income may take values 1650, 1349, 3432, 2000 and so on. In this case, it can take any value within a certain range such as 1890.5.

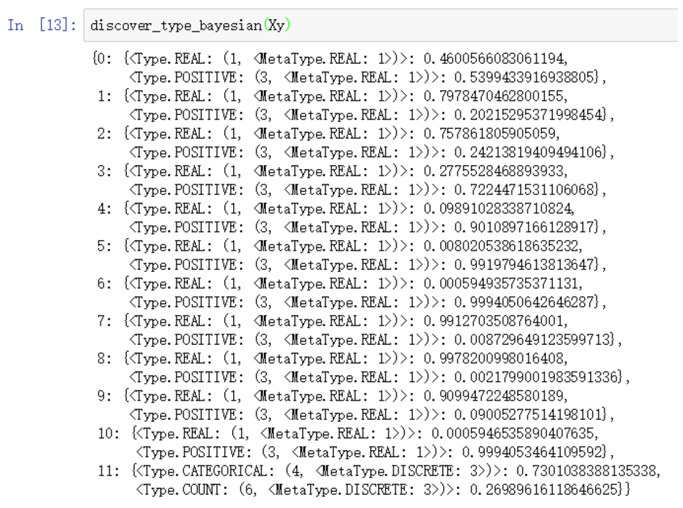
Therefore, instead of simply classify integers as discrete and floats as continuous, we apply some simple logic rules to distinguish between the continuous and the discrete types.

We count the number of unique values that the feature takes and the number of instances of the feature. A feature with less than 10 unique values will be directly regarded as the discrete type. If a feature has more than 10 unique values but less than 2% of number of instances, it will be considered as discrete too. The percentage 2% is subjective and can be modified according to the specific application. This implementation may seem ad hoc, but in order to automate this process, we can only consider the most general cases. And according to our experiments, it works well mostly.

However, this requirement may still be too loose. Imagine a dataset has 1,000,000 instances and 2% unique values, which still leaves 20,000 unique values. It is barely possible for a discrete feature to have so many categories. After inspecting the most-run datasets on OpenML, we notice that the discrete features usually have less than 100 unique values. Hence we set 100 as the upper bound of the number of unique values a discrete feature can have. To summarize, if a feature satisfies either of the following requirements, it will be considered as discrete, otherwise continuous.

- The feature has less than 10 unique values.
- The number of unique values is less than 100 and not more than 2% of the number of instances.

After finding the discrete variables, the next step is to determine the cardinality. In mathematics, the cardinality of a set denotes the number of elements of the set. Here it is slightly different. If we set the number of unique values as the cardinality of a discrete feature, there would be errors running the Bayesian model for some OpneML datasets. With the help of the author Antonio Vergari [66] who implements the Bayesian model, we figured that we should actually consider the maximal value of the discrete variable. The reason is that generally the acquired dataset is finite, and we may only observe a part of all the possible values. As an example, consider the current samples for a certain feature: $[0, 1, 1, 90, 2, 0]$. The number of unique values of this sample is 4 but we can clearly see that the domain stretches up to 90. Besides, for continuous variables, the cardinality is set to 1.



```
In [13]: discover_type_bayesian(Xy)

{0: {<Type.REAL: (1, <MetaType.REAL: 1>)>: 0.4600566083061194,
     <Type.POSITIVE: (3, <MetaType.REAL: 1>)>: 0.5399433916938805},
 1: {<Type.REAL: (1, <MetaType.REAL: 1>)>: 0.7978470462800155,
     <Type.POSITIVE: (3, <MetaType.REAL: 1>)>: 0.20215295371998454},
 2: {<Type.REAL: (1, <MetaType.REAL: 1>)>: 0.757861805905059,
     <Type.POSITIVE: (3, <MetaType.REAL: 1>)>: 0.24213819409494106},
 3: {<Type.REAL: (1, <MetaType.REAL: 1>)>: 0.2775528468893933,
     <Type.POSITIVE: (3, <MetaType.REAL: 1>)>: 0.7224471531106068},
 4: {<Type.REAL: (1, <MetaType.REAL: 1>)>: 0.09891028338710824,
     <Type.POSITIVE: (3, <MetaType.REAL: 1>)>: 0.9010897166128917},
 5: {<Type.REAL: (1, <MetaType.REAL: 1>)>: 0.008020538618635232,
     <Type.POSITIVE: (3, <MetaType.REAL: 1>)>: 0.9919794613813647},
 6: {<Type.REAL: (1, <MetaType.REAL: 1>)>: 0.000594935735371131,
     <Type.POSITIVE: (3, <MetaType.REAL: 1>)>: 0.9994050642646287},
 7: {<Type.REAL: (1, <MetaType.REAL: 1>)>: 0.9912703508764001,
     <Type.POSITIVE: (3, <MetaType.REAL: 1>)>: 0.008729649123599713},
 8: {<Type.REAL: (1, <MetaType.REAL: 1>)>: 0.9978200998016408,
     <Type.POSITIVE: (3, <MetaType.REAL: 1>)>: 0.0021799001983591336},
 9: {<Type.REAL: (1, <MetaType.REAL: 1>)>: 0.9099472248580189,
     <Type.POSITIVE: (3, <MetaType.REAL: 1>)>: 0.09005277514198101},
 10: {<Type.REAL: (1, <MetaType.REAL: 1>)>: 0.0005946535890407635,
      <Type.POSITIVE: (3, <MetaType.REAL: 1>)>: 0.9994053464109592},
 11: {<Type.CATEGORICAL: (4, <MetaType.DISCRETE: 3>)>: 0.7301038388135338,
      <Type.COUNT: (6, <MetaType.DISCRETE: 3>)>: 0.26989616118646625}}
```

Figure 4.5: Discover statistical data types using the Bayesian model

Now we have all the necessary information that the Bayesian model needs and we input them to the Bayesian model and the statistical data types can be discovered, as shown in Figure 4.5. The type with the higher weight will be considered as the type of the feature.

4.1.3 Results

To evaluate the performance of our approach, we detect the data types for the 20 most-run datasets from OpenML. These datasets are frequently explored and have various feature data types. Since some of the data types provided by OpenML may not be accurate, we manually label the ground truth for each dataset used for evaluation. In addition, the Bayesian model needs a pre-set number of iterations, hence we compare the performance after running different iterations.

We compute the accuracy by comparing the number of features whose types are predicted correctly with the total number of features. For the record, the implemented Bayesian model is still naive in the current stage, and the ordinal and interval types remain to be further extended. Hence, we label the ordinal data as the category for now. The interval data are labeled as real-valued or positive real-valued depending on whether there are negative data in it. Considering that the iterations of running the Bayesian model may affect the result, we run the Bayesian model for 1 iteration, 5 iterations, and 10 iterations respectively to see if there is any difference. The evaluation results are summarized in Table 4.1. We also visualize the results in the bar chart as in Figure 4.6.

Dataset ID	Accuracy_1	Accuracy_5	Accuracy_10	Features Checked
31	0.524	0.857	0.809	21
1464	1.0	1.0	1.0	5
334	0.0	0.428	0.571	7
50	0.0	0.4	0.7	10
333	0.0	0.143	0.429	7
1494	0.547	0.285	0.309	42
3	0.081	0.568	0.703	37
1510	0.838	0.806	0.838	31
1489	0.5	1.0	1.0	6
37	1.0	1.0	1.0	9
1479	0.990	0.990	0.990	101
1487	0.931	0.931	0.876	73
1063	0.909	0.909	0.863	22
1471	1.0	1.0	1.0	15
1467	0.905	0.905	0.619	21
1480	0.545	0.909	1.0	11
1068	1.0	0.955	0.955	22
1492	0.984	0.984	0.984	65
1050	0.684	0.684	0.71	28
1462	0.4	0.4	0.8	5

Table 4.1: Results of statistical data type discovery after running Bayesian model for 1, 5 and 10 iterations

As we can see from Table 4.1, the performance of Bayesian model is not very ideal after 1 iteration. The predictions of data types for datasets 334, 50, 333 are even completely wrong. Fortunately, after 5 iterations, Bayesian model achieves a decent performance overall. Compared with the accuracy after 1 iteration, the accuracy after 5 iterations has been greatly improved. The result achieves the best after 10 iterations, but the improvement is not that significant compared with that after 5 iterations. We compute the mean accuracy after 1, 5 and 10 iterations respectively and a bar chart is used to determine the relationship between accuracy and number of iterations as shown in Figure 4.7.

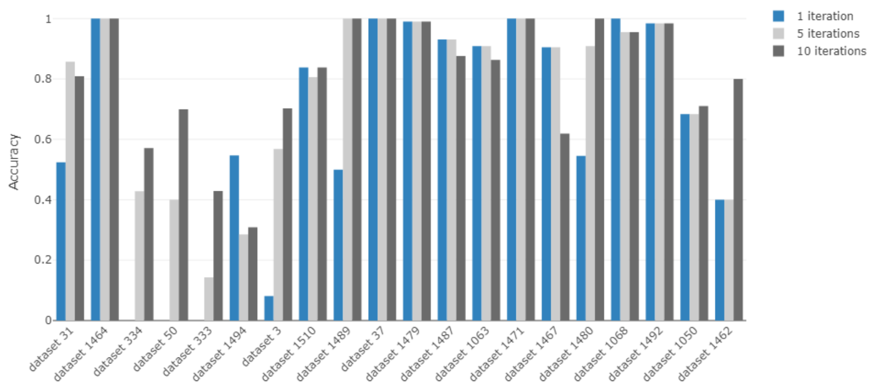


Figure 4.6: Results of statistical data type discovery after running Bayesian model for 1, 5 and 10 iterations

As we can see from Figure 4.7, a positive correlation is found between the accuracy and the number of iterations. However, there are also exceptions. For example, the accuracy for the dataset 1494 decreases with the increase of the number of iterations. We inspect the dataset 1494 and we find that this dataset contains a lot of count type features. The Bayesian model tends to take count type as the categorical type when the number of iterations are increased. Hence, it is not a good idea to run the Bayesian model for too many iterations. Moreover, it also takes more time to run the program with more iterations. Generally speaking, the running time of the Bayesian model is acceptable. Among the evaluated datasets, it takes at most 6 minutes to discover the data types of a dataset.

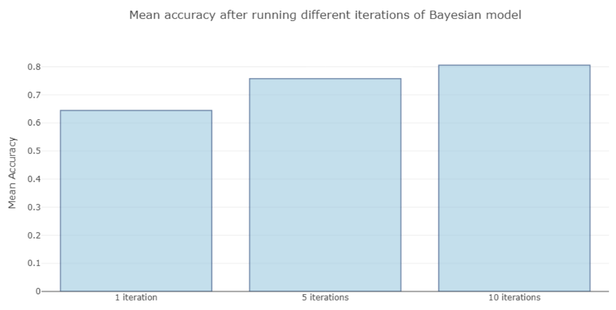


Figure 4.7: Mean accuracy with respect to different number of iterations

4.2 Automatic Missing Value Handling

To handle the missing values, we first have to detect them and then select the appropriate approach to clean them. As mentioned in Section 3.3, there is no algorithm always superior to others. The performance of an algorithm is closely related to the missing mechanism [5]. Hence we divide this task into three subtasks: detect missing values, identify missing mechanisms, and clean missing values.

To provide an overview, we describe the workflow in Figure 4.8. We start from detecting the missing values. Then we present them in effective visualizations to help the user understand the missing mechanism. Afterward, we evaluate the candidate approaches for the given mechanism and recommend the optimal approach to the user.

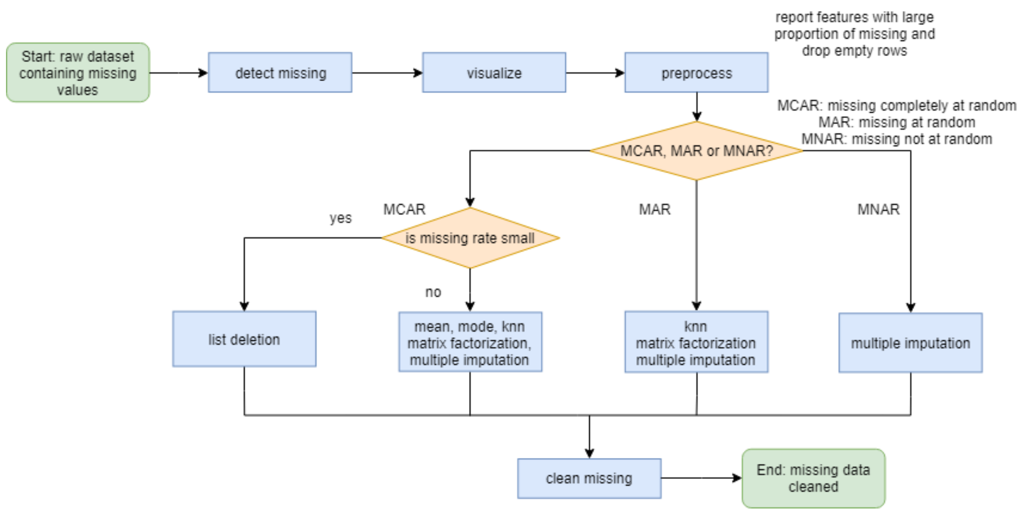
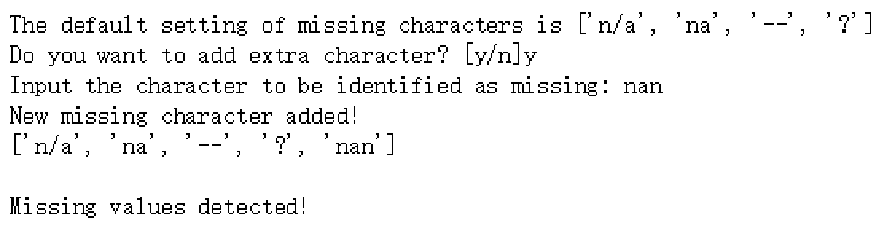

Start: raw dataset containing missing values
detect missing
visualize
preprocess
report features with large proportion of missing and drop empty rows
MCAR: missing completely at random
MAR: missing at random
MNAR: missing not at random
MCAR, MAR or MNAR?
MCAR
MAR
MNAR
yes
is missing rate small
no
list deletion
mean, mode, knn matrix factorization, multiple imputation
knn matrix factorization multiple imputation
multiple imputation
clean missing
End: missing data cleaned

Figure 4.8: Workflow of dealing with missing values

4.2.1 Detect Missing Values

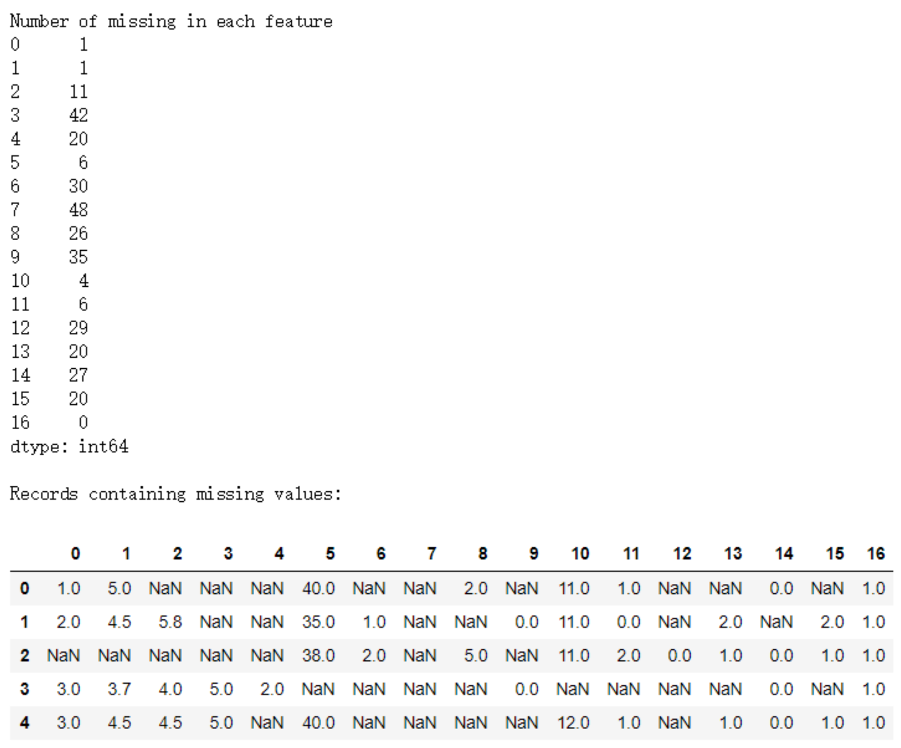
Missing values may appear in the dataset as different formats such as 'na' and '?'. In some datasets, missing values may even be encoded as 0. We cannot take 0 as missing for every dataset. Thus it would be better to detect the missing values in an interactive manner. To be more specific, apart from the most common missing characters such as 'na', we ask the user whether to add any other specific value to be identified as missing every time before the detection. We show this process in Figure 4.9.



```
The default setting of missing characters is ['n/a', 'na', '--', '?']
Do you want to add extra character? [y/n]y
Input the character to be identified as missing: nan
New missing character added!
['n/a', 'na', '--', '?', 'nan']

Missing values detected!
```

Figure 4.9: Detect missing values in an interactive manner

The missing information of the number of missing values in each feature and the records containing missing values will be shown as in Figure 4.10.



```
Number of missing in each feature
0      1
1      1
2     11
3     42
4     20
5      6
6     30
7     48
8     26
9     35
10     4
11     6
12    29
13    20
14    27
15    20
16     0
dtype: int64

Records containing missing values:
```

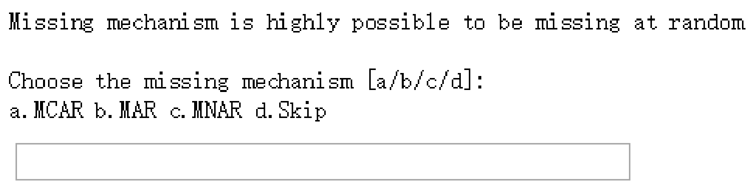
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	1.0	5.0	NaN	NaN	NaN	40.0	NaN	NaN	2.0	NaN	11.0	1.0	NaN	NaN	0.0	NaN	1.0
1	2.0	4.5	5.8	NaN	NaN	35.0	1.0	NaN	NaN	0.0	11.0	0.0	NaN	2.0	NaN	2.0	1.0
2	NaN	NaN	NaN	NaN	NaN	38.0	2.0	NaN	5.0	NaN	11.0	2.0	0.0	1.0	0.0	1.0	1.0
3	3.0	3.7	4.0	5.0	2.0	NaN	NaN	NaN	NaN	0.0	NaN	NaN	NaN	NaN	0.0	NaN	1.0
4	3.0	4.5	4.5	5.0	NaN	40.0	NaN	NaN	NaN	NaN	12.0	1.0	NaN	1.0	0.0	1.0	1.0

Figure 4.10: Information of missing data

This kind of information seems a bit rough and not that helpful for users to understand the data. In the next section, we will introduce some visualizations which present the missing data in a more fancy way and able to assist users in understanding the data adequately.

4.2.2 Identify Missing Mechanisms

We already know that there are three types of missing mechanisms: MCAR, MAR and MNAR. Unfortunately, the MNAR assumption is not testable since the information that is needed for such a test is missing. We may have reasons to suspect that the probability of missingness depends on the values that are missing, for example, people with high incomes may be less likely to report their incomes. But nothing in the data will tell you whether this is the case or not [5]. For MCAR and MAR, we can detect whether there is some correlation of a value being missed in a feature and the value of any other of the features. And if there is, we inform the user that the missing mechanism is possibly MAR, otherwise MCAR. For the record, the test result is not definite. Even if we do not find any correlation between two features, it is still possible to be MAR since a value may be missing as a function of many other features. We only provide information and the user has to make the decision, as shown in Figure 4.11.

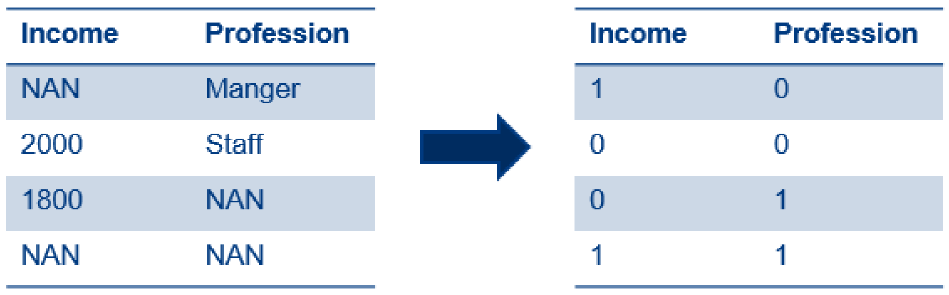


```
Missing mechanism is highly possible to be missing at random

Choose the missing mechanism [a/b/c/d]:
a.MCAR b.MAR c.MNAR d.Skip
```

Figure 4.11: User chooses the missing mechanism

Compute Missing Correlation

To help users make the decision, we first detect whether there is some missing dependency between the features. For this purpose, we compute the Pearson correlation coefficient (PCC) [70]. PCC is a measure of the linear correlation between two variables. We denote the occurrence of the missing values as 1 and absence as 0, then each pair of features can be represented by a series of coordinates as shown in Figure 4.12 (coordinates: (0, 0), (0, 1), (1, 0), (1, 1)). If there is a missing dependency between these two features, then (0, 0) and (1, 1) should be frequently appear and a linear correlation will be detected.



Income	Profession
NAN	Manger
2000	Staff
1800	NAN
NAN	NAN

Income	Profession
1	0
0	0
0	1
1	1

Figure 4.12: Occurrence of missing values denoted as 1, otherwise 0

The PCC is defined as follows:

$$\rho_{X,Y} = \frac{\operatorname{cov}(X,Y)}{\sigma_X \sigma_Y}$$

where:

$\cdot$ cov is the covariance

$\cdot$ σ_X is the standard deviation of X

$\cdot$ σ_Y is the standard deviation of Y

This correlation is symmetric, i.e., $\rho_{X,Y} = \rho_{Y,X}$, since $\operatorname{cov}(X,Y) = \operatorname{cov}(Y,X)$. The higher the correlation, the stronger the missing dependency. When the PCC between two features is higher than 0.8, the tool will prompt the user that the missing mechanism is probably MAR.

Visualize Missing Values

Another effort we made to help users understand the missing data is to present the missingness in effective visualizations. We take advantage of a powerful missing data visualization library missingno [3]. We provide the following missing data visualizations: bar chart, matrix and heatmap.

Bar chart: Bar chart is the most basic visualization. It presents the missingness by showing the number of observed records for each feature, as shown in Figure 4.13. Thus users can quickly see which features contain missing values.

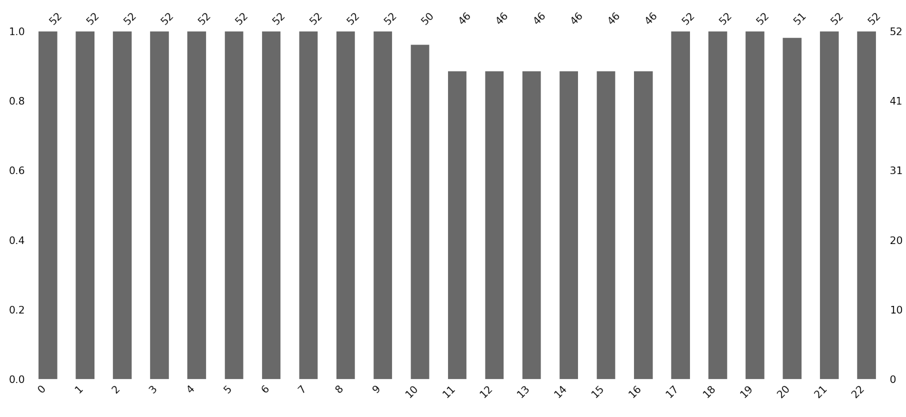
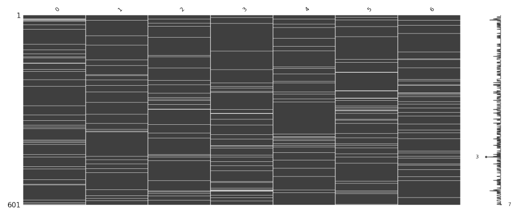
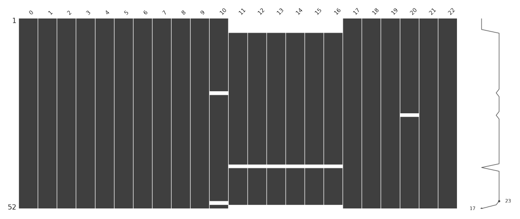


Figure 4.13: Visualize missing data with bar chart

Matrix: Matrix basically does the same thing as bar chart. It shows the completeness of a dataset by a density display. Compared with bar chart, matrix is more useful for identifying the missing patterns. As an example, we compare the matrix of two datasets as shown in Figure 4.14(a) and Figure 4.14(b). We can clearly see that the missingness in Figure 4.14(a) is totally random. On the other side, in Figure 4.14(a), it is apparent that there exists some missing correlation in feature 11 to feature 16. Whenever one of them is missing, the others are always missing as well. The matrix pattern can help users understand the data and give clues about the missing mechanism. For example, Figure 4.14(b) tends to be MCAR while Figure 4.14(a) is more probable to be MAR. Besides, the sparkline at right summarizes the general shape of the data completeness and points out the maximum and minimum rows.



(a) Matrix: no obvious feature dependency



(b) Matrix: significant feature dependency

Figure 4.14: Visualize missing data with matrix

Heatmap: Heatmap is specially aimed at visualizing the missing correlation between two features both containing missing values: how strongly the presence or absence of one variable affects the presence of another. The missing correlation is computed by the Pearson Correlation Coefficient (PCC) as we discussed previously. Since PCC is symmetric, the heatmap is symmetric too. The correlation ranges from -1 to 1. When correlation is -1, it means if one variable appears (not missing) the other definitely does not. When correlation is 0, the presence or absence of one variable has no effect on the other. If correlation is 1, then as long as one variable appears the other definitely also does. Features without missing values will be neglected in this heatmap.

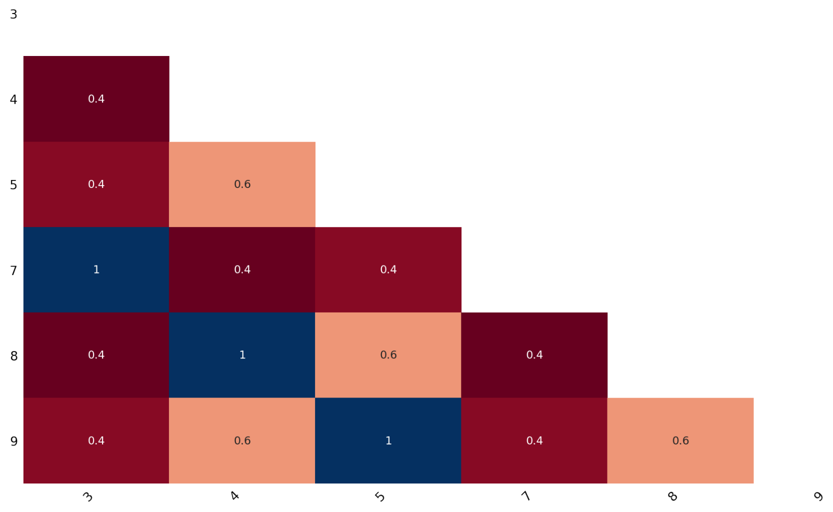


Figure 4.15: Visualize missing data with heatmap

The visualizations are aimed to help users understand data better such that users can infer the missing mechanism by themselves. The missing mechanism decides how the tool recommends the missing cleaning approach.

4.2.3 Clean Missing Values

Drop Redundant Information

After detecting the missing values in the dataset, sometimes we may observe that the values of some records or feature are largely missing. In this situation, these features or records do not provide any information for training the machine learning model. Hence, before choosing the specific approach to deal with the missing values, we first preprocess the dataset to remove the useless information. We directly drop the empty records as they are meaningless. For the features with a substantial proportion of missing values, we detect them and report them to the users. The reason we do not delete them directly is that sometimes these features may be important and have a significant effect on the result of classification.

Candidate Approaches for Each Missing Mechanism

The various approaches may be confusing to a non-expert user. Consequently, it is important that the tool can recommend the optimal approach to the user. Different approaches are suitable for different missing mechanisms. In this thesis, the following approaches are considered: list deletion, mean, mode, k nearest neighbor, matrix factorization, and multiple imputation. We implement these approaches using scikit learn and fancyimpute [4]. Based on the literature study in Section 3.3, we summarize the candidate approaches for each mechanism as follows:

- MCAR: list deletion, mean, mode, k nearest neighbor, matrix factorization, multiple imputation.
- MAR: k nearest neighbor, matrix factorization, multiple imputation.
- MNAR: multiple imputation

MCAR: MCAR is not usually the case, but if MCAR is a reasonable assumption, then there are a lot of convenient methods for handling missing data. All the methods provided in this thesis can be considered. For the list deletion approach, we further detect the missing percentage. We will only recommend the list deletion if the missing percentage is very low. If there are too many records containing missing values, list deletion should be avoided. Otherwise too much information would be dropped.

MAR: Most research papers assume MAR. In this case, there are strong correlations between features. Consequently, statistical methods mean and mode should not be used since they are just making up data without considering the feature dependency.

MNAR: MNAR is the most difficult case to handle. Technically speaking, data should be cleaned manually using deductive methods in this case. As an example, if we observed that someone has two children in year 2014, NA children in year 2015, and two children in year 2016, we can probably impute that they have two children in 2015. This deductive imputation normally requires context. However, since we are trying to automate the cleaning process, we take the multiple imputation as the only candidate because it can still achieve a decent performance even in MNAR [47].

Recommend the Approach

To recommend an approach, we first have to predict the performance of the candidate approaches. Strictly speaking, the performance of an imputation approach should be computed by comparing the imputed values and the ground truth. However, acquiring the ground truth of missing values is not realistic in practice. Moreover, it is difficult to find that many real datasets with different types of missing mechanisms. Most papers evaluate the imputation methods by applying a classifier after the data has been completed to see if the classifier performance has been improved [59]. List deletion usually produces better performance than imputation methods in this kind of evaluation [59]. Consider the extreme case that a dataset only has one record that does not contain missing values, after list deletion, only one record remains. And there is no need to perform classification at this point. Apparently it is not reasonable. Hence list deletion are not considered in this case.

To predict the performance of imputation methods, our approach is to apply some simple classifiers after the imputation and compute mean accuracy as the score of the approach. The following simple classifiers are used:

- Naive Bayes Learner: Naive Bayes Learner is a probabilistic classifier, based on Bayes' Theorem:

  $$p(X|Y) = \frac{p(Y|X) \cdot p(X)}{p(Y)}$$

  where $p(X)$ is the prior probability and $p(X|Y)$ is the posterior probability. It is called naive, because it assumes independence of all attributes to each other.

- Linear Discriminant Learner: Linear Discriminant Learner is a type of discriminant analysis, which is understood as the grouping and separation of categories according to specific features. Linear discriminant is basically finding a linear combination of features that separates the classes best. The resulting separation model is a line, a plane, or a hyperplane, depending on the number of features combined.

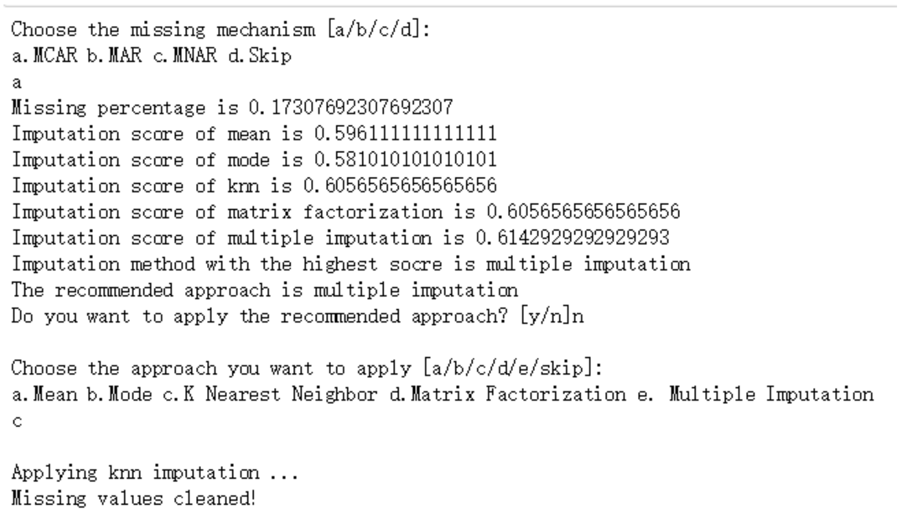
- One Nearest Neighbor Learner: One Nearest Neighbor learner is a classifier based on instance-based learning, which means instead of performing explicit generalization, it compares new problem instances with instances already seen in training. A test point is assigned to the class of the nearest point within the training set.

- Decision Node Learner: Decision Node Learner is a classifier based on the information gain of attributes. The information gain indicates how informative an attribute is with respect to the classification task using its entropy. The higher the variability of the attribute values, the higher its information gain. This learner selects the attribute with the highest information gain. Then, it creates a single node decision tree consisting of the chosen attribute as a split node.

- Randomly Chosen Node Learner: Randomly Chosen Node Learner is a classifier that results also in a single decision node, based on a randomly chosen attribute.

These simple learners are often used to compute landmarking meta-features for describing a dataset, which we will discuss in the outlier detection section later.

Instead of cleaning data with the approach with the highest score, we show the scores of each candidate approach and recommend the approach with the highest score to the user. The user can decide whether to adopt the recommendation or apply other approaches. Figure 4.16 shows the interactive process of cleaning missing values.



```
Choose the missing mechanism [a/b/c/d]:
a.MCAR b.MAR c.MNAR d.Skip
a
Missing percentage is 0.17307692307692307
Imputation score of mean is 0.5961111111111111
Imputation score of mode is 0.581010101010101
Imputation score of knn is 0.6056565656565656
Imputation score of matrix factorization is 0.6056565656565656
Imputation score of multiple imputation is 0.6142929292929293
Imputation method with the highest socre is multiple imputation
The recommended approach is multiple imputation
Do you want to apply the recommended approach? [y/n]n

Choose the approach you want to apply [a/b/c/d/e/skip]:
a.Mean b.Mode c.K Nearest Neighbor d.Matrix Factorization e. Multiple Imputation
c

Applying knn imputation ...
Missing values cleaned!
```

Figure 4.16: Clean missing data interactively

4.3 Automatic Outlier Detection

One-dimensional outliers can be detected through standard deviation easily. While for multi-dimensional outliers, there are more available approaches. As we discussed in Section 3.4, isolation forest (iForest), local outlier factor (LOF) and one class support vector machine (OCSVM) are all feasible algorithms. However, the performance of these algorithms may vary a lot on different datasets, and there is no algorithm uniformly better than all the others. Consequently, it is difficult for a non-expert to decide which algorithm to use. Our strategy is to leverage the idea of meta-learning [53, 21] which is a technique for predicting the performance of an algorithms on a given dataset. For a new dataset, we first describe it by meta-features. Next we apply the trained meta-learner to recommend the optimal outlier detection algorithm for the given dataset. Then we detect outliers and report them to the user. Finally we ask the user whether to drop the outliers. The workflow is described in Figure 4.17.

In this section, we first explain how meta-learning works for outlier detection. After that we describe the benchmarking of outlier detection algorithms and present the results. Then we demonstrate how we train the recommendation model. Finally, we illustrate how we present the outliers to users.

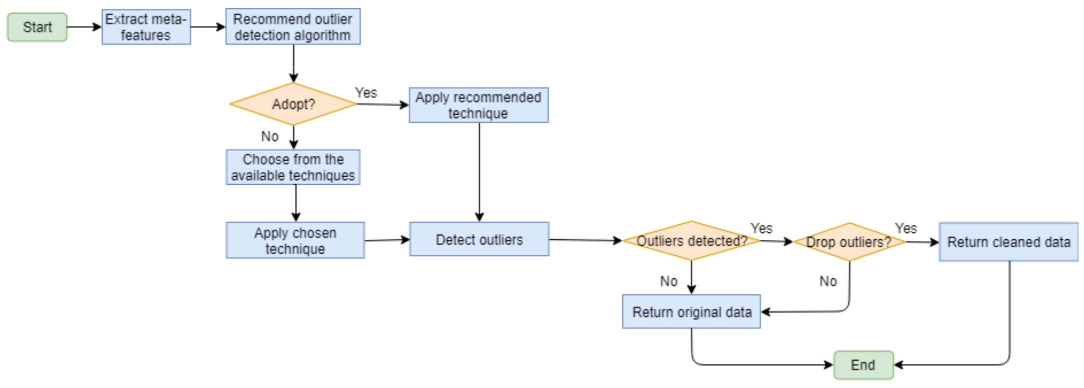


Figure 4.17: Workflow of outlier detection

4.3.1 Meta-learning for Outlier Detection

Experiment Design

Meta-learning aims to find the relationships between dataset characteristics and algorithms [53]. In meta-learning, datasets are described as meta-features. These meta-features along with the performance of the target algorithm form a training record which will be used to train the meta-learning model. This process is described in Figure 4.18.

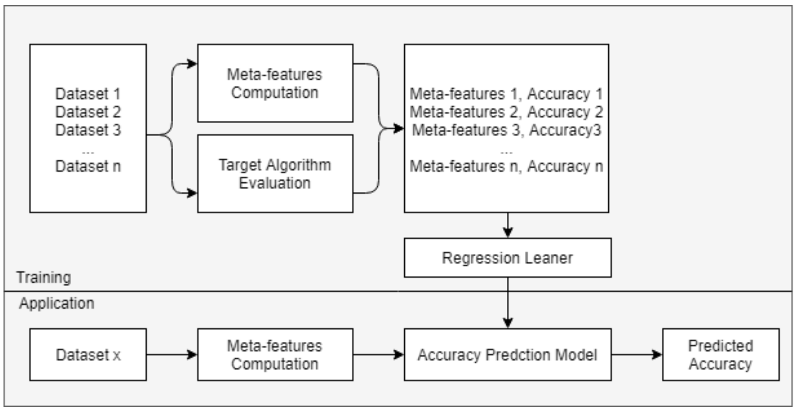

Dataset 1
Dataset 2
Dataset 3
...
Dataset n
Meta-features Computation
Target Algorithm Evaluation
Meta-features 1, Accuracy 1
Meta-features 2, Accuracy 2
Meta-features 3, Accuracy3
...
Meta-features n, Accuracy n
Regression Leaner
Training
Application
Dataset x
Meta-features Computation
Accuracy Predction Model
Predicted Accuracy

Figure 4.18: Meta-learning for predicting the performance of an algorithm on a given dataset

As we can see, the target algorithm is performed on each training dataset and the performance metric accuracy is computed. Then each pair of meta-features and the accuracy is treated as a training record, and a regression learner is trained on these records. Hence, when a new dataset comes, we only have to compute the meta-features of this new dataset and use the trained regression learner to predict the accuracy of the target algorithm on this dataset. Thus in our case, we can predict the performance of iForest, LOF and OCSVM on the new dataset respectively and then recommend the outlier detection algorithm with the best performance to the user. However, it is noticeable that the predicted performance may not be that accurate and these biases add together may weaken the best algorithm prediction badly. Consider that the true score of iForest, LOF and OCSVM on a random dataset is 0.4, 0.5 and 0.6 respectively and the best algorithm is OCSVM in this case. While the predicted score is 0.52, 0.45 and 0.5, in which case the predicted best algorithm is iForest. As we can see, although the predicted score are all close to their true score, these small biases together may lead to the worst algorithm. Besides, this method is not robust either, once the prediction of one algorithm has a big bias, the prediction of the best algorithm will be considerably affected.

In fact, we do not really need to know the performance of each outlier detection algorithm on the new dataset. We only care about which algorithm is the optimal. Hence, we made a improvement on the current design. We first evaluate all the candidate algorithms on the training datasets. Then instead of pairing the meta-features and the performance to get three regression learners (iForest, LOF, OCSVM), we pair the meta-features with the optimal algorithm, as shown in Figure 4.19. After that we train a classifier which can predict the best algorithm for a given dataset directly.

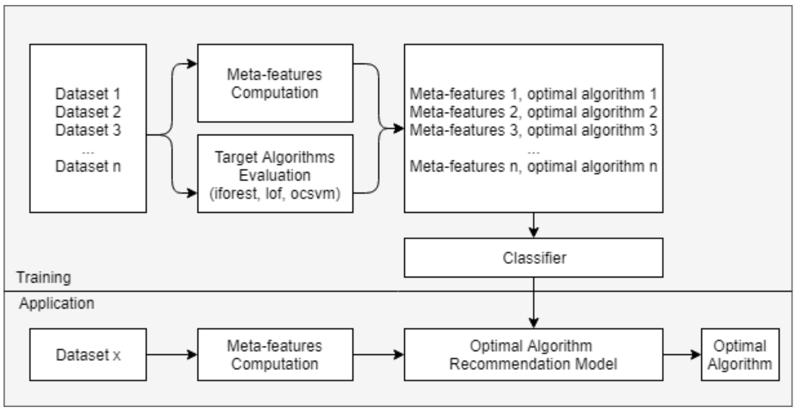

Dataset 1
Dataset 2
Dataset 3
...
Dataset n
Meta-features Computation
Target Algorithms Evaluation (iforest, lof, ocsvm)
Meta-features 1, optimal algorithm 1
Meta-features 2, optimal algorithm 2
Meta-features 3, optimal algorithm 3
...
Meta-features n, optimal algorithm n
Classifier
Training
Application
Dataset x
Meta-features Computation
Optimal Algorithm Recommendation Model
Optimal Algorithm

Figure 4.19: Meta-learning for predicting the optimal outlier detection algorithm on a given dataset

Metric Selection

It is essential to evaluate the outlier detection algorithms with the proper performance metric. We should be aware that datasets are usually imbalanced since outliers only take a small part. Consequently, the metric accuracy should not be used. Before determining the metric, we first review the terms in binary classification and explain the meanings in the outlier detection context.

- True Positive (TP): Outliers predicted as outliers.
- False Positive (FP): Normal data predicted as outliers.
- False Negative (FN): Outliers predicted as normal data.
- True Negative (TN): Normal data predicted as normal data.
- Precision: Precision is used when the goal is to limit FPs. In our case, precision represents the ratio of correctly predicted outliers to the total predicted outliers.

$$Precision = \frac{TP}{TP + FP}$$

- Recall: Recall is used when the goal is to limit FNs. In our case, recall represents the ratio of correctly predicted outliers to all the actual outliers.

$$Recall = \frac{TP}{TP + FN}$$

Apparently we want the precision and recall both to be high, hence we finally take f1-score as the metric.

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

F1-score trades off precision and recall and is robust when the data are imbalanced.

Meta-features selection

Most meta-learning systems are aimed for the classification task, thus most of the proposed meta-features describe datasets with known class labels [21].Meta-features can be divided into several categories [10]:

- Simple: Simple meta-features are easily accessible from the given dataset, such as number of attributes, number of classes and dataset dimensionality.
- Statistical: Dataset characteristics are computed by statistical approaches such as linear correlation coefficient, skewness, kurtosis and standard deviation.
- Information theoretic: These meta-features class label and entropy measures of attributes such as normalize attribute entropy and mutual information and signal to noise ratio.
- Model based: Model based meta-features are based on the assumption that data can be modeled in a decision tree structure. Different properties of this tree are used as meta-features such as number of leaves, number of nodes and node per attribute.
- Landmarking: Landmarking meta-features are computed by simple and significant machine learning algorithms. They include one nearest neighbor learner, decision node, naive bayes, linear discriminant, worst node and random node.

There are so many meta-features, and obviously, we cannot use all of them. We need to select out the effective meta-features with less computational cost. Feuer et al. [53] empirically evaluated all these five categories of meta-features and found that landmarking meta-features can achieve nearly the same performance with using all the meta-features above, but with times less computational effort. Therefore, we adopt the accuracy of the following five simple learners along with their running time to describe the datasets: **one nearest neighbor**, **decision node**, **naive bayes**, **linear discriminant** and **random node**. We already described them in Section 4.2.3.

Moreover, remember that we are evaluating the unsupervised learning algorithms iForest, LOF, OCSVM (the dataset represents a supervised classification problem but which data are outliers are unknown), we add three clustering metrics of the unsupervised learning algorithm K-means as meta-features to describe the dataset: Silhouette Coefficient, Calinski-Harabaz Index and Davies-Bouldin Index. We select these three clustering metrics because they do not need the ground truth of outliers. Other clustering metrics such as purity and mutual information based score require the ground truth, while we do not have advanced knowledge of which are the outliers in a new dataset. We further describe the three metrics next.

Silhouette Coefficient [54]: A higher Silhouette Coefficient score relates to a model with better defined clusters. Silhouette Coefficient is defined for each sample and is composed of two scores:

- a: The mean distance between a sample and all other points in the same class.
- b: The mean distance between a sample and all other points in the next nearest cluster.

The Silhouette Coefficient s for a single sample is then given as:

$$s = \frac{b-a}{max(a,b)}$$

The Silhouette Coefficient for a set of samples is given as the mean of the Silhouette Coefficient for each sample.

Calinski-Harabaz Index [38]: A higher Calinski-Harabaz score relates to a model with better defined clusters. For k clusters, the Calinski-Harabaz score s is given as the ratio of the between-clusters dispersion mean and the within-cluster dispersion:

$$s(k) = \frac{\operatorname{Tr}(B_k)}{\operatorname{Tr}(W_k)} \times \frac{N-k}{k-1}$$

where N is the number of points in the data, B_k is the between group dispersion matrix and W_k is the within-cluster dispersion matrix.

Davies-Bouldin Index [18]: A lower Davies-Bouldin index relates to a model with better separation between the clusters. The index is defined as the average similarity between each cluster C_i for $i = 1, \ldots, k$ and its most similar one C_j. In the context of this index, similarity is defined as a measure R_{ij} that trades off:

- s_i: the average distance between each point of cluster i and the centroid of that cluster.
- d_{ij}: the distance between cluster centroids i and j.

Then the Davies-Bouldin index is defined as:

$$DB = \frac{1}{k} \sum i = 1^k \max_{i \neq j} R_{ij}$$

The above are all the meta-features we are going to use to describe the dataset.

4.3.2 Benchmarking of Outlier Detection Algorithms

Now we have determined the metrics to evaluate algorithms and the meta-features to describe the datasets, we can start the benchmarking of the outlier detection algorithms.

Benchmarking Datasets

It is not easy to find that many datasets with the ground truth of outliers for benchmarking. Our solution is using the highly imbalanced datasets from OpenML and treat the minority class as outliers. We select the highly imbalanced datasets by reference to ODDS (Outlier Detection Datasets) which is a website recommending the datasets suitable for outlier detection. For each benchmarking dataset, we map the minority class label to 1 (outlier) and the other classes to 0 (normal).

Benchmarking Setting

All the three outlier detection algorithms have an important parameter (iForest: contamination, LOF: contamination, OCSVM: nu) which indicates the contamination of the outliers in the dataset and this parameter has a great effect on f1-score, as can be seen in Figure 4.20.

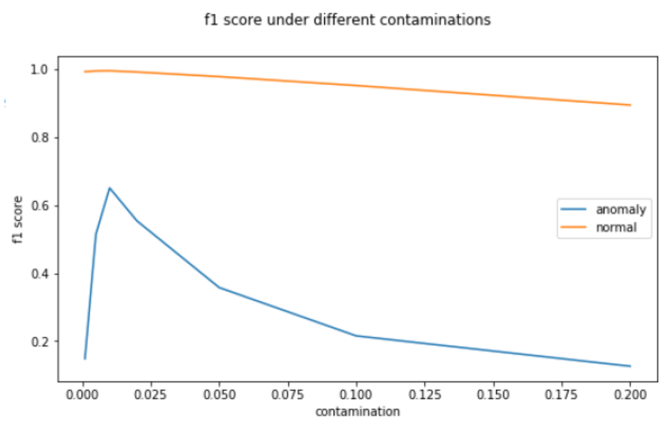
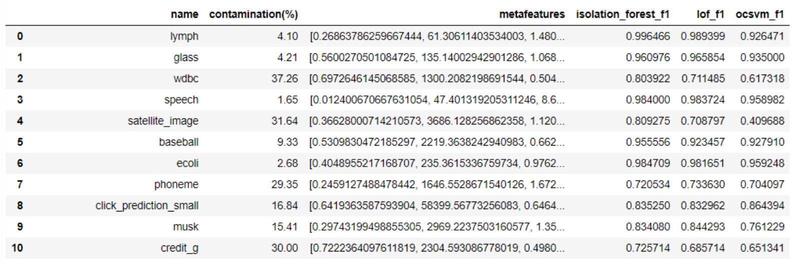


Figure 4.20: The effect of parameter contamination on f1-score

The f1-score reaches the highest when the parameter contamination is set at the actual outlier percentage. Hence, for the sake of fairness, we set this parameter as the actual contamination value during the evaluation. For a random unseen dataset, we do not know the outlier percentage in advance. We will explain how to estimate the outlier percentage for a new dataset later.

Benchmarking Results

After the benchmarking, we get the meta-features for each dataset and corresponding f1-score of each outlier detection algorithm. We show a part of the results in Figure 4.21.



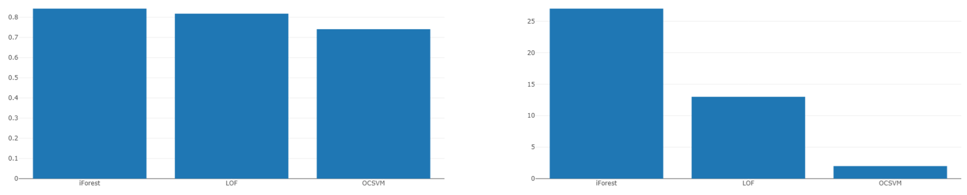
	name	contamination(%)	metafeatures	isolation_forest_f1	lof_f1	ocsvm_f1
0	lymph	4.10	[0.26863786259667444, 61.30611403534003, 1.480...	0.996466	0.989399	0.926471
1	glass	4.21	[0.5600270501084725, 135.14002942901286, 1.068...	0.960976	0.965854	0.935000
2	wdbc	37.26	[0.6972646145068585, 1300.2082198691544, 0.504...	0.803922	0.711485	0.617318
3	speech	1.65	[0.012400670667631054, 47.401319205311246, 8.6...	0.984000	0.983724	0.958982
4	satellite_image	31.64	[0.36628000714210573, 3686.128256862358, 1.120...	0.809275	0.708797	0.409688
5	baseball	9.33	[0.5309830472185297, 2219.3638242940983, 0.662...	0.955556	0.923457	0.927910
6	ecoli	2.68	[0.4048955217168707, 235.3615336759734, 0.9762...	0.984709	0.981651	0.959248
7	phoneme	29.35	[0.2459127488478442, 1646.5528671540126, 1.672...	0.720534	0.733630	0.704097
8	click_prediction_small	16.84	[0.6419363587593904, 58399.56773256083, 0.6464...	0.835250	0.832962	0.864394
9	musk	15.41	[0.29743199498855305, 2969.2237503160577, 1.35...	0.834080	0.844293	0.761229
10	credit_g	30.00	[0.722236409761819, 2304.593086778019, 0.4980...	0.725714	0.685714	0.651341

Figure 4.21: Part of benchmarking results

We briefly compare the three outlier detection algorithms. We compute the mean f1-score and the count of being the optimal algorithm for each of them, and the results are summarized in Table 4.2 and visualized in Figure 4.22.

Outlier Detection Algorithm	iForest	LOF	OCSVM
Count of being the optimal algorithm	27	13	2
Mean f1-score	0.843	0.818	0.741

Table 4.2: Comparison between iForest, LOF and OCSVM



(a) Mean f1-score

(b) Count of being the optimal algorithm

Figure 4.22: Comparison between iForest, LOF and OCSVM

As we can see, even though their mean f1-scores are very close, the count of being the optimal varies a lot. IForest performs mostly better than the other two algorithms. OCSVM, however, only performs best 2 of 32 datasets. Considering that the datasets are collected randomly, we conclude that iForest generally performs best.

Moreover, we also wonder which meta-features contribute more useful information. Hence we build the random forest model and compute the important features which are shown in Figure 4.23. It can be seen that naive_bayes_times has the highest importance. The time of running a machine learning algorithm is related to many factors such as the number of instances, number of features and dataset complexity. Moreover, we can observe that the three clustering metrics (Silhouette Coefficient, Calinski-Harabaz Index and Davies-Bouldin Index) rank second, third and fifth, which are higher than most other meta-features. They are indeed informative as we expected.

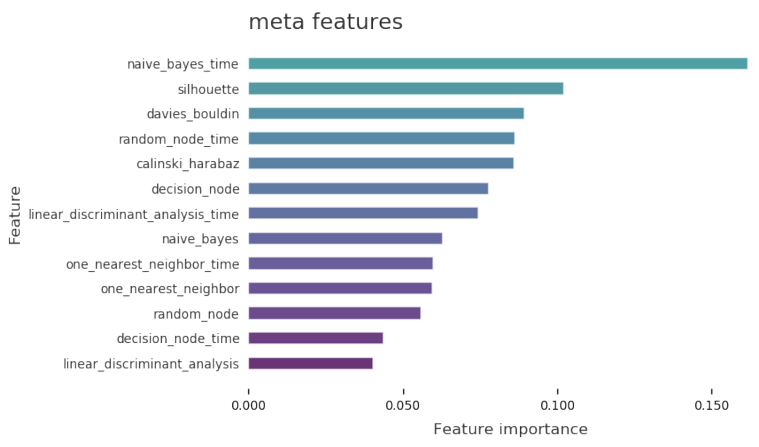


Figure 4.23: Important meta-features

4.3.3 Recommendation Model

With the benchmarking results, we can now train the meta-learners (classifiers) to recommend the optimal outlier detection algorithm based on the meta-features of the dataset. The following meta-learners are considered:

- Random forest: Random forests creates decision trees on randomly selected data samples, gets prediction from each tree and selects the best solution by means of voting. It can also compute the feature importance as we saw previously.
- Support vector machine (SVM): Support vector machine aims to find hyperplane that separates the examples of each class. When dealing with a non-linear classification, SVM can create a non-linear boundary by projecting data to the high-dimensional space using what is called the kernel trick.
- K nearest neighbor: The principle behind k nearest neighbor methods is to find k training samples closest in distance to the new point, and predict the label from these samples.

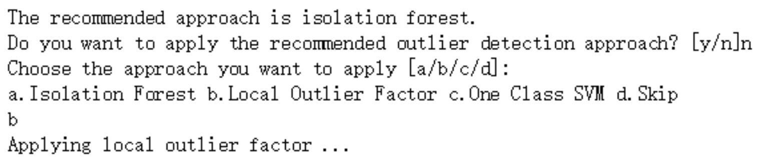
We tune the hyperparameters of these meta-learners by the grid search and save the model with the best score. We summarize the best performance of these three meta-learners in Table 4.3.

Meta-learner	**Accuracy**
Random Forest	0.718
Support Vector Machine	0.656
K Nearest Neighbor	0.687

Table 4.3: Meta-learner best performance

We can see that the random forest has the highest performance, hence we save the learned random forest model as our recommendation model. Now, for an unseen raw dataset, we can compute the meta-features of this dataset and predict the optimal outlier detection algorithm using our recommendation model. The recommended approach can be used to identify multi-dimensional outliers. There still remains one problem, we do not know the outlier percentage of this new dataset. Our strategy is to estimate the outlier percentage through the results of one-dimensional outlier detection. We estimate the outlier percentage by the ratio of the number of records containing one-dimensional outliers and the number of records.

We recommend the optimal algorithm to the user and user can decide whether to adopt these approach, as shown in Figure 4.24.



```
The recommended approach is isolation forest.
Do you want to apply the recommended outlier detection approach? [y/n]n
Choose the approach you want to apply [a/b/c/d]:
a.Isolation Forest b.Local Outlier Factor c.One Class SVM d.Skip
b
Applying local outlier factor ...
```

Figure 4.24: User chooses the outlier detection algorithm

We then apply the algorithm decided by the user to detect the outliers. As outliers are not equal to bad data, we ask the user whether to drop outliers or not.

4.3.4 Visualize Outliers

We present the outliers to users in multiple visualizations.

Box Plot: The one-dimensional outliers are presented to users by box plot, as shown in Figure 4.25. The outliers are represented by the black dots outside the box. The distribution of data is clearly shown.

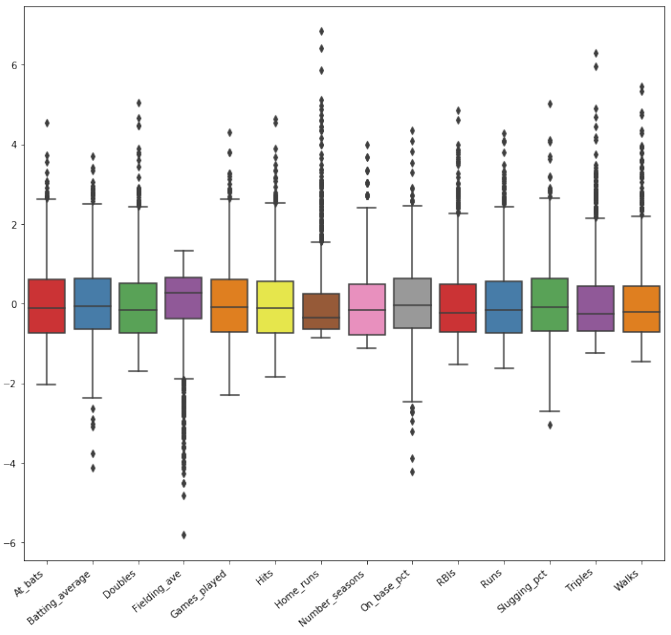
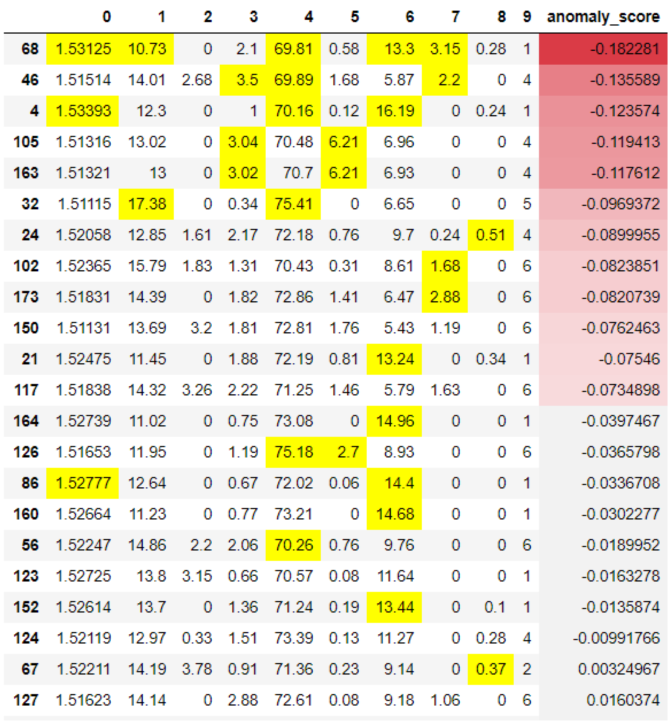


Figure 4.25: Visualize one-dimensional outliers by box plot

Styled DataFrame: The visualization of multi-dimensional outliers is a bit more tricky since we are visualizing high-dimensional data. Our first solution is the styled Pandas DataFrame, as shown in Figure 4.26. We compute the anomaly score of each record and concatenate it to the original DataFrame. Then we rank the DataFrame by the anomaly score and combine it with the heatmap. The red color indicates the anomaly score of each record. The redder, the more anomaly. We also highlight the one-dimensional outliers in yellow as can be seen from Figure 4.26.



	0	1	2	3	4	5	6	7	8	9	anomaly_score
68	1.53125	10.73	0	2.1	69.81	0.58	13.3	3.15	0.28	1	-0.182281
46	1.51514	14.01	2.68	3.5	69.89	1.68	5.87	2.2	0	4	-0.135589
4	1.53393	12.3	0	1	70.16	0.12	16.19	0	0.24	1	-0.123574
105	1.51316	13.02	0	3.04	70.48	6.21	6.96	0	0	4	-0.119413
163	1.51321	13	0	3.02	70.7	6.21	6.93	0	0	4	-0.117612
32	1.51115	17.38	0	0.34	75.41	0	6.65	0	0	5	-0.0969372
24	1.52058	12.85	1.61	2.17	72.18	0.76	9.7	0.24	0.51	4	-0.0899955
102	1.52365	15.79	1.83	1.31	70.43	0.31	8.61	1.68	0	6	-0.0823851
173	1.51831	14.39	0	1.82	72.86	1.41	6.47	2.88	0	6	-0.0820739
150	1.51131	13.69	3.2	1.81	72.81	1.76	5.43	1.19	0	6	-0.0762463
21	1.52475	11.45	0	1.88	72.19	0.81	13.24	0	0.34	1	-0.07546
117	1.51838	14.32	3.26	2.22	71.25	1.46	5.79	1.63	0	6	-0.0734898
164	1.52739	11.02	0	0.75	73.08	0	14.96	0	0	1	-0.0397467
126	1.51653	11.95	0	1.19	75.18	2.7	8.93	0	0	6	-0.0365798
86	1.52777	12.64	0	0.67	72.02	0.06	14.4	0	0	1	-0.0336708
160	1.52664	11.23	0	0.77	73.21	0	14.68	0	0	1	-0.0302277
56	1.52247	14.86	2.2	2.06	70.26	0.76	9.76	0	0	6	-0.0189952
123	1.52725	13.8	3.15	0.66	70.57	0.08	11.64	0	0	1	-0.0163278
152	1.52614	13.7	0	1.36	71.24	0.19	13.44	0	0.1	1	-0.0135874
124	1.52119	12.97	0.33	1.51	73.39	0.13	11.27	0	0.28	4	-0.00991766
67	1.52211	14.19	3.78	0.91	71.36	0.23	9.14	0	0.37	2	0.00324967
127	1.51623	14.14	0	2.88	72.61	0.08	9.18	1.06	0	6	0.0160374

Figure 4.26: Visualize outliers by styled DataFrame

Parallel Coordinates: As discussed in Section 3.5, parallel coordinates is a common way of visualizing high-dimensional data. The outliers can be distinguished as they are isolated or with different slopes than its neighbors. We also color the normal data and outliers respectively such that they can be more easily distinguished.

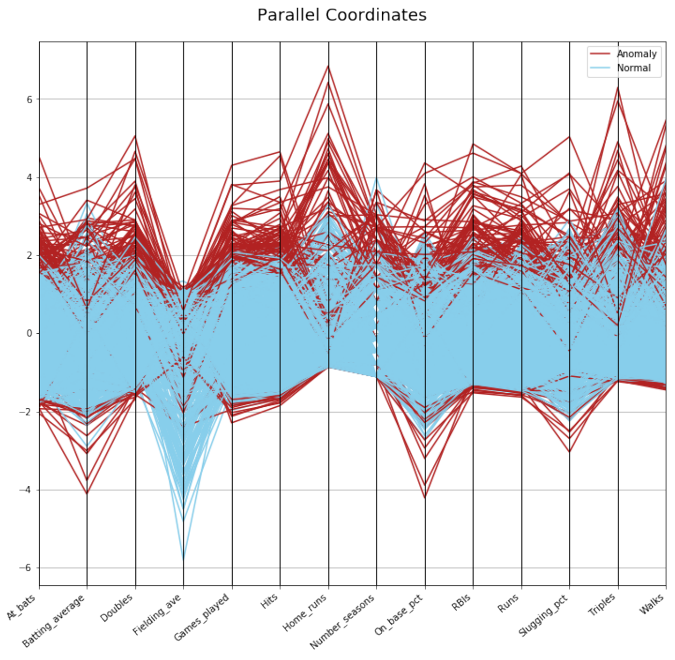


Figure 4.27: Visualize outliers by parallel coordinates plot

Scatter Plot and Bar Chart: The last visualization is by combining the scatter plot and the bar chart. We select out two features most likely to have outliers and then plot them as shown in Figure 4.28.

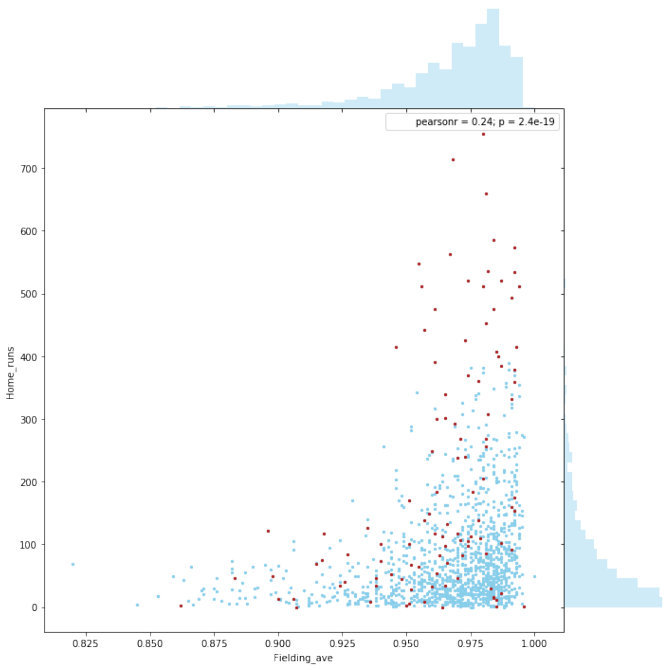


Figure 4.28: Visualize outliers by combination of scatter plot and bar chart

4.4 Other Capabilities

There are some other capabilities of this data cleaning tool but they are not the focus in this thesis. Hence we briefly present them in this section.

4.4.1 Show Important Features

The tool computes the most important features of the given dataset using random forest and present the 15 most useful features to the user, as shown in Figure 4.29.

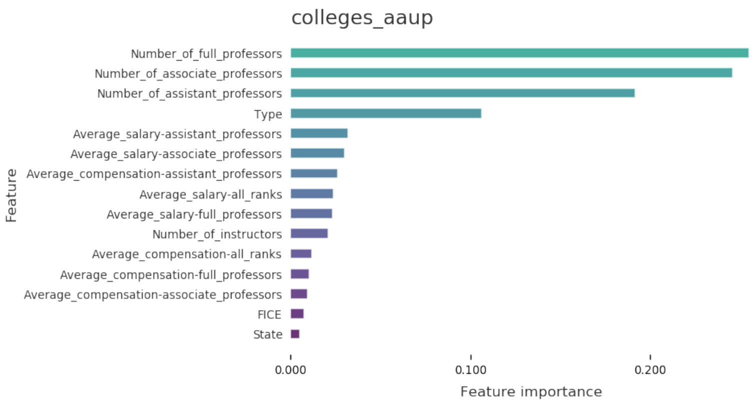
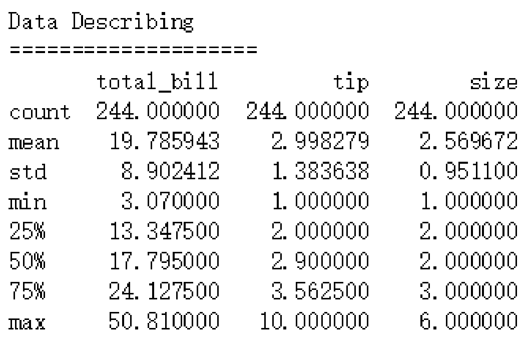


Figure 4.29: Show important features

4.4.2 Show Statistical Information

To help users gain a better understanding of the data distribution, statistical information are presented as in Figure 4.30.

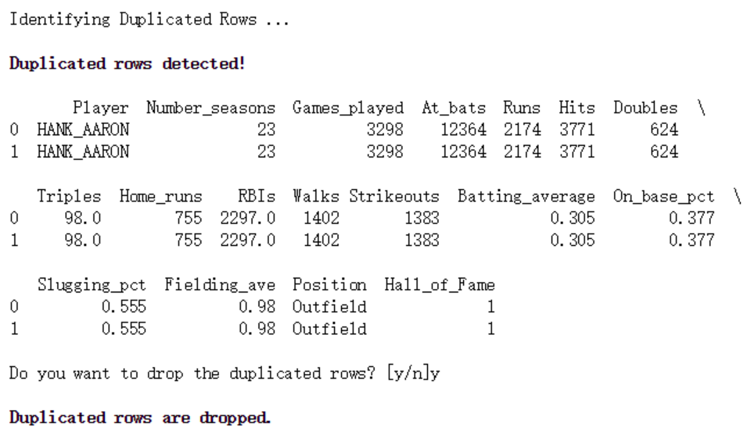


```
Data Describing
====================
       total_bill         tip        size
count  244.000000  244.000000  244.000000
mean    19.785943    2.998279    2.569672
std      8.902412    1.383638    0.951100
min      3.070000    1.000000    1.000000
25%     13.347500    2.000000    2.000000
50%     17.795000    2.900000    2.000000
75%     24.127500    3.562500    3.000000
max     50.810000   10.000000    6.000000
```

Figure 4.30: Show statistical information

4.4.3 Detect Duplicated Records

The tool is capable of detecting the duplicated records in the data and report them to users. Users can decide whether to drop these duplicated records. Figure 4.31 shows this process.

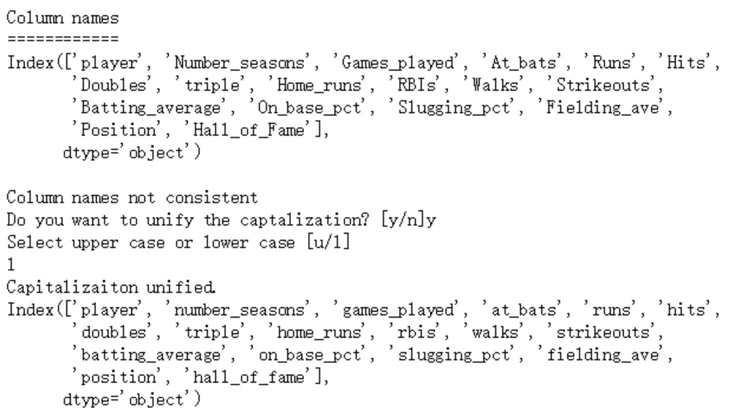


```
Identifying Duplicated Rows ...

Duplicated rows detected!

     Player  Number_seasons  Games_played  At_bats  Runs  Hits  Doubles  \
0  HANK_AARON            23          3298    12364  2174  3771      624
1  HANK_AARON            23          3298    12364  2174  3771      624

   Triples  Home_runs    RBIs  Walks Strikeouts  Batting_average  On_base_pct  \
0     98.0        755  2297.0   1402       1383            0.305        0.377
1     98.0        755  2297.0   1402       1383            0.305        0.377

   Slugging_pct  Fielding_ave  Position  Hall_of_Fame
0         0.555          0.98  Outfield             1
1         0.555          0.98  Outfield             1

Do you want to drop the duplicated rows? [y/n]y

Duplicated rows are dropped.
```

Figure 4.31: Detect duplicated records

4.4.4 Unify Inconsistent Capitalization

Inconsistent capitalization of column names can also be detected and reported to users. Users can decide whether to unify them or not. The capitalization can be unified to either upper case or lower case, as shown in Figure 4.32.



```
Column names
============
Index(['player', 'Number_seasons', 'Games_played', 'At_bats', 'Runs', 'Hits',
       'Doubles', 'triple', 'Home_runs', 'RBIs', 'Walks', 'Strikeouts',
       'Batting_average', 'On_base_pct', 'Slugging_pct', 'Fielding_ave',
       'Position', 'Hall_of_Fame'],
      dtype='object')

Column names not consistent
Do you want to unify the captalization? [y/n]y
Select upper case or lower case [u/l]
l
Capitalizaiton unified.
Index(['player', 'number_seasons', 'games_played', 'at_bats', 'runs', 'hits',
       'doubles', 'triple', 'home_runs', 'rbis', 'walks', 'strikeouts',
       'batting_average', 'on_base_pct', 'slugging_pct', 'fielding_ave',
       'position', 'hall_of_fame'],
      dtype='object')
```

Figure 4.32: Unify Inconsistent Capitalization

Chapter 5

Conclusions

In this thesis, we automated the process of data cleaning. We investigated the common data problems and the existing techniques to clean them. To support the study, a Python tool is developed to identify the potential issues in the data and report results and recommendations to the user. The tool is aimed for the machine learning field and the following aspects are meaningfully automated: data type discovery, missing value handling, and outlier detection.

5.1 Contributions

As mentioned in Section 3.6, existing data cleaning tools normally have the following limitations:

- They may only provide simple and plain data cleaning approaches.
- They may focus on addressing a single data problem.
- User assistance is not provided for approach selection.
- Most of them are not aimed at machine learning tasks.

Our data cleaning tool makes an improvement on these aspects. Instead of addressing one specific data issue, our tool covers a variety of data problems, including incorrect data types, missing values, outliers, duplicated records, and inconsistent column names. Advanced approaches are integrated such as the Bayesian method for discovering data types and multiple imputation for handling missing values. For each data problem, our tool evaluates the available approaches and recommends the optimal one based on the characteristics of the given dataset. The evaluation of approaches takes the performance of classifiers into account such that better machine learning models can be obtained after cleaning data. Besides, instead of cleaning the data problems, we also present the data in various visualizations to help the user understand the data better. Users can clean data smoothly with our tool. Moreover, our data cleaning tool is designed with a strong connection to OpenML which is a platform where people can easily share data, experiments and machine learning models. Users can easily inspect and clean the datasets from OpenML with the dataset ID using our tool.

For automatic discovery of data types, we introduced the approach to discover statistical data types which are more important for machine learning tasks. For automatic missing value handling, we examined the state-of-the-art techniques and summarized the different situations each approach can be applied according to the missing mechanism. For automatic outlier detection, we took advantage of meta-learning to select the optimal algorithm effectively.

5.2 Future Work

Data cleaning is a quite general task and there are so many aspects we can focus on, for example, the algorithm, the visualization or a specific data problem. Each aspect can be investigated individually as a project. Our data cleaning tool tries to cover as many data problems as possible, and consequently, there leaves much room for improvement on many aspects. We make it more specific next.

- **Automatic Data Type Discovery**: The tool can discover four statistical data types at the current stage: real-valued, positive real-valued, category, and count. There are still two types left out: ordinal and interval. The implemented Bayesian model can be extended by adding the likelihood functions of ordinal and interval. Also, the Bayesian model can be run at different settings of parameters, we can further automatically tune these parameters according to the characteristic of the dataset.
- **Automatic Missing Value Handling**: For the missing value detection, we predict the performance of an approach by evaluating it on some simple machine learning classifiers. However, this does not guarantee that the recommended approach is also optimal for user's classifier. We can interactively ask the user to input their classifier and directly evaluate the approaches on this user-specified classifier.
- **Automatic Outlier Detection**: We only considered iForest, LOF and OCSVM in our tool and there are more available outlier detection algorithms. Besides, the datasets we used for training our recommendation model is not really enough, and more datasets need to be used. There are also more possibilities for selecting the meta-features to describe the dataset. A further project can be performed to explore more meaningful meta-features for the outlier detection task. Moreover, we can also let the users choose to run multiple techniques and report all outliers detected by different techniques.
- **Visualization**: We provide various ways for users to visualize data while this may still be not enough considering the data are complex and high-dimensional. An interactive visualization is a better choice, the user can directly operate on the visualization to see what they want.
- **Machine Learning Tasks**: Our tool is limited to supervised classification while it can be extended to more kinds of tasks such as clustering. In that case, the evaluation of missing value imputation approaches should be adjusted to clustering algorithms such as k-means and DBSCAN. In addition, meta-features to describe the datasets should avoid using landmarkings which require the class label.

Bibliography

[1] Edgar Acuña and Caroline Rodriguez. On detection of outliers and their effect in supervised classification. *University of Puerto Rico at Mayaguez*, 2004. 15

[2] Edgar Acuna and Caroline Rodriguez. The treatment of missing values and its effect on classifier accuracy. In *Classification, clustering, and data mining applications*, pages 639–647. Springer, 2004. 10

[3] Aleksey Bilogur. Missingno: a missing data visualization suite, 2018. 35

[4] Alex Rubinsteyn, Sergey Feldman. fancyimpute: Version 0.0.16, May 2016. 37

[5] Paul D Allison. *Missing data*, volume 136. Sage publications, 2001. 11, 13, 32, 34

[6] Mennatallah Amer, Markus Goldstein, and Slim Abdennadher. Enhancing one-class support vector machines for unsupervised anomaly detection. In *Proceedings of the ACM SIGKDD Workshop on Outlier Detection and Description*, pages 8–15. ACM, 2013. 17

[7] Theodore W Anderson. Maximum likelihood estimates for a multivariate normal distribution when some observations are missing. *Journal of the american Statistical Association*, 52(278):200–203, 1957. 14

[8] Christophe Andrieu, Nando De Freitas, Arnaud Doucet, and Michael I Jordan. An introduction to mcmc for machine learning. *Machine learning*, 50(1-2):5–43, 2003. 10, 12

[9] Melissa J Azur, Elizabeth A Stuart, Constantine Frangakis, and Philip J Leaf. Multiple imputation by chained equations: what is it and how does it work? *International journal of methods in psychiatric research*, 20(1):40–49, 2011. 12

[10] Ashvini Balte, Nitin Pise, and Parag Kulkarni. Meta-learning with landmarking: A survey. *International Journal of Computer Applications*, 105(8), 2014. 42

[11] Ovunc Bozcan and Ayse Basar Bener. Handling missing attributes using matrix factorization. In *Realizing Artificial Intelligence Synergies in Software Engineering (RAISE), 2013 2nd International Workshop on*, pages 49–55. IEEE, 2013. 13

[12] Leo Breiman. Statistical modeling: The two cultures (with comments and a rejoinder by the author). *Statist. Sci.*, 16(3):199–231, 08 2001. 8

[13] Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. Lof: identifying density-based local outliers. In *ACM sigmod record*, volume 29, pages 93–104. ACM, 2000. 17

[14] George Casella and Roger L Berger. *Statistical inference*, volume 2. Duxbury Pacific Grove, CA, 2002. 15

[15] Wo-Ruo Chen, Yong-Huan Yun, Ming Wen, Hong-Mei Lu, Zhi-Min Zhang, and Yi-Zeng Liang. Representative subset selection and outlier detection via isolation forest. *Analytical Methods*, 8(39):7225–7231, 2016. ix, 19

[16] Software Testing Class. What is xml schema inference? xi, 9

[17] Sven F Crone, Stefan Lessmann, and Robert Stahlbock. The impact of preprocessing on data mining: An evaluation of classifier sensitivity in direct marketing. *European Journal of Operational Research*, 173(3):781–800, 2006. 7

[18] David L Davies and Donald W Bouldin. A cluster separation measure. *IEEE transactions on pattern analysis and machine intelligence*, pages 224–227, 1979. 43

[19] Iris Eekhout. *Don't Miss Out!: Incomplete data can contain valuable information.* Amsterdam: Vrije Universiteit, 2015. 12

[20] Craig K Enders. *Applied missing data analysis.* Guilford press, 2010. 12

[21] Andrey Filchenkov and Arseniy Pendryak. Datasets meta-feature description for recommending feature selection algorithm. In *Artificial Intelligence and Natural Language and Information Extraction, Social Media and Web Search FRUCT Conference (AINL-ISMW FRUCT), 2015*, pages 11–18. IEEE, 2015. 39, 42

[22] Caitlin Garrett. Data on the edge: Handling outliers. 19

[23] Markus Goldstein and Seiichi Uchida. A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data. *PloS one*, 11(4):e0152173, 2016. ix, 15, 16, 17

[24] Rolf HH Groenwold, Ian R White, A Rogier T Donders, James R Carpenter, Douglas G Altman, and Karel GM Moons. Missing covariate data in clinical research: when and when not to use the missing-indicator method for analysis. *Canadian Medical Association Journal*, 184(11):1265–1269, 2012. 14

[25] Frank E Grubbs. Procedures for detecting outlying observations in samples. *Technometrics*, 11(1):1–21, 1969. 8

[26] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *Journal of machine learning research*, 3(Mar):1157–1182, 2003. 7

[27] Yoel Haitovsky. Missing data in regression analysis. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 67–82, 1968. 12

[28] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009. 24

[29] Jiawei Han, Jian Pei, and Micheline Kamber. *Data mining: concepts and techniques.* Elsevier, 2011. 7, 8

[30] Julian Heinrich and Daniel Weiskopf. State of the art of parallel coordinates. In *Eurographics (STARs)*, pages 95–116, 2013. 23

[31] Mauricio A Hernández and Salvatore J Stolfo. Real-world data is dirty: Data cleansing and the merge/purge problem. *Data mining and knowledge discovery*, 2(1):9–37, 1998. 7

[32] Melissa Humphries. Missing data & how to deal: An overview of missing data. *Population Research Center. University of Texas. Recuperado de: http://www. google. com/url*, pages 39–41, 2013. 11

[33] Sean Kandel, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer. Wrangler: Interactive visual specification of data transformation scripts. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 3363–3372. ACM, 2011. 7

[34] Zakka Kevin. A complete guide to k-nearest-neighbors with applications in python and r, July 2016. 14

[35] Muzammil Khan and Sarwar Shah Khan. Data and information visualization methods, and interactive mechanisms: A survey. *International Journal of Computer Applications*, 34(1):1–14, 2011. ix, 20, 23

[36] Jörg-Uwe Kietz, Floarea Serban, Simon Fischer, and Abraham Bernstein. semantics inside! but lets not tell the data miners: Intelligent support for data mining. In *European Semantic Web Conference*, pages 706–720. Springer, 2014. 7

[37] Won Kim, Byoung-Ju Choi, Eui-Kyeong Hong, Soo-Kyung Kim, and Doheon Lee. A taxonomy of dirty data. *Data mining and knowledge discovery*, 7(1):81–99, 2003. 7

[38] Marcin Kozak. a dendrite method for cluster analysis by caliński and harabasz: A classical work that is far too often incorrectly cited. *Communications in Statistics-Theory and Methods*, 41(12):2279–2280, 2012. 42

[39] Open Knowledge Labs. Parsing for messy tables, 2016. 9

[40] Roderick JA Little and Donald B Rubin. *Statistical analysis with missing data*, volume 333. John Wiley & Sons, 2014. 10, 11

[41] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, pages 413–422. IEEE, 2008. 19

[42] Angelo Majoor. Auto-cleaning dirty data: the data encoding bot, 2017. 27

[43] Ms R Malarvizhi and Antony Selvadoss Thanamani. K-nearest neighbor in missing data imputation. *International Journal of Engineering Research and Development*, 5(1):5–7, 2012. 14

[44] Wes McKinney. pandas: a foundational python library for data analysis and statistics. *Python for High Performance and Scientific Computing*, pages 1–9, 2011. 24

[45] Wes McKinney. *Python for data analysis: Data wrangling with Pandas, NumPy, and IPython.* " O'Reilly Media, Inc.", 2012. 24

[46] M. Arthur Munson. A study on the importance of and time spent on different modeling steps. *SIGKDD Explor. Newsl.*, 13(2):65–71, May 2012. 7

[47] Michikazu Nakai and Weiming Ke. Review of the methods for handling missing data in longitudinal data analysis. *International Journal of Mathematical Analysis*, 5(1):1–13, 2011. 11, 38

[48] Nasser M Nasrabadi. Pattern recognition and machine learning. *Journal of electronic imaging*, 16(4):049901, 2007. 7, 17

[49] Marija J Norusis. *SPSS: Statistical data analysis.* SPSS, 1990. 23

[50] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011. 24

[51] Anne Helby Petersen and Claus Thorn Ekstrm. dataMaid: Your personal assistant for cleaning up the data cleaning process, 2017. 7, 25

[52] Therese D Pigott. A review of methods for missing data. *Educational research and evaluation*, 7(4):353–383, 2001. 11

[53] Matthias Reif, Faisal Shafait, Markus Goldstein, Thomas Breuel, and Andreas Dengel. Automatic classifier selection for non-experts. *Pattern Analysis and Applications*, 17(1):83–96, 2014. 39, 40, 42

[54] Peter J Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65, 1987. 42

[55] Donald B Rubin. Inference and missing data. *Biometrika*, 63(3):581–592, 1976. 10

[56] Donald B Rubin. *Multiple imputation for nonresponse in surveys*, volume 81. John Wiley & Sons, 2004. ix, 12, 13

[57] Bernhard Schölkopf, Robert C Williamson, Alex J Smola, John Shawe-Taylor, and John C Platt. Support vector method for novelty detection. In *Advances in neural information processing systems*, pages 582–588, 2000. 16

[58] Valerie Sessions and Marco Valtorta. The effects of data quality on machine learning algorithms (research-in-progress iq concepts, tools, metrics, measures, models, and methodologies). 7

[59] Jaemun Sim, Jonathan Sangyun Lee, and Ohbyung Kwon. Missing values and optimal selection of an imputation method and classification algorithm to improve the accuracy of ubiquitous computing applications. *Mathematical Problems in Engineering*, 2015, 2015. 38

[60] Marina Soley-Bori. Dealing with missing data: Key assumptions and methods for applied analysis. *Boston University*, 2013. 23

[61] Phil Spector. An introduction to the sas system. *Department of Statistics, University of California, Berkeley. URL: http://www. stat. berkeley. edu/classes/s100/sas. pd f*, 2003. 23

[62] Fei Tang and Hemant Ishwaran. Random forest missing data algorithms. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 10(6):363–377, 2017. 14

[63] R Core Team et al. R: A language and environment for statistical computing, 2013. 23

[64] Isabel Valera and Zoubin Ghahramani. Automatic discovery of the statistical types of variables in a dataset. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 3521–3529, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR. 9, 10, 26

[65] Joaquin Vanschoren, Jan N Van Rijn, Bernd Bischl, and Luis Torgo. Openml: networked science in machine learning. *ACM SIGKDD Explorations Newsletter*, 15(2):49–60, 2014. 2, 26

[66] Antonio Vergari, Alejandro Molina, Robert Peharz, Zoubin Ghahramani, Kristian Kersting, and Isabel Valera. Automatic bayesian density analysis. *arXiv preprint arXiv:1807.09306*, 2018. 30

[67] Roemer Vlasveld. Introduction to one-class support vector machines, 2013. 17

[68] Wikipedia contributors. Local outlier factor — Wikipedia, the free encyclopedia, 2018. [Online; accessed 7-November-2018]. ix, 18

[69] Wikipedia contributors. Missing data — Wikipedia, the free encyclopedia, 2018. [Online; accessed 3-November-2018]. 8

[70] Wikipedia contributors. Pearson correlation coefficient — Wikipedia, the free encyclopedia, 2018. [Online; accessed 11-November-2018]. 35

[71] Wikipedia contributors. Standard deviation — Wikipedia, the free encyclopedia, 2018. [Online; accessed 7-November-2018]. ix, 16

[72] Wikipedia contributors. Xml schema — Wikipedia, the free encyclopedia, 2018. [Online; accessed 8-November-2018]. 9

[73] Wikiversity. Duplicate record detection — wikiversity,, 2018. [Online; accessed 16-May-2018]. 7

[74] Leland Wilkinson and Michael Friendly. The history of the cluster heat map. *The American Statistician*, 63(2):179–184, 2009. 22

[75] Yang C Yuan. Multiple imputation for missing data: Concepts and new development (version 9.0). *SAS Institute Inc, Rockville, MD*, 49:1–11, 2010. 12

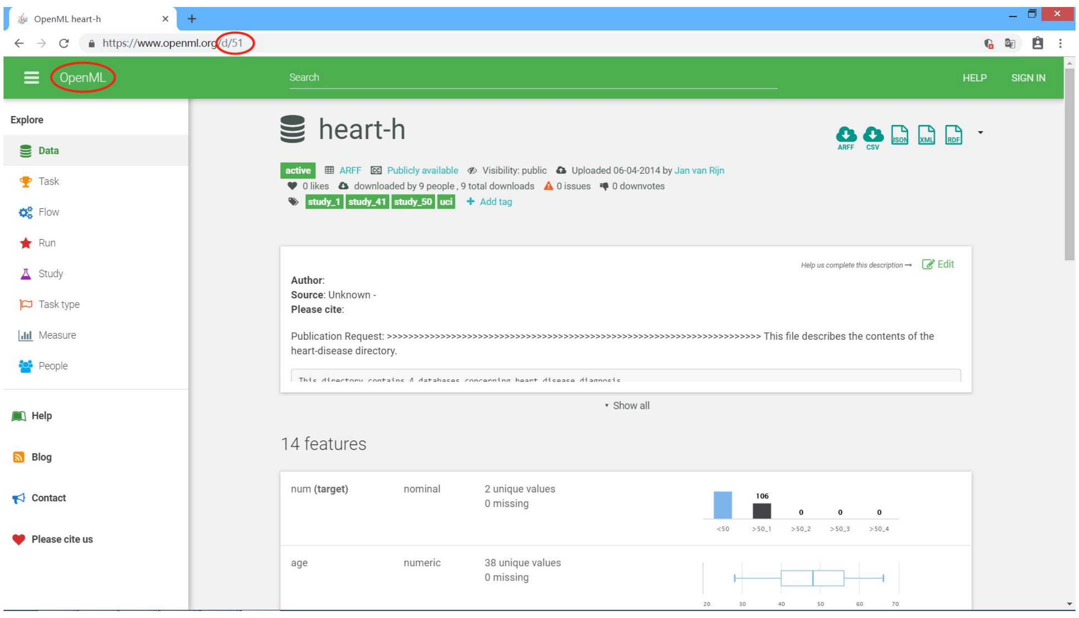
Appendix A

Demo

This chapter shows how to clean a random dataset from OpenML using the automatic data cleaning tool developed in this thesis.

A.1 Acquire Datasets from OpenML

The first step is to acquire the datasets from OpenML. The dataset ID can be found in the address bar as shown in the red circle in Figure A.1. We need the dataset ID as the input of the automatic data cleaning tool.



heart-h

active ARFF Publicly available Visibility: public Uploaded 06-04-2014 by Jan van Rijn
0 likes downloaded by 9 people , 9 total downloads 0 issues 0 downvotes
study_1 study_41 study_50 uci Add tag

Help us complete this description →

Author:
Source: Unknown -
Please cite:

Publication Request: >> This file describes the contents of the heart-disease directory.

```
This directory contains 4 databases concerning heart disease diagnosis
```

14 features

num **(target)**	nominal	2 unique values 0 missing	106 0 0 0
age	numeric	38 unique values 0 missing	

Figure A.1: Demo: get dataset ID on OpenML

A.2 Auto Clean with Automatic Cleaning Tool

We take the dataset 51 heart-h as an example to show how to use the tool. The process is quite simple as shown in the code below. We only have to input the dataset ID to the function *autoclean*() and the cleaned data will be returned in DataFrame. During the automatic cleaning process, the information of the dataset will be presented to the user and questions will be put forward when it is necessary for the user to intervene.

A.2.1 Input dataset ID

```
import dataclean as dc

# input openml dataset id
df_cleaned = dc.autoclean(51)
```

A.2.2 Show important features

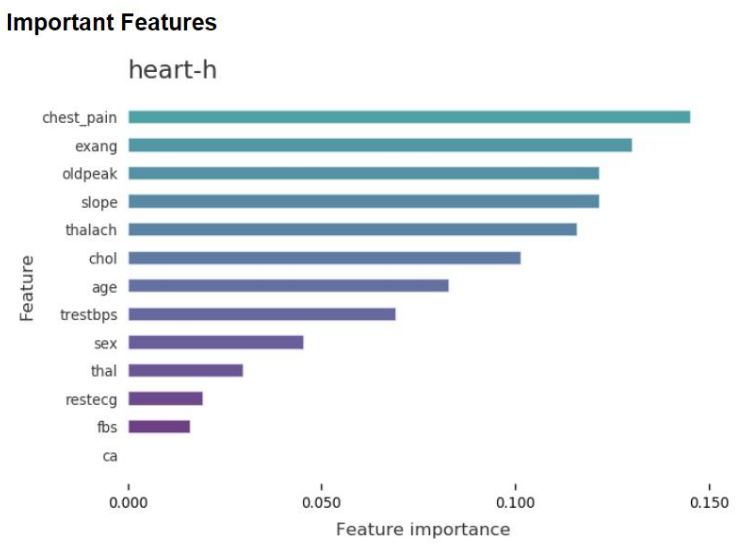
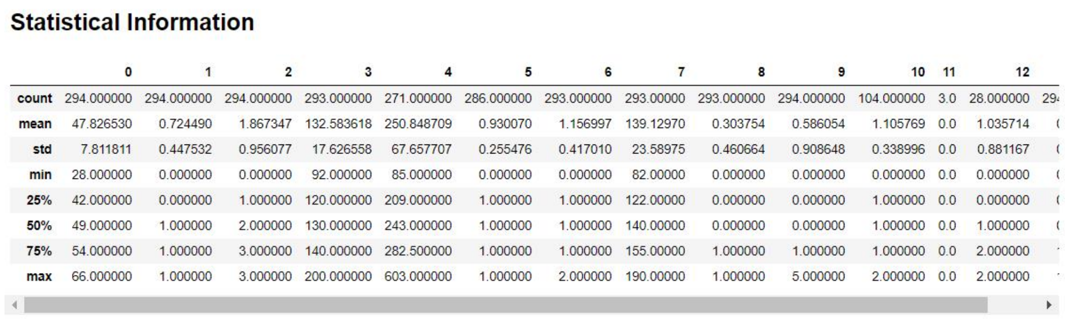


Figure A.2: Demo: show important features

A.2.3 Show statistical information

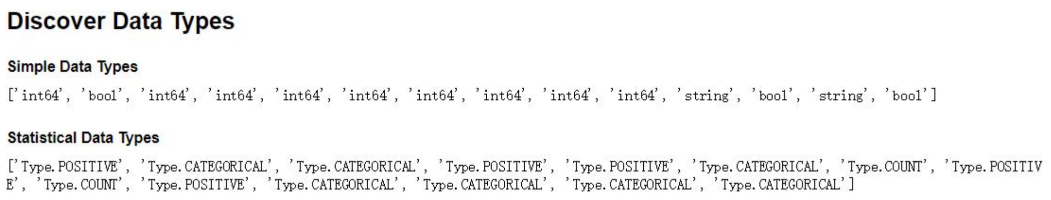


Statistical Information

	0	1	2	3	4	5	6	7	8	9	10	11	12
count	294.000000	294.000000	294.000000	293.000000	271.000000	286.000000	293.000000	293.00000	293.000000	294.000000	104.000000	3.0	28.000000
mean	47.826530	0.724490	1.867347	132.583618	250.848709	0.930070	1.156997	139.12970	0.303754	0.586054	1.105769	0.0	1.035714
std	7.811811	0.447532	0.956077	17.626558	67.657707	0.255476	0.417010	23.58975	0.460664	0.908648	0.338996	0.0	0.881167
min	28.000000	0.000000	0.000000	92.000000	85.000000	0.000000	0.000000	82.00000	0.000000	0.000000	0.000000	0.0	0.000000
25%	42.000000	0.000000	1.000000	120.000000	209.000000	1.000000	1.000000	122.00000	0.000000	0.000000	1.000000	0.0	0.000000
50%	49.000000	1.000000	2.000000	130.000000	243.000000	1.000000	1.000000	140.00000	0.000000	0.000000	1.000000	0.0	1.000000
75%	54.000000	1.000000	3.000000	140.000000	282.500000	1.000000	1.000000	155.00000	1.000000	1.000000	1.000000	0.0	2.000000
max	66.000000	1.000000	3.000000	200.000000	603.000000	1.000000	2.000000	190.00000	1.000000	5.000000	2.000000	0.0	2.000000

Figure A.3: Demo: show statistical information

A.2.4 Automatic discovery of data types



Discover Data Types

Simple Data Types

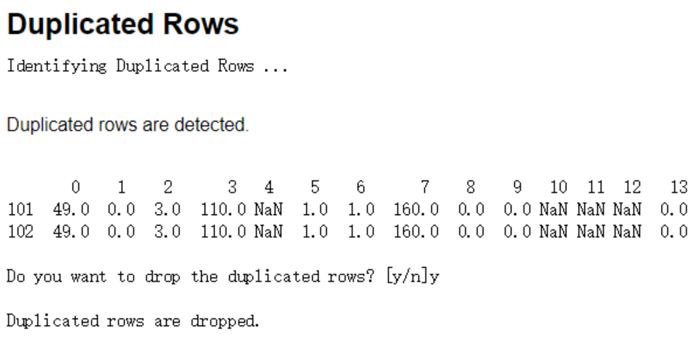
```
['int64', 'bool', 'int64', 'int64', 'int64', 'int64', 'int64', 'int64', 'int64', 'int64', 'string', 'bool', 'string', 'bool']
```

Statistical Data Types

```
['Type.POSITIVE', 'Type.CATEGORICAL', 'Type.CATEGORICAL', 'Type.POSITIVE', 'Type.POSITIVE', 'Type.CATEGORICAL', 'Type.COUNT', 'Type.POSITIV
E', 'Type.COUNT', 'Type.POSITIVE', 'Type.CATEGORICAL', 'Type.CATEGORICAL', 'Type.CATEGORICAL', 'Type.CATEGORICAL']
```

Figure A.4: Demo: automatic discovery of data types

A.2.5 Detect duplicated rows



Duplicated Rows

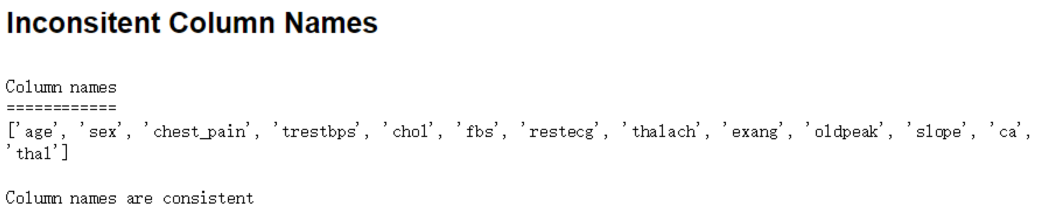
```
Identifying Duplicated Rows ...
```

Duplicated rows are detected.

```
        0    1    2      3   4    5    6      7    8    9  10  11  12   13
101  49.0  0.0  3.0  110.0 NaN  1.0  1.0  160.0  0.0  0.0 NaN NaN NaN  0.0
102  49.0  0.0  3.0  110.0 NaN  1.0  1.0  160.0  0.0  0.0 NaN NaN NaN  0.0

Do you want to drop the duplicated rows? [y/n]y

Duplicated rows are dropped.
```

Figure A.5: Demo: detect duplicated rows

A.2.6 Detect inconsistent column names

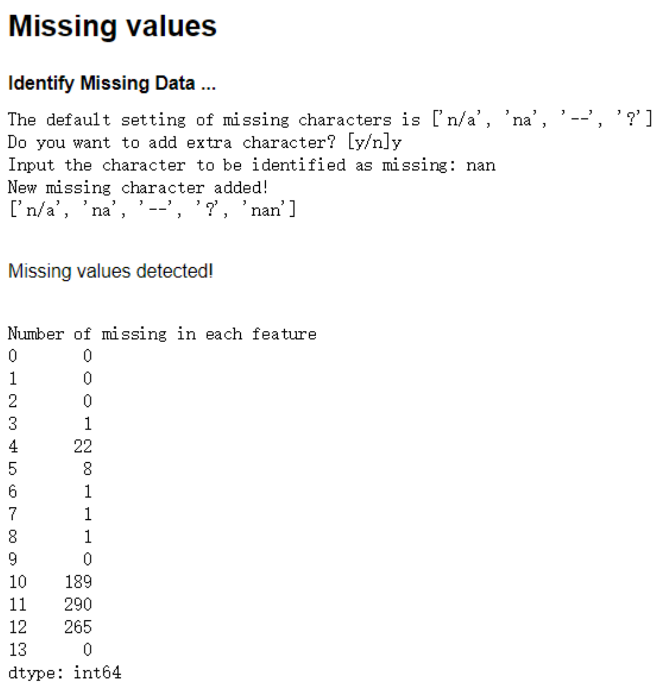


Inconsitent Column Names

```
Column names
============
['age', 'sex', 'chest_pain', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach', 'exang', 'oldpeak', 'slope', 'ca',
'thal']

Column names are consistent
```

Figure A.6: Demo: detect inconsistent column names

A.2.7 Automatic missing value handling



Missing values

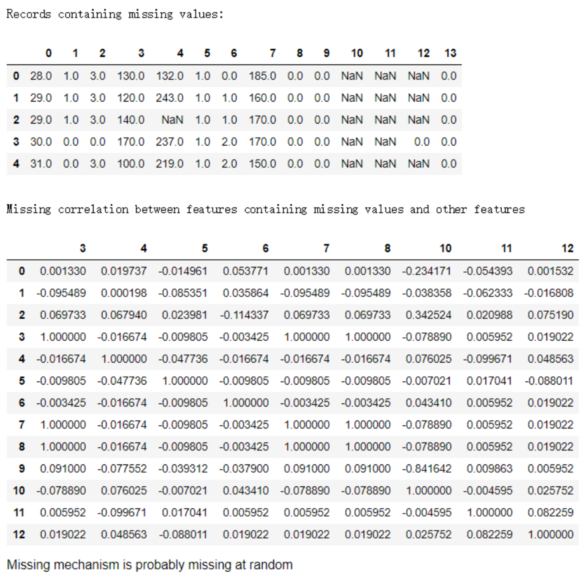
Identify Missing Data ...

```
The default setting of missing characters is ['n/a', 'na', '--', '?']
Do you want to add extra character? [y/n]y
Input the character to be identified as missing: nan
New missing character added!
['n/a', 'na', '--', '?', 'nan']
```

Missing values detected!

```
Number of missing in each feature
0       0
1       0
2       0
3       1
4      22
5       8
6       1
7       1
8       1
9       0
10    189
11    290
12    265
13      0
dtype: int64
```

Figure A.7: Demo: identify missing values



```
Records containing missing values:
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	28.0	1.0	3.0	130.0	132.0	1.0	0.0	185.0	0.0	0.0	NaN	NaN	NaN	0.0
1	29.0	1.0	3.0	120.0	243.0	1.0	1.0	160.0	0.0	0.0	NaN	NaN	NaN	0.0
2	29.0	1.0	3.0	140.0	NaN	1.0	1.0	170.0	0.0	0.0	NaN	NaN	NaN	0.0
3	30.0	0.0	0.0	170.0	237.0	1.0	2.0	170.0	0.0	0.0	NaN	NaN	0.0	0.0
4	31.0	0.0	3.0	100.0	219.0	1.0	2.0	150.0	0.0	0.0	NaN	NaN	NaN	0.0

```
Missing correlation between features containing missing values and other features
```

	3	4	5	6	7	8	10	11	12
0	0.001330	0.019737	-0.014961	0.053771	0.001330	0.001330	-0.234171	-0.054393	0.001532
1	-0.095489	0.000198	-0.085351	0.035864	-0.095489	-0.095489	-0.038358	-0.062333	-0.016808
2	0.069733	0.067940	0.023981	-0.114337	0.069733	0.069733	0.342524	0.020988	0.075190
3	1.000000	-0.016674	-0.009805	-0.003425	1.000000	1.000000	-0.078890	0.005952	0.019022
4	-0.016674	1.000000	-0.047736	-0.016674	-0.016674	-0.016674	0.076025	-0.099671	0.048563
5	-0.009805	-0.047736	1.000000	-0.009805	-0.009805	-0.009805	-0.007021	0.017041	-0.088011
6	-0.003425	-0.016674	-0.009805	1.000000	-0.003425	-0.003425	0.043410	0.005952	0.019022
7	1.000000	-0.016674	-0.009805	-0.003425	1.000000	1.000000	-0.078890	0.005952	0.019022
8	1.000000	-0.016674	-0.009805	-0.003425	1.000000	1.000000	-0.078890	0.005952	0.019022
9	0.091000	-0.077552	-0.039312	-0.037900	0.091000	0.091000	-0.841642	0.009863	0.005952
10	-0.078890	0.076025	-0.007021	0.043410	-0.078890	-0.078890	1.000000	-0.004595	0.025752
11	0.005952	-0.099671	0.017041	0.005952	0.005952	0.005952	-0.004595	1.000000	0.082259
12	0.019022	0.048563	-0.088011	0.019022	0.019022	0.019022	0.025752	0.082259	1.000000

Missing mechanism is probably missing at random

Figure A.8: Demo: show information of missing values

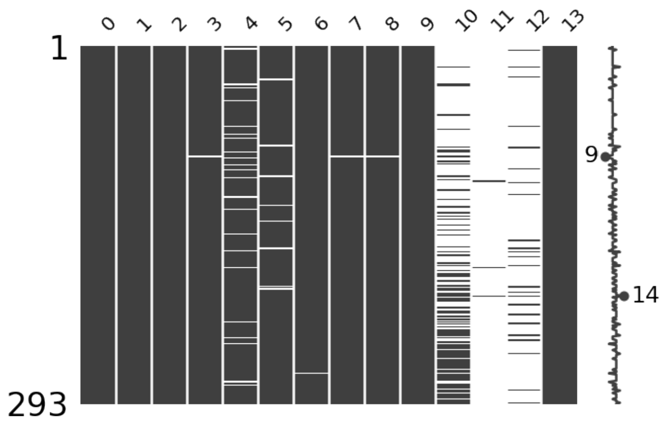


Figure A.9: Demo: visualize missing values with matrix

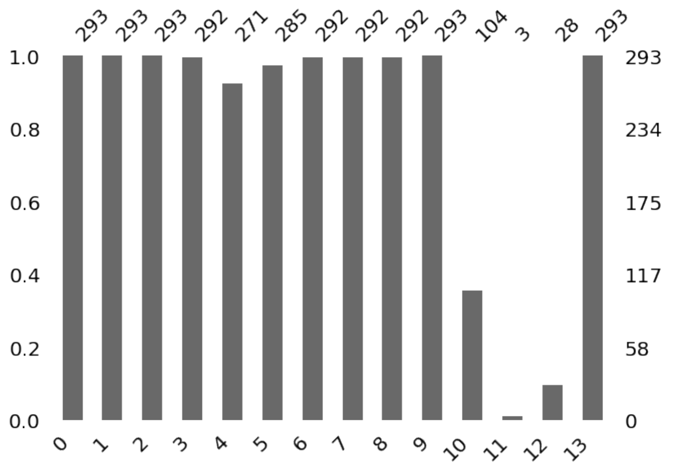


Figure A.10: Demo: visualize missing values with bar chart

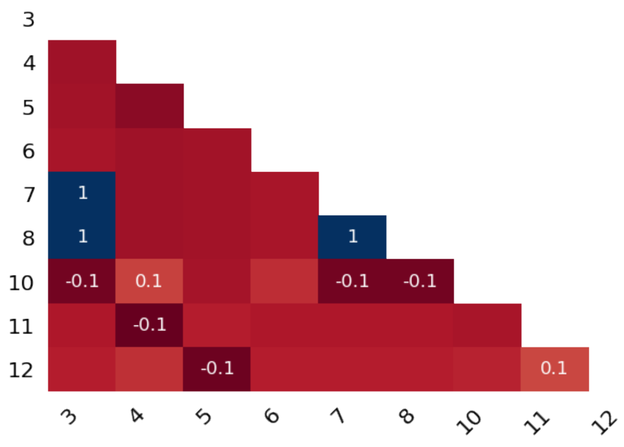
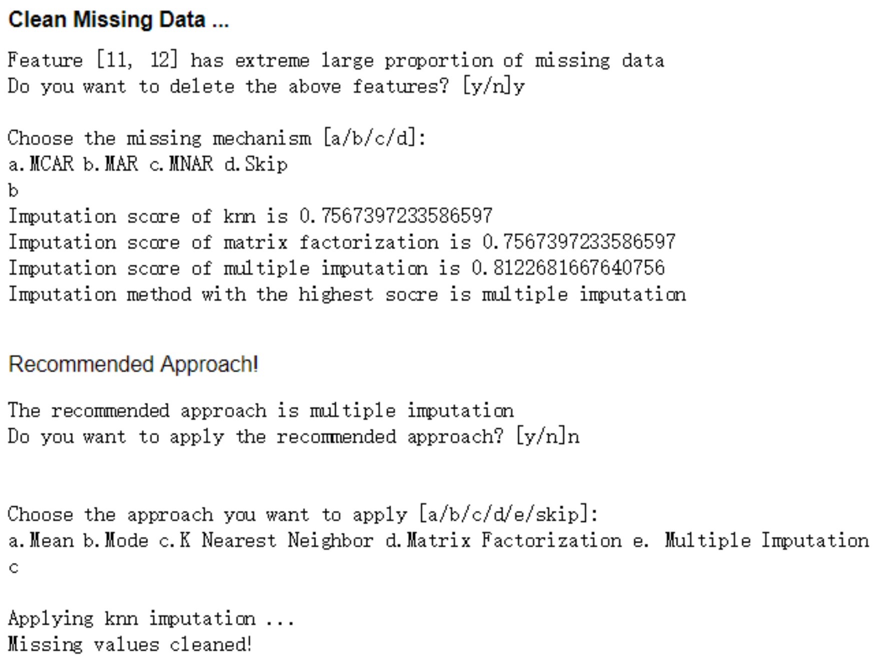


Figure A.11: Demo: visualize missing values with heatmap



Clean Missing Data ...

```
Feature [11, 12] has extreme large proportion of missing data
Do you want to delete the above features? [y/n]y

Choose the missing mechanism [a/b/c/d]:
a.MCAR b.MAR c.MNAR d.Skip
b
Imputation score of knn is 0.7567397233586597
Imputation score of matrix factorization is 0.7567397233586597
Imputation score of multiple imputation is 0.8122681667640756
Imputation method with the highest socre is multiple imputation
```

Recommended Approach!

```
The recommended approach is multiple imputation
Do you want to apply the recommended approach? [y/n]n


Choose the approach you want to apply [a/b/c/d/e/skip]:
a.Mean b.Mode c.K Nearest Neighbor d.Matrix Factorization e. Multiple Imputation
c

Applying knn imputation ...
Missing values cleaned!
```

Figure A.12: Demo: clean missing values

A.2.8 Automatic Outlier Detection

Visualize Outliers ...

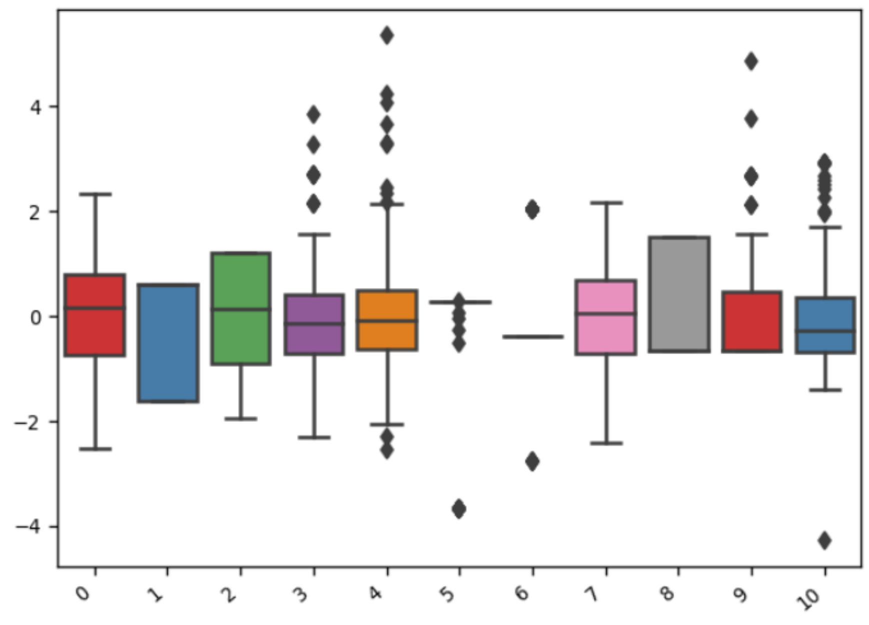
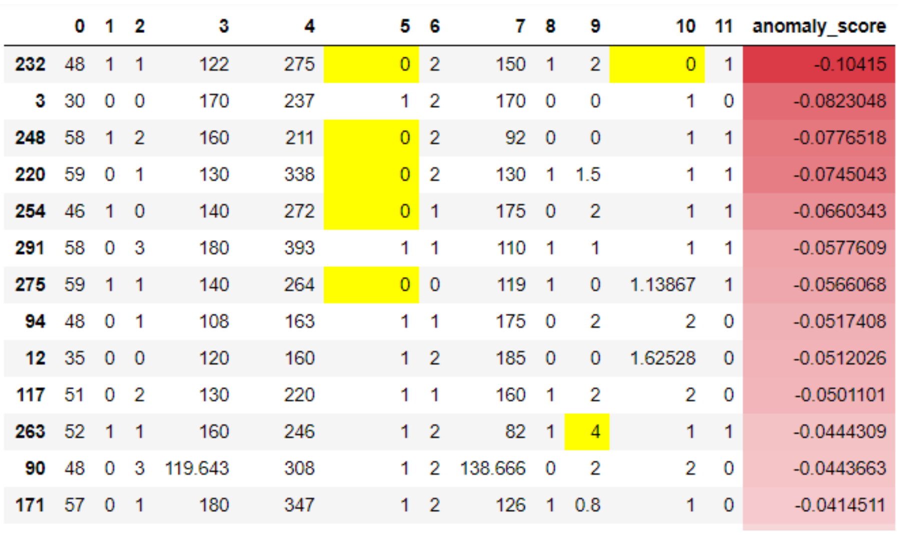


Figure A.13: Demo: visualize outliers with box plot



	0	1	2	3	4	5	6	7	8	9	10	11	anomaly_score
232	48	1	1	122	275	0	2	150	1	2	0	1	-0.10415
3	30	0	0	170	237	1	2	170	0	0	1	0	-0.0823048
248	58	1	2	160	211	0	2	92	0	0	1	1	-0.0776518
220	59	0	1	130	338	0	2	130	1	1.5	1	1	-0.0745043
254	46	1	0	140	272	0	1	175	0	2	1	1	-0.0660343
291	58	0	3	180	393	1	1	110	1	1	1	1	-0.0577609
275	59	1	1	140	264	0	0	119	1	0	1.13867	1	-0.0566068
94	48	0	1	108	163	1	1	175	0	2	2	0	-0.0517408
12	35	0	0	120	160	1	2	185	0	0	1.62528	0	-0.0512026
117	51	0	2	130	220	1	1	160	1	2	2	0	-0.0501101
263	52	1	1	160	246	1	2	82	1	4	1	1	-0.0444309
90	48	0	3	119.643	308	1	2	138.666	0	2	2	0	-0.0443663
171	57	0	1	180	347	1	2	126	1	0.8	1	0	-0.0414511

Figure A.14: Demo: visualize outliers with styled dataframe

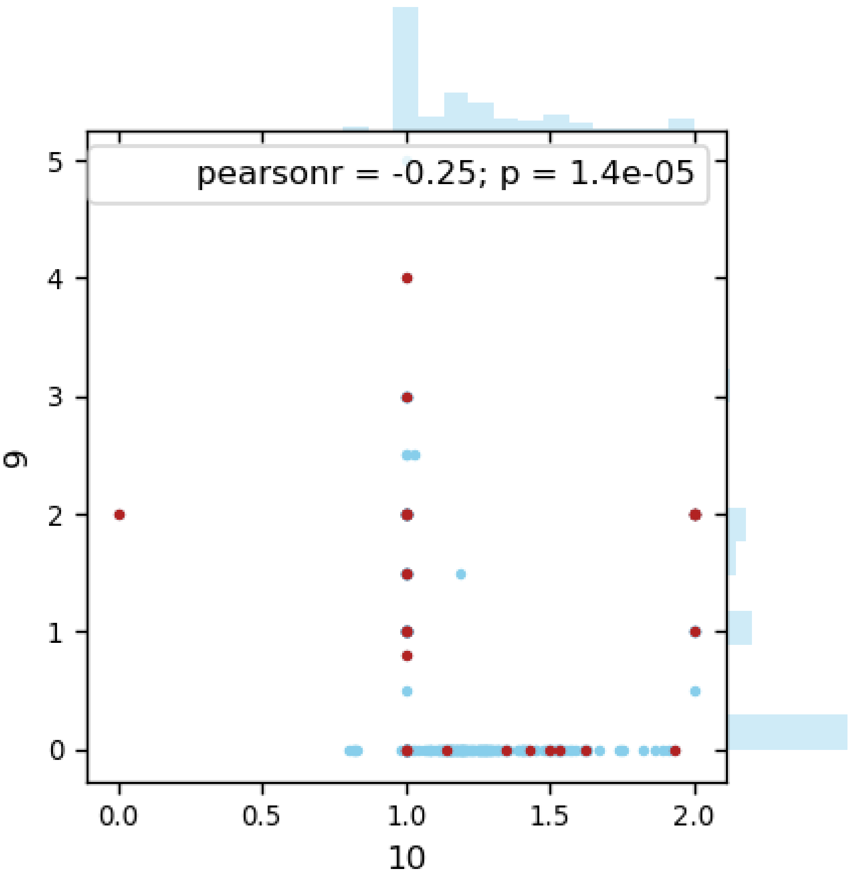


Figure A.15: Demo: visualize outliers with scatter plot

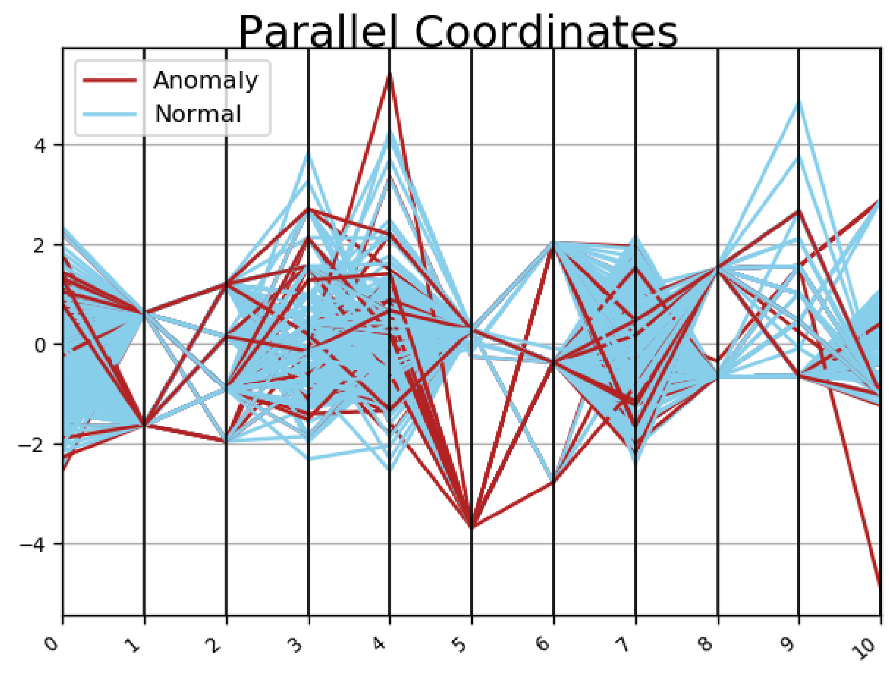
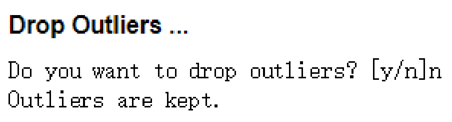


Figure A.16: Demo: visualize outliers with parallel coordinates plot



Drop Outliers ...

```
Do you want to drop outliers? [y/n]n
Outliers are kept.
```

Figure A.17: Demo: drop outliers

Appendix B

Datasets

B.1 Datasets Used in Automatic Discovery of Data Types

Table B.1 shows the datasets used for the evaluation of Bayesian method in Section 4.1. These datasets are the twenty most-runs datasets with various features types. They can be acquired from OpenML through the dataset ID.

OpenML Dataset ID	Name	Number of Features
31	credit-g	21
1464	blood-transfusion-service-center	5
334	monks-problems-2	7
50	tic-tac-toe	10
333	monks-problems-1	7
1494	qsar-biodeg	42
3	kr-vs-kp	37
1510	wdbc	31
1489	phoneme	6
37	diabetes	9
1479	hill-valley	101
1487	ozone-level-8hr	73
1063	kc2	22
1471	eeg-eye-state	15
1467	climate-model-simulation-crashes	21
1480	ilpd	11
1068	pc1	22
1492	one-hundred-plants-shape	65
1050	pc3	28
1462	banknote-authentication	5

Table B.1: Datasets used in automatic discovery of data types

B.2 Datasets Used in Automatic Outlier Detection

Table B.2 shows the datasets used in automatic outlier detection part. They are used for benchmarking of outlier detection algorithms and training the recommendation model as described in Section 4.3. These datasets have to be preprocessed by taking the minority class as outliers. All these datasets can be acquired from OpenML though the dataset ID.

OpenML Dataset ID	Name	Outlier Percentage(%)
10	lymph	4.1
214	glass	4.21
1510	wdbc	37.26
40910	speech	1.65
294	satellite_image	31.64
185	baseball	9.33
39	ecoli	2.68
1489	phoneme	29.35
1216	click_prediction_small	16.84
1116	musk	15.41
31	credit_g	30.0
37	diabetes	34.89
13	breast_w	34.48
1464	blood_transfusion_service_center	23.79
1565	heart	45.54
1017	arrhythmia	45.79
44	spambase	39.4
1063	kc2	20.49
1480	ilpd	28.64
1068	pc1	6.94
183	abalone	0.43
40536	speed_dating	16.47
1560	cardiotocography	12.94
38	sick	6.12
179	adult	23.93
1053	jm1	19.35
312	scene	17.91
1467	climate_model_simulation_crashes	8.52
772	quake	44.49
40597	yeast	0.34
40701	churn	14.14
40983	wilt	5.39
719	veteran	31.39
1054	mc2	32.3
11	balance_scale	7.84
34	postoperative_patient_data	2.22
23	cmc	22.61
2	anneal	0.89
26	nursery	2.54
4134	bioresponse	45.77
1075	datatrieve	8.46
338	grub_damage	31.61

Table B.2: Datasets used in automatic outlier detection