School of Information Technology and Engineering at the ADA University

School of Engineering and Applied Science
at the George Washington University

SEMI-AUTOMATED DATA CLEANSING

A Thesis
Presented to the Graduate Program of Computer Science and Data Analytics
of the School of Information Technology and Engineering
ADA University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in Computer Science and Data Analytics
ADA University

By
Reyhana Karimli

April, 2023

THESIS ACCEPTANCE

This Thesis by: Reyhana Karimli
Entitled: *Semi-automated Data Cleansing*

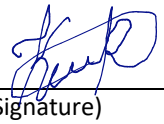
has been approved as meeting the requirement for the Degree of Master of Science in Computer Science and Data Analytics of the School of Information Technology and Engineering, ADA University.

Approved:

(Adviser)	(Date)
(Program Director)	(Date)
(Dean)	(Date)

ACADEMIC INTEGRITY STATEMENT

“I affirm that this is my own work, I attributed where I used the work of others, I did not facilitate academic dishonesty for myself or others, and I used only authorized resources for my Thesis, per the ADA University Academic Integrity requirements. If I failed to comply with this statement, I understand consequences will follow my actions. Consequences may range from failing the course to expulsion from the program/university and may include a transcript notation.”

Reyhana Karimli		24.04.23
(Full Name)	(Signature)	(Date: DD.MM.YY)

ABSTRACT

The aim of this master thesis was to develop a semi-automated data cleansing tool that can automatically perform data cleansing procedures on noisy datasets with minimal user input, resulting in clean and quality datasets ready for modelling. As a part of this project, I developed a desktop application which consists of a simple-to-use and informative graphical user interface that can take the initial dataset as input from the user and produce the cleaned one as an output along with the log file listing the actions made during the processing. In addition to the application, I also developed an automated rule-based algorithm that decides which data cleaning and pre-processing techniques to use for better results in terms of the quality of the produced dataset.

The application was developed by using PyQt package of Python, whereas the program logic utilized other machine learning and visualization packages such as sklearn and matplotlib. The performance of the application was evaluated in terms of accuracy, quality, and reproducibility. For this purpose, the datasets were added noisiness in terms of the number of missing values and duplicates in four different ratios and then cleaned and pre-processed by using the semi-automated data cleansing tool. The produced cleaned datasets are then fed into the random forest classifier to evaluate the variations in the performance of the model in terms of accuracy. The performance of the program was also examined in terms of the processing time depending on the noise ratio in the dataset. Finally, I evaluated the performance of the automatic mode of the program by comparing the accuracy of the machine learning model depending on the cleaning mode, namely Auto and Manual.

Overall, the application successfully performed automatic data cleansing and pre-processing techniques such as duplicate data handling, outlier handling, missing data imputation, normalization, and standardization without user interference. Additionally, it produced mostly better results in terms of the quality of the data in the automatic mode rather than in the manual. Finally, the application produced not only a cleaned dataset, but also successfully visualized the detected noise in the dataset and logged them with the respective modification made to it for further reference by the user.

Table of Contents

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

Information purging, otherwise called information cleaning or information scouring, is an urgent move toward information preprocessing that goes before information analysis, visualization, and making informed business decisions. For making decisions that are both accurate and dependable in a variety of business contexts, having access to data of high quality is of the utmost importance. However, in almost all real-world scenarios, data is frequently noisy and of poor quality, resulting in inaccurate and unreliable decisions.

The data cleansing pipeline includes several steps, including dealing with outliers and noise in the dataset, dealing with misplaced values, imputing missing values, and removing incorrect data. The process of removing incorrect data entails locating and eliminating data that is incorrect or erroneous because of human error, a malfunctioning system, or a variety of other factors. Dealing with lost values includes recognizing and revising information that is in some unacceptable configuration or area. Imputing missed values involves substituting estimated values derived from statistical analysis or other methods for the data that is missing. Identifying and removing data points that are significantly different from most of the data or may be caused by errors or anomalies in the data is one way to address outliers and noise in the dataset.

The issue of noisy data affects not only tabulated data but also image and textual data. Image processing requires numerous preprocessing ventures prior to being utilized as a contribution to a model. In a similar vein, the text data for textual modelling must be preprocessed and noise-free by eliminating obtrusive characters and words like stop words, punctuation, and redundant links.

Consequently, there is a need to direct further exploration in this field to foster summed-up answers for data cleansing. These summed-up arrangements ought to have the option to help in cleaning classified information as well as in unstructured information like text and pictures. Artificial intelligence, machine learning, and natural language processing are some of the developed methods that can be utilized to automate data cleansing procedures and enhance the accuracy and dependability of business decisions. Businesses in a variety of industries, including healthcare, finance, retail, and others, will be significantly impacted by the creation of these kinds of efficient data-cleansing solutions.

In the process of preprocessing data, data cleansing is an essential step that comes before modelling. Correct data removal, handling misplaced values, imputing missed values, and dealing with outliers and noise in the dataset are all part of the process. The issue of dirty data affects all kinds of information in real-world situations for a variety of reasons, some of which can be pre-determined and prevented while others cannot. As a result, more research needs to be done in this area to come up with general solutions for data cleansing that can help businesses make decisions that are accurate and dependable.

1.1 Definition of the Problem

One of the first and most time-consuming steps in creating a machine learning model is structured data preprocessing. The data scientist is responsible for feeding the clean and complete data to the machine learning model to obtain the most accurate results because the models are designed to work with clean and complete data. There are numerous approaches and stages involved in the data pre-processing process itself.

Missing records are one of the most frequently resolved issues during data cleaning. To deal with the lack of data, several ideas and strategies have been suggested. For instance, eradicating the records that are missing, substituting values derived from statistical calculations, or employing machine learning algorithms.

In addition, datasets containing categorical or linguistic variables frequently contain misspelt values, which means that the same word is counted multiple times as distinct due to its distinct wrong spellings. Also, data that have the same meaning but are interpreted differently (for example USA and United States) should be considered. Modelling and analysis will be erroneous because of this. In addition, due to the large number of records in the dataset, the individual will not be able to locate and correct every misspelt word.

In real-world situations, a few factors and values should be modified and changed in view of specific custom standards and necessities as far as the worth reaches to guarantee the rightness of the qualities. A negative age column value, for instance, indicates incorrect data, which should be asserted and corrected before entering the data into the model or performing the analysis.

1.2 The Objective of Study

The creation of a pipeline for automating the laborious data-cleaning procedure is the primary objective of this study. The goal of the project is to learn about the most important aspects of cleaning procedures and create logic that will automatically choose which methods to apply to each dataset that is provided as input. Due to the project's semi-automatic nature, the final project will process the input file with the fewest user interactions possible. On the other hand, it ought to permit the user to modify the cleaning procedure in accordance with their requirements and specific applications. The final project will not only automatically identify the methods and functions for cleaning the data, but it will also show statistical information about the inconsistencies and dirtiness that were found in the original dataset.

The user will be able to understand the proportion of dirtiness for each column and the dataset as a result, allowing the chosen method to be based on the dataset's individual factors and treating them according to predefined logic that chooses the preferred cleaning methods and techniques for the input.

Additionally, the task application will yield the log record expressed line by line with a timestamp, data, debugging and warnings that happened in the dataset during the handling. If the user wishes to modify the logic or some of the semiautomated steps, this will show the user every step taken during the cleaning. In addition, to determine the efficiency and performance of the chosen logic and pipeline, the project will conduct an analysis of the outcomes using both automatic and manual logic.

1.3 Significance of the Problem

The data scientists will be able to export data of the highest quality to analytics or data warehouses with the assistance of the tool that has improved functions as part of the master project, such as outlier detection, missing data imputation, and categorical data handling. This will pave the way for better insights, precise predictions, simplified data preparation for analytics, machine learning data, and more. It aims to cut down on noise and inconsistent data as well as the time spent by scientists and data analysts processing, debugging, and researching various data cleaning functions. As a result, it was crucial to develop this automated data-preprocessing tool to improve the data's overall quality for data analytics, modelling, and storage.

As a data analyst in the banking industry, it's common to deal with customer data that has missing or incorrect values. However, reports in the banking sector typically follow a similar format and undergo numerous pre-processing steps prior to analysis. This gives us a chance to automate the data cleansing procedure for this field and boost the accuracy of the models' performance. Additionally, the enormous amount of time required for data preparation prior to modelling can be reduced by automating the dataset's preprocessing.

When working with multilingual datasets, the improved tool's language detection and keyword extraction capabilities can help identify the language of the data. Additionally, it can separate catchphrases from the information, empowering the expert to distinguish fundamental data and examples. The accuracy of the outcomes can be enhanced by using the extracted keywords as modelling features.

Additionally, the banking sector's automation of the data cleansing procedure has the potential to lessen the likelihood of human error, manual processing, and system failure-related errors. The accuracy and consistency of the data can be enhanced through the application of machine learning methods like outlier handling, duplicate detection, and missing data handling. The comparability of the variables and features across a dataset that has been prepared for modelling will be further ensured by using data preprocessing techniques like standardization and normalization in addition to the cleaning of the data.

Additionally, data scientists and researchers go through hours investigating the various information purifying capabilities and methods, as well as managing blunders and troubleshooting the issues that

occurred because of different reasons in their code going from linguistic structure to straightforward composing mistakes. In addition to paving the way for a significant increase in the improvement of the quality of data exported to analytics and data warehouses in a wide variety of industries, the semi-automated data cleansing tool developed as part of this master project will also reduce the amount of time spent exploring, comparing, and analyzing various data pre-processing and data cleaning methodologies and techniques to determine the one that works best for the specific industry and application field. By allowing users to experiment with all the functions in one location without having to read the documentation or syntax, the semi-automated data cleansing tool may be able to assist users in manipulating and pre-processing the raw dataset. It can also help users deal with errors and debugging that arise.

In this way, the difficulties, and issues for both the clients and the organizations expressed beforehand to make the requirement for the mechanization of the data cleansing and pre-handling methods is a definitive answer for some issues and difficulties looked at by information researchers in their everyday work. The semi-mechanized information purifying device cannot just address the conflicting and loud information issue by working on the nature of the information yet in addition work on the precision and consistency of the models prepared on the info datasets, which will decrease the gamble of mistaken ends, bits of knowledge and business choices related with the results and forecasts made by the researchers utilizing those models.

At long last, the utilization of a semi-automated information purifying tool that joins the different strategies and systems for exception dealing with, copy location, missing information taking care of, normalization, and standardization can additionally further develop the expectation exactness of the AI models as well as prepare to reproducibility and the likeness of the outcomes created by those models. Using a dataset from the real world, future research can examine how well an improved tool works in other sectors.

1.4 Review of Significant Research

This project examines five distinct studies concerning the crucial step of data preprocessing known as data cleansing. The first study by F. Ridzual and W. M. Nazmee [1] discusses the necessity of big data-friendly methodologies and the significance of data cleansing for accurate modelling. D. Mertz's second study [2] gives an overview of the various data cleansing methods and libraries that are available to students, data scientists, and software engineers. The third concentrate by Xin Wang, Howard J. Hamilton, and Yashu Biter [3] proposes a cosmology-based way to deal with information cleaning and portrays a structure for different data-cleansing undertakings. A transformer-based attention algorithm for textual models is proposed and compared to other neural network models in the fourth study by A. Vaswani et al. [4]. The goal of the fifth study by Y. Li, A. Anastasopoulos, and A. W. Black [5] is to create a grammatical error correction model that is based on BERT.

The studies shed light on various facets of data cleansing and emphasize the significance of considering domain-specific knowledge when cleaning data. The ontology-based strategy that Xin Wang et al. [3] stands out as an efficient method for removing syntactic and semantic errors from data. In a similar vein, A. Vaswani et al.'s transformer-based attention algorithm [6] has demonstrated promising results for accelerating textual model performance. The study by Y. Li, A. Anastasopoulos, and A. W. Black [5] sheds light on the drawbacks of the existing models for correcting grammatical errors and highlights the need for improved methods.

In general, there are studies that provide data scientists, software engineers, teachers, and students working with data with useful information and methods for data cleansing. These studies emphasize the requirement for data cleansing strategies and domain-specific knowledge. These studies' insights can be used to make modelling more accurate and make big data processing more efficient.

1.5 Assumptions and Limitations

The conducted research revealed several limitations of data cleansing in computer vision, textual domains, structured data, and big data. The computer vision data cleansing processes can be extremely

computationally expensive for the recurrent neural networks to process each pixel of an image. Another challenge of image preprocessing is the collection of a reasonable dataset for the modelling. Although in training and test procedure, readily available datasets are used, in real-world situations, not all the images are of the same size, shape and geometry. The collection of the data for the modelling and classification of the new domain becomes even more complex as there are no available datasets to train and test the model.

The preprocessing of the text differs in terms of the steps from others. It consists of several stages such as tokenization, stemming, creation of a stop word list etc. For some languages, there are readily available datasets due to an enormous number of research conducted for those languages such as English. For other languages, such as Azerbaijani, building a machine learning or deep learning model can be relatively complex due to the lack of the language corpus available online. This can be also a challenge during the tokenization process. Each language has its own punctuation and word separation rules. The tokenization method used for English cannot be used for Chinese or Japanese. In languages such as Arabic with complex morphology, there can be extracted several from each word.

Even with less complex datasets such as tabulated records, data cleaning can be an enormous challenge to tackle. Usually, such datasets suffer from missing values, misplaced values, outliers, and duplicated values. For example, missing value imputation can cause a huge bias in the dataset and lead to false conclusions about the insights that might be derived from the dataset. Moreover, usually, there is no ground truth to compare the imputed values and estimate the model in terms of accuracy. Therefore, before applying any data cleansing algorithm the dataset must be well understood by the subject domain expert.

Big Data can contain all the previously mentioned kinds of datasets. However, in this case, data cleansing techniques should meet all five criteria of Big Data: velocity, volume, variety, veracity, and value. The data cleansing methodologies and frameworks should be fast enough to keep up with the velocity of the big data and be able to proceed high volume of data within a short amount of time. Additionally, it is of high importance to consider the subject domain before attempting to modify the dataset so that the original value of the dataset is not lost or changed. Wrong imputed data may lead to wrong decisions in the business processes.

2 LITERATURE REVIEW

There has been a lot of research on data cleansing in various domains ranging from computer vision to tabulated excel sheet datasets. In of the papers [1] F. Ridzual and W. M. Nazmee state that the not cleaned datasets lead to poor modelling in the industry in terms of the accuracy of the model. They reviewed various data cleansing methodologies for big data. Data cleansing methodologies were separated into two wide kinds, namely, traditional, and big data cleansing. It was concluded that modem methodologies cannot fully meet the big data criteria such as scalability, veracity, and value of the data. Thus, during the design of the data cleansing methodologies, all five criteria of big data need to be considered by the subject expert.

D. Mertz describes in his book various techniques available for data cleansing that can be useful and helpful for data scientists, software engineers, teachers, students, and anyone working with data [2]. He described and explained the features of the pandas, SciPy, scikit-learn, and Tidyverse, which are the libraries of the Python and R programming languages. The described techniques allow users to work with various data formats such as JSON, CSV, SQL, or NoSQL. The described methodologies can be used in feature engineering to deal with outliers, imbalanced classes, and dimensionality in the dataset. The book mentions the techniques that are effective tools for missing data imputation, anomaly detection and data ingestion.

The scientists from Canada, Xin Wang, Howard J. Hamilton, and Yashu Biter, in their paper, described the Oncology Base Approach to Data Cleaning [3]. They were able to clean data both from syntactic and semantic errors by using the ontology-based approach, which is the classification and explanation of the domain. By using this approach, they were able to detect and correct misspellings, missing values and misplace values. The study group created a framework for various data cleansing

tasks. The framework consists of an abstract class, which is a superclass of all other possible classes related to the tasks that the users may need to perform during data cleansing. They achieved various precision and recall levels. It was concluded that there was a tradeoff between high precision and recall. As a future work, they state that each domain needs its own data cleansing methodology to achieve good results. This is because without domain knowledge it is impossible to clean effectively semantic errors in a particular dataset.

A. Vaswani et al proposed a Transformer based on an attention algorithm which can be used for textual models such as language translators [4]. In the paper, the attention-based transformer model is compared to other neural network models such as recurrent neural networks, long short-term memory, and gated recurrent neural networks. The Transformer model proposed in this paper instead of recurrent recurrence completely relies on the attention mechanism. It consists of the encoder and decoder, which all contain the stacked self-attention fully connected layers. The model is compared to other layers in terms of various performance metrics such as complexity per layer, the minimum number of sequential operations, and maximum path length. As a part of the experiment, they achieved much faster performance with the proposed model than recurrent convolutional neural networks. It was concluded that the model will be improved further to apply it to other problems such as dealing with large inputs for image and video processing.

Another paper related to the language representational model is BERT, which stands for Bidirectional Encoder Representations for Transformers [7] developed by J. Devlin, M. Chang, K. Lee, and K. Toutanova. They proposed the pre-trained BERT model which could solve a wide range of linguistic problems such as question answering, next-sentence prediction and paraphrasing by just changing the output layer. BERT is a framework that consists of pre-training and fine-tuning stages. The proposed model is based on the original transformer. However, instead of using constrained self-attention, BERT was based on bidirectional self-attention. BERT was assessed by several experiments based on GLUE, which stands for General Language Understanding Evaluation. The model was experimenting with different datasets and various hyperparameters such as large and small-sized. The average results showed that the BERT large had better performance than BERT small and other transformer models. It was concluded for the model to have good performance it is important to be pre-trained with large datasets such as big documents instead of just a random bag of words.

Another paper about handling missing data was written by Jesse E. Robles-Alcaraz in 2021 [8]. He compares the model-based and multiple imputations to simple analytical and single-value imputations. It is stated that the former is more effective. The experiment was conducted on the student asthma dataset with eight variables such as the severity of asthma, age, gender, allergy, and the reading test score. There were four variables with missing values. The most missing values were in reading test scores and allergy variables, with 48.7% and 46.1%, respectively. It was concluded that although the model-based approaches tend to have more realistic, unbiased, and accurate results, they could not be trusted with a high number of missing values. This is due to the reason that missing value imputation cannot be fully unbiased. Moreover, he does not believe that the dataset is completely random as in this study the number of students with positive cases was not enough to make accurate predictions.

Another problem in data cleansing the spelling and grammar error correction. Students at Carnegie Mellon University Y. Li, A. Anastasopoulos and A. W. Black completed a study on developing a BERT-based grammatical error correction [5]. They highlighted that current grammatical error correction models consider the tasks as sequence generation, which needs a lot of data space. This leads to limitations in using grammatical error correction in software applications. Therefore, they proposed to use contextual information from the pre-trained model such as BERT to overcome the space limitations. Additionally, they stated they stated that using a pre-trained model such as BERT would lead to the application of the model in different languages and that would solve grammatical error correction in multilingual situations. They measured the performance in terms of the model accuracy. The model achieved an accuracy of higher than 70%, which is quite good for such a complex problem. The proposed model needs further work in improving the error fertility to predict the number of the required MASK tokens more precisely. One of the disadvantages of the model is that it could improve

its span detection. In other words, by correcting the words in the sentence the final output to completely another meaning which is unacceptable in some cases. However, considering that there was no prior study on using contextual models for grammatical error correction the model has achieved good results and could pave the way to solving more advanced problems.

S. Zhang, H. Huang, J. Liu, and H. Li proposed a spelling error correction model with a soft-masked BERT [9]. The research was done on spelling error correction in the Chinese language. They stated that due to the mask-based approach used in BERT, it is not possible to detect spelling mistakes in all the positions in the word, which leads to a suboptimal solution. Therefore, novel research has been conducted by the researchers of ByteDance AI to improve the accuracy of spelling error correction. They proposed a model named Soft-Masked BERT, which consists of a network of error detection and error corrections. The experimental results on two datasets showed better performance than the solely BERT-based approach, which was the baseline for the proposed model. Although the model has achieved significant dominance on baseline models such as n-gram, rule-based and BERT-based classifiers in terms of accuracy in spelling error correction, it still needs further work in grammatical error correction, because merely spelling error correction of the words in the sentence might not lead to completely correct contextual meaning.

M. Tan, D. Chen, Z. Li, and P. Wang proposed Character-Phonetic based BERT spelling error correction model in their paper [10]. The researchers added novelty to the existing models by changing using BERT based on the structural transformation. They applied the BiLSTM network to detect the locations of the error characters and then used BERT to correct them. The model was compared to traditional error correction and the BERT-Finetune. The proposed model has achieved dominance in performance compared to both models. According to the experiments, it demonstrated a 5% improvement over the traditional approach and a 2.1% increase in performance over the BERT-Finetune model. The model needs improvement in the algorithm in detecting and correcting grammatical error corrections by using the grammatical error correction network to be fully capable of solving real-life error correction and detection problems.

Another research has been conducted by R. Fakhitah and W. M. Wan Zainon about the data cleansing methodologies in Big Data [11]. They highlighted the importance of efficient data cleansing algorithms by mentioning the massive amount of data availability in the industries. The paper also reveals the challenges that need to be overcome during the data cleansing procedures. There were researched several available data cleansing frameworks such as Cleanix, SCARE, KATARA, and BigDancing in terms of the methods they use, their key features, and the approach that they are based on. The study reveals that current methodologies are not sufficient in solving dirty data problems in Big Data applications. They lack one of the five main Big Data criteria such as scalability. Additionally, the data cleansing techniques need to be improved in terms of time efficiency to keep up with the velocity of the Big Data. Moreover, data cleansing should not overwrite the data but be able to reveal the insights that the datasets might give. Therefore, the veracity and value of the data must not be lost during the imputation of the missing data and values. The study reveals that data cleansing in Big Data applications remains an unsolved problem and needs further research before applying existing frameworks because each dataset should be analyzed and considered in terms of the subject domain not just a table of values.

Y. Bai researched the data cleansing method for an industrial application named wireless sensor network based on the data mining technology [12]. In this article, the specific application of the wireless sensor network is studied and analyzed. The experiment used data cleansing technology based on the data clustering model that serves as a core for the replicated data deletion algorithm. The performance of the model is measured based on the accuracy metrics. The experiments have been conducted on the student talent management system with over 800 records. To get more variables in the dataset, the data tables from other sources have been merged to create one dataset with duplicates, which increased the number of records to over two thousand with over two hundred classes. Then the Canopy clustering method was run to detect the duplicated records. The method was compared to another pre-processed duplicated recording deletion algorithm. Although in the experiment the pre-processed duplicated recording deletion algorithm had better accuracy, it was stated the Canopy would have better results

and performance in real-life situations where the number of records is much larger than in the dataset used for the experiment due to the higher recall rate of the Canopy method. The algorithm needs further study with the improvement of the data mining technology as the duplicated records have become common in daily tasks due to the increase of the amount of data in various tables and the need to merge them to create one dataset with all the variables at one place for the modelling.

N. Parmar et al conducted a study on data cleansing in the computer vision domain. They developed a self-attention-based Transformer model to generate images [6]. The research was inspired by the enormous success of using self-attention in textual modelling. The performance of the model was compared to the existing state-of-art in image generation on ImageNet. The model was also used to experiment with image super-resolution with a high magnification ratio. It was able to fool human observers three times more than the current models. The proposed simple generative model with fewer layers but a larger receptive field paves the way for the modelling of more complex images from ImageNet and super-resolution. The model further will be improved to widen the domain of the application other than text and images. The research can be developed to apply the model in the videos and real-time image processing tasks.

Z. Hu and D. Du researched solving an older but actual data cleansing problem – missing data imputation [13]. They developed a new analytical framework for missing data imputation and classification. They tested the model with the heart failure readmission prediction task. The model is based on the GPLVM, which stands for Gaussian Process Latent Variable for the missing data imputation. The GLVM-based model can output the two parameters, namely, mean estimate and uncertainty. Afterwards, a cSVM, which stands for the constrained support vector machine, was used for the predictions. The best SVM classifier was chosen based on the maximal separation margin parameter. The selected model was compared to other classifiers in terms of accuracy and the area under the curve. The model showed an improvement in accuracy by 7% over the existing classifiers such as LR, NB, SVM, and mNB. The downside of the model is its inability to capture and analyze the time-series dataset for the readmission classification.

A. Petrozziello, I. Jordanov, and C. Sommeregger researched the use of Distributed Neural Networks for the missing data imputation in the Big Data [14]. They developed the framework in Spark and tested the model with the real-world dataset for the recommendation system application. The model was compared to univariate and multivariate imputation techniques in terms of accuracy and speed. The speed of the model was also compared to other Neural Network models with simple Stochastic Gradient Descent Loss function. The proposed model used mini-batch Stochastic Gradient Descent, which was faster than the former one. The model achieved high results both in terms of the R2 metric and speed-up tests in the Recommendation System dataset with over 400 hundred samples and with 600 features, while 57 of them contained missing values in various proportions.

A. Folch-Fortuny, F. Artega and A. Ferrer conducted research on the assessment of the maximum likelihood principal component analysis (MLPCA) of missing data imputation which was analyzed in six different datasets [15]. The proposed method was compared to TSR, which stands for Trimmed Regression Score, ML-TSR, projection of model plane (PMP), and PCA in varying proportions of missing data in the test dataset. It was summarized that TSR performed better than other models in medium-high and low percentages of missing values. However, in the case of high percentages of missing values, ML-TSR outperformed MLPCA. It is concluded that TSR should be chosen over MLPCA or ML-TSR because the accuracy of the former over the latter ones in the construction of the missing values was higher.

Despite many studies that have been conducted about the actual pre-processing and data cleansing steps in the dataset to prepare for the modelling, they failed to highlight one of the most actual problems in the real world which is the scalability of the data. Z. Y. Khayyat tried to handle this issue in his dissertation named “Scaling Big Data Cleansing” [16]. In his dissertation, he handles three issues related to the data cleansing of Big Data, which are the computational expensiveness of pre-processing, naïve parallelization of the inequality joins and lastly, skewed distribution of the large errors. He applied a general system named BigDansing that tackles the aforementioned issues and highlighted its pros and

cons. Due to the inability of BigDansing to catch the error discovery, he also introduced other algorithms developed by him IEJoin and Mizan. The former was compared to the BigDansing in terms of speed and scalability, whereas the latter is claimed to be a general-purpose algorithm that handles the data skewness in runtime. Finally, he concluded that Mizan demonstrated 84% improvement compared to other models and algorithms, which is a significant improvement in terms of performance, and this paves the way for semi-automated and automated data cleansing and pre-processing algorithm to be applied in many real-world datasets that would likely to be fed for the machine learning and deep learning algorithms for the modelling and predictive analytics.

3 METHODOLOGY

The current project involves a desktop application created in Python by me, utilizing different libraries like PyQt, sklearn, and matplotlib for data preprocessing, purifying, and analysis. The exploration project included the utilization of datasets, which were additionally exposed to noisiness to evaluate the vigour of the created application. The machine learning strategies utilized in the project included anomaly dealing, duplicate identification, missing data handling, normalization, and standardization.

The main aim of the project was to create and examine the adequacy of the created desktop application for preprocessing and cleansing datasets. To accomplish the initial segment of the goal, which is the improvement of the semi-automated data cleansing application, the datasets were first exposed to preprocessing methods, for example, outlier handling, duplicate data detection, and missing data dealing. Then, at that point, the handled datasets were additionally normalized and standardized to guarantee consistency and likeness of the factors and the elements of those datasets, and the rightness of the outcomes created by the machine learning models trained by them. The subsequent part was accomplished by a wide variety of trials directed with the created application to gauge the presentation regarding precision, reproducibility, ease to the client and time proficiency.

The underlying analysis of the first datasets imported by the client was performed utilizing the Matplotlib library, which empowered the representation of the data and the distinguishing proof of likely examples or patterns as far as uproar and irregularity across the various elements of the dataset. Also, machine learning strategies were used to distinguish and deal with any anomalies present in the datasets.

The modified datasets were likewise exposed to noisy reductions and cleansing, which gave an understanding of the performance of the created application. The outcomes obtained from the analysis of the datasets exhibited the viability of the desktop application in preprocessing and cleaning datasets.

Overall, the project showed the viability of the desktop area application in preprocessing and cleaning datasets. The utilization of machine learning strategies for outlier handling, duplicate detection, missing data handling, normalization, and standardization gave predictable and solid outcomes. The project results can be applied in different exploration fields, especially in data analysis and pre-processing.

3.1 Architectural Design

As a result of this study, I developed a desktop application utilizing Python programming language and its various libraries for data preprocessing, cleansing, and analysis. Due to the complexity of the application steps before actual programming, I developed the general architecture of the project which consists of several stages. Figure 1 demonstrates the general architecture of the developed application providing an overview of the implementation steps through which the users interact with the program and the dataset undergoes the changes. The first block of the figure shows that the users are required to input the dataset, which is then processed by the Python program. Then, the program returns three outputs, the ultimate output of the program is the cleaned dataset, but before that, a dashboard is produced to visualize the noisiness of the dataset. The dashboard with the charts and the visualization of the data noisiness enables the user to make informed decisions on the cleaning procedure and techniques of the dataset. If the user wishes to interfere with the automatic logic of the program and make their own decision in choosing the cleaning methods, they can do so. The third output is the log

file that logs the modifications made to the dataset, the information, and the debugging warnings about the processing steps.

Diving into the details, the developed application produces a log file listing all the changes applied to the dataset and all the procedures that the program has gone through with the precise timestamp, with the addition of the type of the log such as warning, error, or debug. This log file can be useful for tracking the cleaning process and ensuring transparency and reproducibility of the results. In addition to the log file the dashboard provides valuable insights into the dataset and its cleaning process, enabling the user to make informed decisions based on the analysis. Both generated outputs, namely, the produced dashboard and log file provide the user with the flexibility to observe the current state of the dataset and the changes and modifications applied to that dataset.

Additionally, the program was structured in terms of programming implementation and technical methodologies. I used a wide variety of libraries and packages for the project and application development. In terms of the architectural structure and design of the technical parts of the semi-automated data cleansing tool, which is not shown in Figure 1, I programmed a Jupyter notebook script with the help of the Python libraries which includes PyQt, sklearn, matplotlib, pandas and numpy. PyQt is a library which is a set of Python bindings for the Qt application framework. Sklearn is a popular library for machine learning, which provides a range of tools for data preprocessing. Matplotlib is the main tool used for dashboards and data observation, which is a Python data visualization library for plotting. The machine learning techniques employed in the study included outlier handling, duplicate detection, missing data handling, standardization, and normalization. These techniques enabled consistent and reliable results and provided a robust framework for data preprocessing and analysis.

In conclusion, the developed desktop application utilizing Python programming language and its various libraries for data preprocessing, cleansing, and analysis demonstrated effectiveness in producing a cleaned dataset. The dashboard visualization and log file produced by the application provide valuable insights into the dataset and its cleaning process, enabling informed decision-making by the user. The study findings can be useful in various research fields, particularly in data analysis and processing. Further research can be conducted to evaluate the effectiveness of the developed application on real-world datasets.

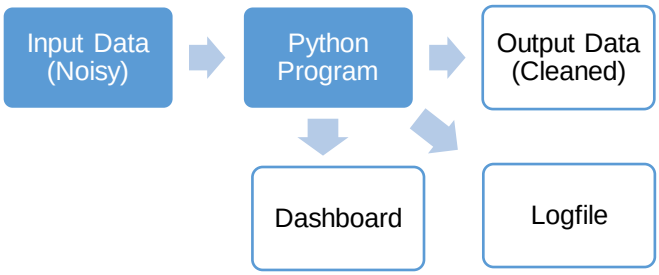


Figure 1: The process flow of the semi-automated data cleansing tool

Figure 2 provides a more detailed overview of the system decomposition of the project, outlining the exact procedures that are accomplished within the Python program. The program is designed to not only detect the noisiness but also handle it in a systematic and effective manner. Specifically, the program includes several procedures such as handling duplicates, outliers, missing data, and categorical attributes.

Handling duplicates is a crucial step in data preprocessing since it can impact the accuracy of the results. The Python program includes a procedure to detect and remove duplicates, ensuring that only unique data points are considered in subsequent analyses. Additionally, the program incorporates outlier handling techniques, which involve identifying data points that fall outside of the normal range and either removing them or adjusting them to improve the accuracy of the results.

Furthermore, the program includes missing data handling techniques that address the issue of incomplete data. These techniques can include imputation, where missing data is estimated or replaced with reasonable values based on the available data. The program also handles categorical attributes, which are variables that take on discrete values rather than continuous ones. Categorical attributes require special handling, and the Python program includes procedures to effectively preprocess and transform these variables.

Overall, the Python program is designed to provide an automated solution to the data cleansing and preprocessing challenge. By incorporating these procedures, the program can effectively identify and address issues such as duplicates, outliers, missing data, and categorical attributes. This ensures that the final cleaned dataset is ready for analysis and modelling and can provide accurate and reliable results.

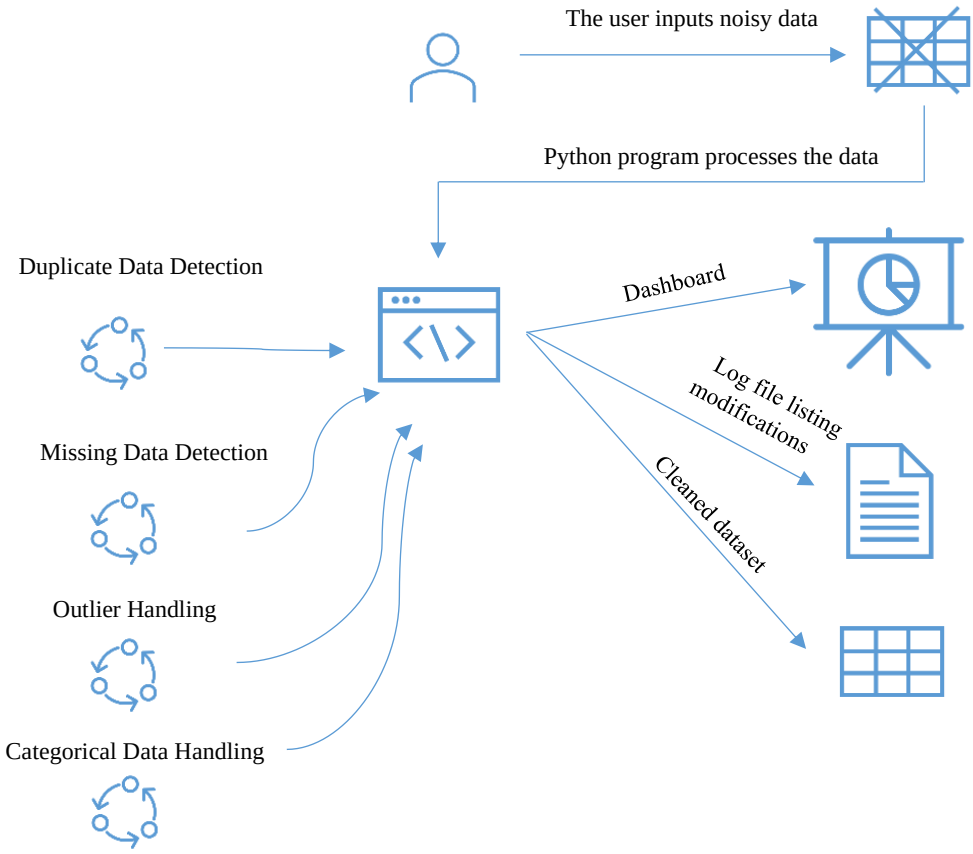


Figure 2: Decomposition of the project architecture

The system first prompts the user to input the original dataset, the dataset is read by using the pandas read_csv() method, which takes as input the path to the input file. Then the program processes the input dataset and converts it to the pandas data frame. The pandas data frame is like an Excel tabulated format in rows and columns, each cell of which contains a single value. However, the pandas data frame cell may contain not only a single data frame but also a list of values. Figure 3 visualizes the input system decomposition of the project.

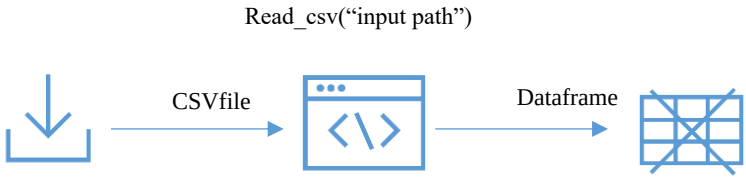


Figure 3: Input subsystem decomposition

The next is the duplicate handling subsystem shown in Figure 4, which is handling upon user request. The users are demonstrated the number of duplicates both in each column and in the whole dataset with the help of the pandas duplicated() method. The method can take input the name of the column in which we are looking for duplicates. To count the number of duplicates either in the whole dataset or in a column we use the sum() method, which sums the values returned by the duplicated() method. As the method returns True and False as a value for duplicated records summing them will sum the number of ones which is in fact the number of True values. By default, the duplicated records are deleted from the dataset. However, when it comes to the duplicated values in a column, if the user requests the program to delete the duplicates they are dropped by using the pandas drop_duplicates() method. This method takes the column as input and returns the new dataset with the deleted rows that contained duplicated column values. This method keeps the first occurrence of the duplicated value and deletes all other rows containing the same value under the same attribute.

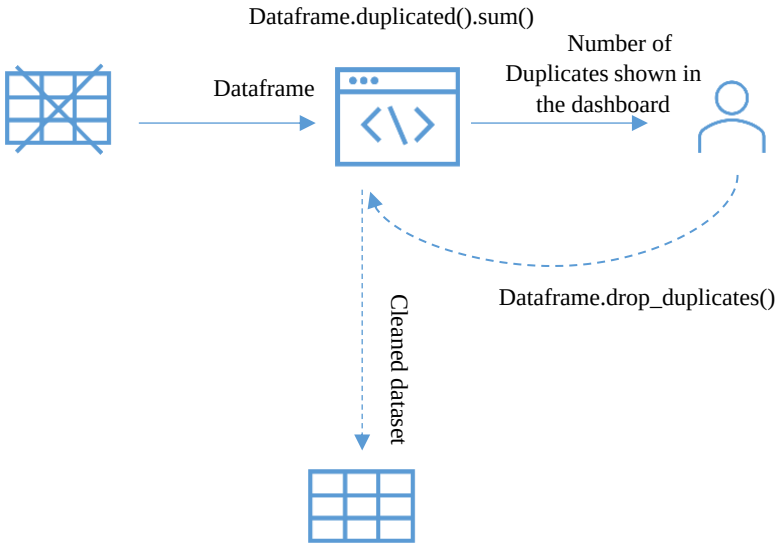


Figure 4: Duplicate handling subsystem decomposition

The missing data handling subsystem is more complex and contains many methods and techniques amongst which the user can choose to impute the empty values. The users have a wide variety of choices for missing data imputation and handling techniques. The missing data is detected by using pandas isna() method. It is same as duplicated() method returns the True and False values based on the missing data in the data frame. If the value is missing it will return True and summing it will give the total number of missing values as the True accounts for one in programming. Based on the user's request the missing values can be dropped, ignored, or imputed by mean, median, KNN (K Nearest Neighbours),

Regression or Iterative Imputation. Figure 5 demonstrates the visual representation of the missing data handling system. The most basic functions are fillna() and dropna(). The pandas fillna() function takes as input the value to be imputed for the missing values. It can be used to fill the missing values with any value prompted as input. In this project, it is used to replace the missing values in statistical numbers obtained from the dataset specific to each column such as mean, median or mode.

The missing values can also be dropped by using dropna() method. It returns modified dataframe that does not contain the rows containing the missing value. The method can further be modified to drop rows or columns only if all the values in those rows or columns are missing. Alternatively as shown in Figure 5 the missing values can be used by more sophisticated machine learning techniques such as Imputer(). It can be either simple to impute mean, median or mode or modified to impute using KNN or regression.

In this project, the missing values are treated separately in terms of column data types. The missing data handling techniques differ if the attribute is in continuous, categorical or datetime format. This is because missing categorical variables cannot be imputed using statistical values such as mean and median. Additionally, the regression imputation differs, for this reason, depending on the data types of the attribute. The missing values of the numerical can be imputed by using Linear Regression, whereas the missing values of the categorical attributes are imputed by using Logistic Regression instead.

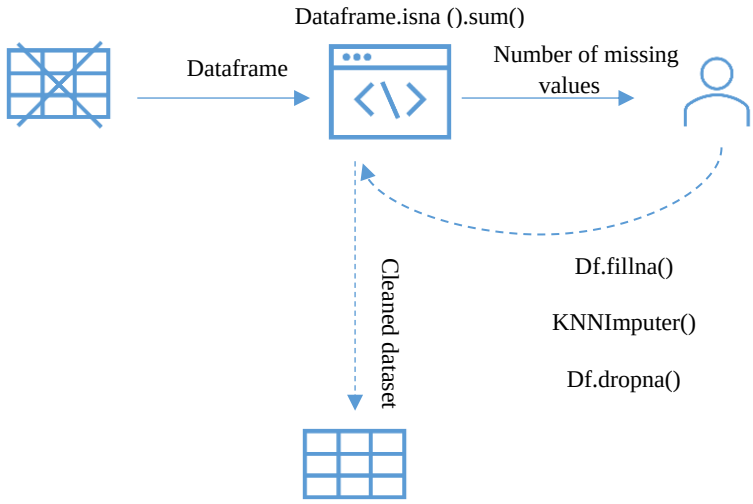


Figure 5: Missing data handling subsystem decomposition

Handling outliers is a critical component of noise reduction in a dataset. Outliers are data points that deviate significantly from most of the other data points and can have a considerable impact on the results of data analysis. Outliers can arise due to various reasons such as measurement errors, data entry errors, or even genuine cases of extreme values. In this project, the outlier handling system is employed to detect and handle outliers in a systematic manner.

I used the Interquartile Range method in a Python program to identify and remove outliers. Interquartile Range is a measure of variability based on the range between the first and third quartiles of a dataset. Therefore, I calculated the Interquartile Range as the difference between the 75th and 25th percentiles of the data. Any data points that fall outside of a certain range from the median are considered outliers. In my Python program, I made use of the IQR values with outlier_param=1.5 to identify potential outliers and calculate the corresponding outlier parameter.

By default, the Python program sets the outlier parameter to 1.5, but this value can be modified based on the user's requirements. If a data point falls outside of the calculated range, it is considered an outlier and is either removed or adjusted to improve the accuracy of the results. This process can be repeated until all outliers have been addressed, and the data is deemed to be clean and ready for analysis.

Additionally, the values that are out of the desired range that appear in the dataset due to either human error or system malfunction can be trimmed out and filtered upon the request of the user both in terms of the minimum and maximum values. Figure 6 demonstrates the visualization of the outlier handling procedure in the system. One of the techniques for trimming the ranges in the data frame is using the pandas clip() method, which takes as input the lower and the upper ranges, with which the values in the selected attribute will be limited as a threshold. Another technique for this is by using pandas .iloc functionality that is effectively used for data frame slicing. This can be used not only to trim the values but also the filter out those values from the dataset and delete them.

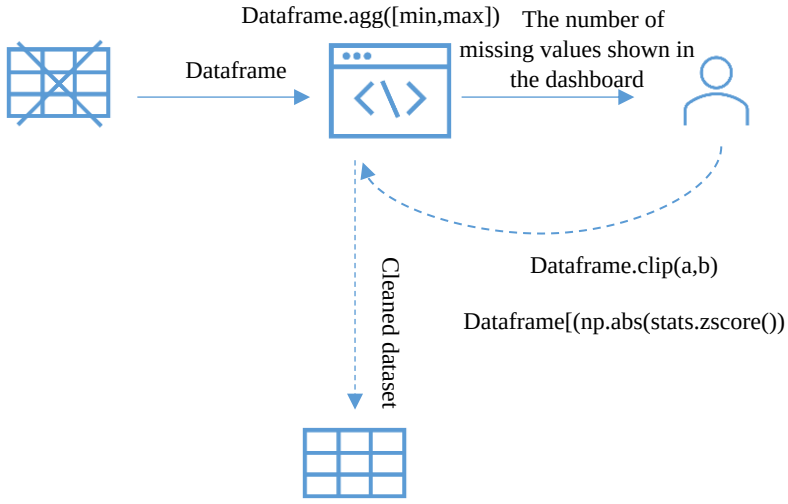


Figure 6: Outlier handling subsystem decomposition

The last part of the system is the output, which consists of three parts namely, the dashboard, the log file, and the cleaned dataset. The dashboard is created after the original dataset has been uploaded by the user and demonstrates the noisiness of the dataset. The visualization is intended to help the user to make their decision about their choices in the cleaning techniques and methodologies in case they would like to choose them manually. Additionally, the visualization is important to inform and persuade the users about the modifications made to the original dataset automatically based on the program logic and pipeline. The dashboard is created using Python's matplotlib package, which contains a wide variety of functions for plotting line plots, bar plots, pie charts, histograms, correlation plots and heat maps. Moreover, the log file produced by the program is a crucial component for the reproducibility and transparency of the data-cleaning process. It contains a detailed list of all the changes and procedures applied to the dataset. The log file also includes the precise timestamp of each change or procedure, which is important for tracking and monitoring the data-cleaning process. This can be helpful for quality control purposes and for identifying potential issues or errors that may have occurred during the data cleaning process. Furthermore, the cleaned dataset is the final output of the program, which is a dataset that has been pre-processed, cleaned, and modified based on the program's logic and pipeline. This dataset is ready for further analysis and modelling, and it can be exported to various formats, such as CSV or Excel, depending on the user's preference. Overall, the output of the system provides the user with a comprehensive and transparent overview of the data-cleaning process and enables them to work with a cleaned dataset that is ready for further analysis and modelling.

Figure 7 demonstrates these three outputs generated by the program. The last two are generated after the cleaning process starts. The log file is generated alongside the cleaning procedure as it lists all the steps, changes and data preprocessing functions applied to each column and the dataset. It writes down as a separate row the timestamp, the type of information (such as warning, info, or debug) the log

contains and the actual information. The log file is created using Python's logger package. The logs are created by using logger.info(), logger.debug(), and logger.warning() methods, which means information, debugging and warning logs, respectively.

The final output that the program produces is the cleaned dataset which is saved as a separate CSV file in the same directory as the input file. The CSV stands for Comma Separated File, which is widely used as a structured dataset input for modelling due to the high speed of saving and the small file size, which makes it an ideal choice for saving a large, structured dataset as a CSV file. The CSV files can be opened in a Microsoft Excel program as a typical XLSX file and be further observed, edited, or formatted. The cleaned dataset is saved by using pandas to_csv() method, which takes as input the path to the file location where it should be located.

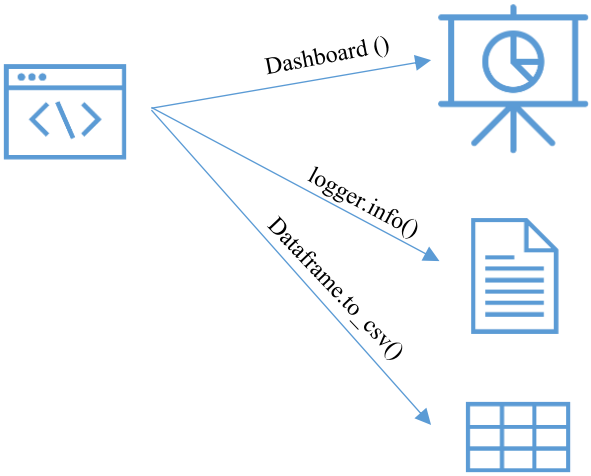


Figure 7: Output subsystem decomposition

Initially, the project was written as a function that takes arguments for different modes of preprocessing techniques for various issues in the dataset such as outliers, missing data, and duplicates. I will be using Python libraries for data cleansing processes. The missing data will be imputed using mean, mode and median, KNN and regression and the duplicates will be detected using Pandas. The results will be shown via visualizations of the statistics about the detected errors in the original dataset. Then, the imputed values will be compared to the true values based on the R2 score. The missing data experiment results were shown in a tabulated form where columns showed the methods for the data imputation and the values showed the accuracy of the model using the cleaned dataset as an input. The initial project was written in Python programming language in a Google Colab notebook. I used Python libraries such as NumPy, pandas, matplotlib, and sklearn for data cleansing and plotting the visualization of the statistics about the changes and transformations made to the original dataset. Additionally, all the changes and modifications were logged in a log file with the predefined log types such as INFO, DEBUG and WARNING. The information and the logs are listed by their respective timestamp of occurrence.

I obtained the original clean dataset from Kaggle [17]. It is used for customer churn prediction in the banking industry and consists of customer demographic and transaction data. Overall, there are 11 input attributes and 1 target input, customer churn. Figure 8 described the general view of the numerical features of the original synthetic dataset by using the pandas describe() method.

The customer_id is the unique id column, which will not be used during the modelling and is preferable to be dropped. This also shows the outliers in the dataset and differences in scales. The difference between the mean and 50% percentile of the balance is proof of the possible outliers.

Additionally, the values are different in terms of their absolute value range, which means that the dataset should be normalized before modelling. Otherwise, the prediction of the target value churn will be artificially dependent on the values with higher absolute values due to the modelling weights even though they directly do not affect the final target value.

	customer_id	credit_score	age	tenure	balance	products_number	credit_card	active_member	estimated_salary	churn
count	1.000000e+04	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.00000	10000.000000	10000.000000	10000.00000
mean	1.569094e+07	650.528800	38.921800	5.012800	76485.889288	1.530200	0.70550	0.515100	100090.239881	0.20370
std	7.193619e+04	96.653299	10.487806	2.892174	62397.405202	0.581654	0.45584	0.499797	57510.492818	0.40276
min	1.556570e+07	350.000000	18.000000	0.000000	0.000000	1.000000	0.00000	0.000000	11.580000	0.00000
25%	1.562853e+07	584.000000	32.000000	3.000000	0.000000	1.000000	0.00000	0.000000	51002.110000	0.00000
50%	1.569074e+07	652.000000	37.000000	5.000000	97198.540000	1.000000	1.00000	1.000000	100193.915000	0.00000
75%	1.575323e+07	718.000000	44.000000	7.000000	127644.240000	2.000000	1.00000	1.000000	149388.247500	0.00000
max	1.581569e+07	850.000000	92.000000	10.000000	250898.090000	4.000000	1.00000	1.000000	199992.480000	1.00000

Figure 8: The overview of the numerical variables of the input dataset

Next, the project was developed as a desktop application which accepts the input dataset as a CSV file from the user and then preprocesses the dataset, after which the visualization page opens where the user can see information and statistics about the dirtiness of the whole dataset and column-wise. On the visualization page, I analyzed and demonstrated to the user the three main features of data cleansing, which as missing data, outliers, and duplicates. After the visualization page, the dataset is processed and cleaned both based on the user input and automatically depending on the modes of the selections.

The architectural design of the application is shown in a structured form in the diagram shown in Figure 9. In general, there are two modes for each of the data processing techniques to be selected, namely, Auto and Manual. In addition to that the cleaning techniques are separated into groups according to the column types. After the cleaning procedures are accomplished on each column, the whole dataset can be normalized and standardized to prepare fully the final cleaned dataset to feed to the machine learning model for further predictive analysis.

The mode refers to the method of operation for a machine learning model. In the Auto mode, the program automatically selects the appropriate settings for the given data, while in the Manual mode, the user manually selects the settings for each of the columns.

Column type refers to the data type of a feature in a dataset. The three main column types are categorical, numerical, and datetime. Categorical data consists of discrete values that represent categories or labels, while numerical data consists of continuous or discrete numeric values. DateTime data represent values in the pandas timestamp and datetime format.

Standardization is a technique used to transform numerical data to have a mean of 0 and a standard deviation of 1. This can help make the data more comparable and easier to work with. The two main methods of standardization are the Robust Scaler and Standard Scaler. The Robust Scaler is less sensitive to outliers, while the Standard Scaler is more commonly used. The model in Auto mode implements the Robust scaler if there are many outliers detected in the dataset. Otherwise, it automatically scales the data with the Standard Scaler. Additionally, as standardization is inevitable for modelling in real-life scenarios because most machine learning models are designed for standardized datasets. Moreover, there are usually datasets with numerical values represented in different scales and comparing them without standardization leads to inaccurate model predictions and estimations.

Normalization is a technique used to scale numerical data to a specific range. The two main methods of normalization are the MinMax Scaler and MaxAbs Scaler. The MinMax Scaler scales the data to a range between 0 and 1, while the MaxAbs Scaler scales the data to a range between -1 and 1. This data-cleaning procedure is extremely useful when the data across the columns differ a lot in terms of their absolute values. Without normalization, the model would produce inaccurate results due to the larger weights applied to the attributes with larger absolute values. Even though actually the larger absolute value does not mean a higher impact on the target value and thus, should not have more effect on the prediction than other attributes with smaller absolute values.

It's important to note that not all data needs to be standardized or normalized, therefore by default the data is standardized and normalized in our project. However, it can be changed and skipped upon the user's request, because it depends on the problem to solve and the type of data. For example, decision trees and random forests do not require standardized or normalized data, while some models like neural networks require it for better performance.

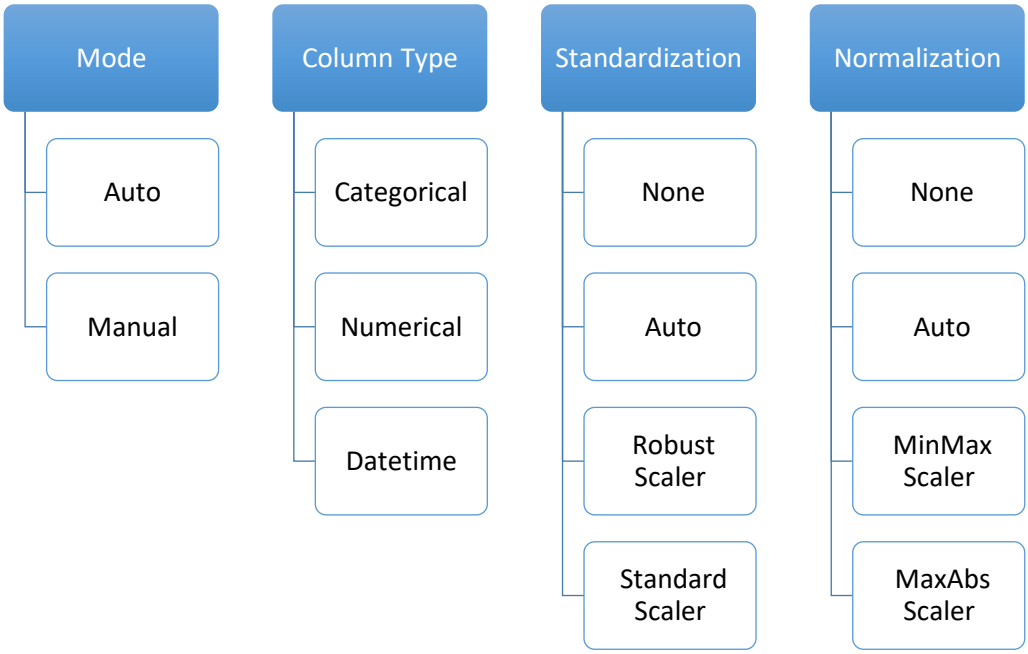


Figure 9: Functional level decomposition of the system

A detailed architectural description of the functions and techniques applied to the original dataset in a column-wise order is described in Figure 10. The given description provides a set of options for handling different types of data in a dataset. The three types of data considered are categorical, numerical, and datetime data. For each type of data, the description outlines different approaches that can be taken to handle missing or duplicate data, drop irrelevant columns, and encode the data. Additionally, the description provides options for handling outliers for numerical data.

For categorical data, the options for handling missing data include imputing with the mode or using logistic regression or KNN for imputation. The encoding type can be one-hot encoding or label encoding. For handling duplicate data, there is an auto option and a drop option. There is also an option to drop irrelevant columns.

For numerical data, missing data can be imputed with the mean, mode, median, linear regression, or KNN. Outliers can be handled using winsorization, trimming, or dropping them altogether. There are also options for handling duplicate data and dropping irrelevant columns.

For datetime data types, the options are like those for categorical data, except for handling missing data with the mode, logistic regression, or KNN. The columns are automatically detected and converted into pandas datetime columns before the processing steps. To do this I used the to_datetime() function, which can take as input the data frame with given column values which we would like to convert. Besides, the function takes a second input the format from which it is needed to convert the data frame feature from.

Overall, the description provides a range of options for handling different types of data, and the approach chosen will depend on the specific dataset. The provided options are the final version of the developed desktop app. However, the process, pipeline and functions can be further developed to fully

automated processes and to consider all the possible real-life cases where specific cleaning techniques may work, and others may not work as expected and produce the expected performance.

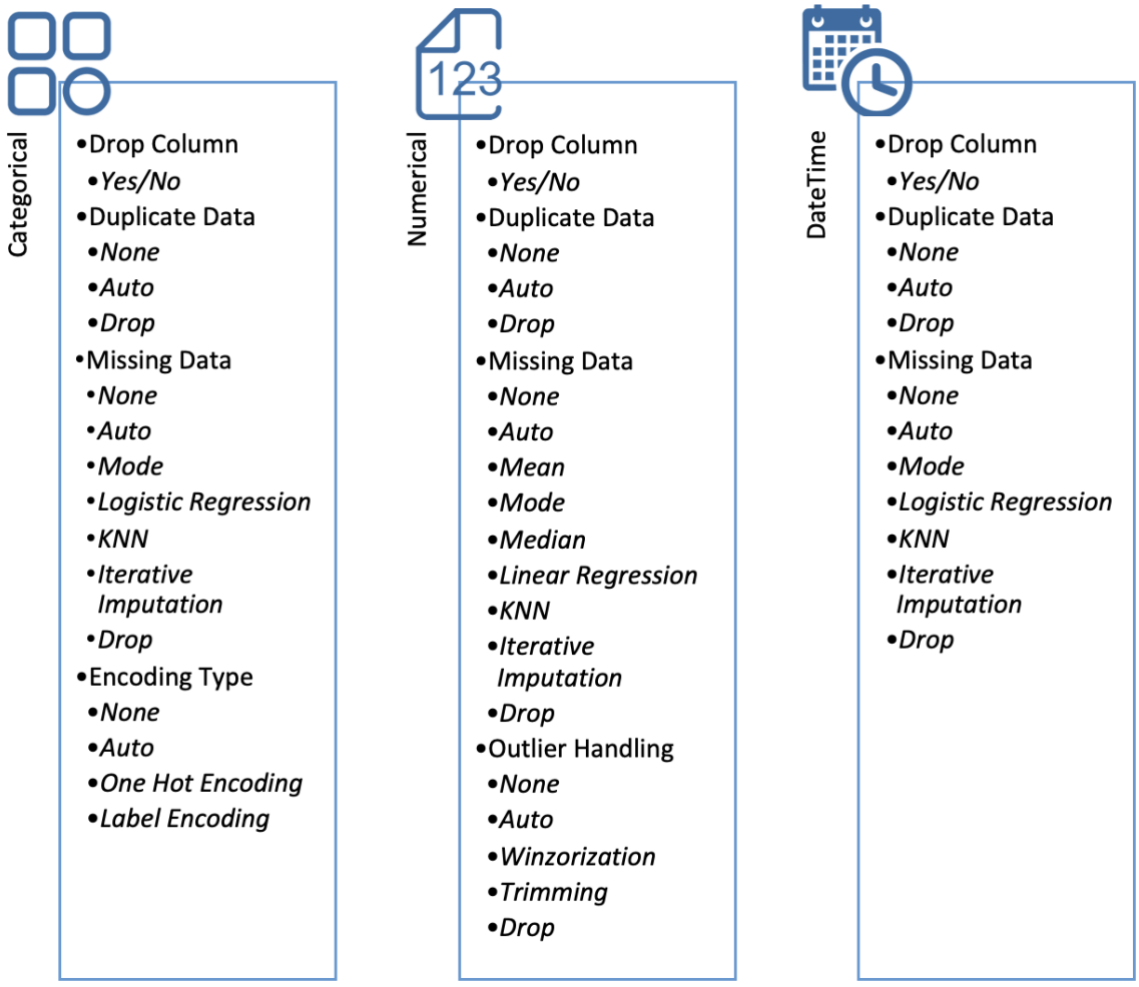


Figure 10: Functional description of the system on a column level

3.2 Automatic Logic Algorithm

The Auto mode is a special trait of this project which makes the data cleaning process semi-automated. It chooses the predefined functions on a rule basis developed after careful consideration and research of other missing data techniques. The rule basis is performed depending on the different features of the dataset and attribute values. Algorithm 1 describes the automatic choice of the standardization algorithm. The technique for standardization depends on the number of outliers and the skewness of the dataset. If the share of the outliers and the skewness is large RobustScaler is applied to the dataset, otherwise by default the StandardScaler is chosen. I applied threshold values for the determination of the standardization algorithm. First, the algorithm checks if there is a skewness in the dataset, if there is a skewness greater than 1 the kurtosis is calculated. If the absolute value of the kurtosis is greater than 3 then there are outliers and therefore, RobustScaler is performed, otherwise, StandardScaler is applied to the dataset.

ALGORITHM 1: The Standardization Auto Mode Logic

If the ABS(skewness) $>$ 1:

Calculate kurtosis

```
    If ABS(kurtosis) > 3:
        Perform RobustScaler
    Else:
        Perform StandardScaler
Else:
    Skip standardization
```

Algorithm 2 shows the basic logic for the selection of the scaler for the normalization of the dataset. If the difference between minimum and maximum values in the dataset is higher than the threshold, which is 100 in this project, normalization should be performed. Then the maximum and the absolute values of the maximum are calculated, if both are less than 1 the MinMaxScaler would perform better, otherwise MaxAbsScaler is chosen as a normalization technique.

ALGORITHM 2: The Normalization Auto Mode Logic

```
If data frame MAX – MIN > 100:
    Perform Normalization
    If  ABS( MAX) <= 1:
        Perform MinMaxScaler
    Else:
        Perform MaxAbsScaler
Else:
    Skip Normalization
```

Below is shown decision process for the encoding algorithm of the categorical variables. Algorithm 3 states that if the number of categories is larger than 10 and less than 20, then automatically LabelEncoder is performed, if the number of the categories is less than 10, OneHotEncoder is chosen for the conversion of the categorical features into the numerical ones for the modelling. Otherwise, if the number of categories is greater than 20 then encoding is skipped.

ALGORITHM 3: The Encoding Auto Mode Logic

```
If the number of categories < = 10:
    Perform OneHotEncoder
Else if the number of categories<=20:
    Perform LabelEncoder
Else:
    Skip the Encoding
```

The last algorithm is for the determination of the missing data imputation techniques. It was experimentally determined throughout the research that if the share of the missing values is greater than 30% of the data, then it is not recommended to impute as the imputation accuracy will be low. Therefore, in this algorithm, I determined a maximum threshold of 30% for the automatic imputation of the missing values with the regression algorithms, otherwise the records containing missing values are dropped.

ALGORITHM 4: The Missing Data Imputation Logic

```
If the number of missing values in column >= 30 %:
    Drop records containing missing values
Else:
    Perform Regression Imputation
```

The other two techniques, which are outliers and duplicate handling are ignored in a column-level logic in the Auto mode, because, both outliers and duplicates are cleaned from the whole dataset, by using standardization and duplicate record dropping. In other words, the outliers are automatically handled at the end of the data cleansing during the standardization, via RobustScaler, whereas the duplicates are cleaned from the dataset before the preprocessing. The Adjust class in the project drops duplicated records from the dataset automatically without the user request and interference, because it is not acceptable to have two records with repeated values across all the features.

3.3 Theoretical Foundations

In this section, I described the theoretical background needed to implement the project. It briefly covers the logic behind the algorithm and techniques and then describes how they were used in the automation process of the project as well as its evaluation in terms of performance. Additionally, the section highlights the advantages of the techniques over the other ones, which is crucial in the selection steps of the algorithms. The techniques with the highest performance and the most benefits are used in the auto mode of the project, whereas the others can be applied upon the user's request through the GUI.

3.3.1 KNN Imputation

The k-Nearest Neighbors algorithm is a lazy learning algorithm that assumes that the K number of nearest objects are like each other. It is a supervised machine-learning algorithm that is widely used in classification. Therefore, it trains based on the K nearest neighbours of the missing values and detects them, after which the missing values are filled with that data.

KNN algorithm estimates the distance between the objects by using the Euclidian distance, however, it can also work with Manhattan, Chebyshev or Mahalanobis distances based on the application field. Once, the distance metric is selected the algorithm calculates the K nearest neighbours of the missing value based on the distance metric of that algorithm. The value of the K is not fixed and can also be tunable by a cross-validation algorithm.

The imputed value is based on the feature type with the missing values, such as numerical, ordinal, or categorical. This defines whether the mean, mode or median of the K nearest values is imputed in place of that missing value. Therefore, this algorithm is advantageous over other simple imputation techniques. First, this is a non-parametric algorithm, meaning it does not need any initial parameters such as distribution fed to it as input about the dataset. Instead, it generalizes the model after it is required to predict the label based on the new input data. Secondly, it considers the local structure of the dataset, which is extremely useful when the dataset is large and high-dimensional which makes the connections between the features hard to understand. Lastly, it is capable of imputation both categorical and numerical values, which makes it the best choice when the information about the dataset is unknown, or they are of mixed types.

In practice, the KNN algorithm is mixed with other algorithms such as mean value imputation. After the mean values of the available variables are imputed KNN is performed on the remaining missing variables. The KNN algorithm can also be further improved and adapted based on the dataset to perform the imputation that is much better than the traditional plain imputation techniques in terms of the imputation accuracy of the values.

Overall, the robustness of the KNN algorithm, its simple non-parametric approach and its tunability make it popular and one of the best choices during the missing data imputation tasks. Therefore, it is

one of the default algorithms used in this project in Auto mode, when the input from the user is not provided about the selection of other algorithms.

3.3.2 Regression Imputation

Regression is a statistical algorithm that is based on the relationship between the dependent and independent variables. The model fits the trained data and learns the mathematical dependence between variables, which then by using the trained weights and coefficients predict the output of the new variables or test dataset.

There are generally two types of regression, namely logistic and linear regression, which are applied based on the nature of the predicted variable. Linear regression is used for the prediction of the continuous variables, whereas logistic regression is used for classification purposes when the output variable is in binary format.

The regression model has many benefits in terms of the ability to handle the whole dataset and learn the relationship between the different types of variables. The resulting model can be effectively used to impute the missing variables, which are fed to the model as a test dataset, whereas the non-missing values can be used as a training dataset.

The algorithm has also some downsides, which need to be considered during the application of the algorithm as a part of the semi-automate data cleansing project. The first issue about the regression algorithm is the fact that it assumes the randomness of the nature of the missing values. Another issue with this algorithm is the possibility of bias and overfitting, which will ultimately lead to the inaccuracy of the predicted variables.

Overall, this algorithm is a powerful and useful technique for missing data imputation. However, considering the downsides of the regression, it is combined with the KNN algorithm in this project for the automatic imputation of the missing values of different types in terms of format and quality.

3.3.3 Statistical Imputation

Statistical imputation is about using mean, mode, and median values of the non-missing data to impute the missing ones as constants. While this technique is not used as an automatic imputation algorithm, it is widely used among data scientists due to its simplicity and speed performance.

The main benefit of this technique apart from the simplicity and speed is the ability of the technique to cover and preserve the overall distribution and the variability of the data after the imputation of the missing values. Additionally, statistical imputation can be used for both missing categorical and numerical data.

The downsides of this technique are due to pure statistical reasons such as the distribution of the variables. First, the imputation of the mean and median to all the missing cells would potentially reduce the variance of the data, which will affect the analysis and decision-making process negatively. Moreover, mode imputation in real-life scenarios may not be the correct technique in terms of statistics as there can be several modes at the same time and the imputation of one of them would lead to potential mistakes both in analysis and modelling.

3.3.4 Iterative Imputation

Iterative imputation is a recursive technique for filling in missing values. It is based on mean and regression as it imputes the missing values for the first time. Next, it repeats itself iteratively to impute the values until the convergence or the maximum number of iterations is reached.

This is a powerful technique due to its ability to produce accurate results even when the nature of the missing values is not random. However, this algorithm is computationally expensive and may not work as expensive for complex and large datasets. Additionally, the algorithm needs to be tuned the parameters such as the maximum number of iterations, which influences the final convergence of the algorithm and not properly selected parameters would lead to failure to converge and produce inaccurate values in place of the missed data.

3.3.5 Dataset Metrics

The metrics are important values to understand the dataset in terms of the range of values, variance, outliers, and standard deviation. In this project, I used several of these parameters to describe the dataset and automatically detect which algorithm to use for data cleaning and pre-processing. Below are listed those metrics:

- Interquartile range
- Standard Deviation
- Range
- Kurtosis
- Skewness
- Mean
- Mode
- Median

When it comes to the application of these metrics in this project, each of them was used in separate parts of it. Interquartile range and standard deviation were used to detect and compute the number of outliers in each of the attributes in the dataset. The range, kurtosis and skewness are the parameters used in the automatic data-cleansing decision-making process. First, the range is calculated by subtracting the minimum value from the maximum in the dataset to determine if the dataset needs normalization. If the range of the dataset is large or passes a certain threshold, then normalization should be applied. In this project, a threshold value of 100 is chosen for the value of the range, if the value is less than the normalization is skipped.

Another parameter for automated data cleansing is skewness. If the absolute value of the skewness is large, which is 1 in our case then normalization should be applied to the dataset. The skewness of greater than 1 or less than -1 is the indicator of the non-normal distribution and asymmetry in the dataset. Another parameter that is important in standardization is kurtosis, which shows if the dataset has longer tails and a flatter peak or vice versa. If the absolute value of kurtosis is larger than 3, which is an indicator of the outliers the Robust Scaler is used, otherwise the Standard Scaler is used for the standardization.

The mean, mode and median are used for the missing data imputation upon the user's request. Since this project is about semi-automated data cleansing, the user may request for the statistical missing data imputation in some cases, therefore they are added as additional functionality. The mean of the values is the average, which is calculated as the sum of all the values divided by the number of them. The mode is the most frequent value in the dataset. There can also be several modes which make the data imputation using this parameter inaccurate. The median is the middle value if all the values are placed in a sorted format, if there are two values in the middle then the mean of them is considered the median of the dataset.

3.3.6 Random Forest Classifier

Random forest classifier is the algorithm that I used for the extrinsic evaluation of the project. It is the supervised machine-learning algorithm that is used for the prediction of categorical variables. It is based on the idea of using different parts of the dataset for training several decision trees. The output of these decision trees is then combined to produce the final output for the classification. As the dataset is about customer churn, a random forest classifier is used for the prediction of the churn based on the dataset provided by the semi-automated data cleansing tool and then compared them to the original clean dataset to evaluate the performance of the data cleaner.

The decision to use the random forest classifier for the extrinsic evaluation of this project is based on the advantages in terms of the algorithm and performance. The first reason for choosing a random forest classifier is its robustness to the parameters and format of the dataset. It can perform equally well both with categorical and numerical values. Secondly, it is not prone to overfitting due to the nature of its algorithmic structure as an ensemble method. Lastly, the random forest classifier performs

extremely well with default settings which makes it fairer to evaluate it with different datasets without the need for further parameter tuning.

Although the model has also some disadvantages in terms of computational expensiveness, interpretability, and other factors, it does not affect the comparison of the same dataset. The performance metrics of the model described below are compared as a deciding factor in the comparison of the cleaned datasets which were initially added noisiness in varying ratios.

3.3.7 Performance Metrics

There are various performance metrics that can be used for the evaluation of the machine learning model such as accuracy, precision, recall, and f1-score. Accuracy is the proportion of how many predicted values are correct, which is misleading when the dataset is imbalanced and one of the classes dominates in terms of the number of records over the other.

Precision is evaluating how precise the model is, by dividing the predicted true positives by the number of all predicted positives. Ideally, the model should detect all the positives in the dataset, therefore, the more closely the values of precision to 1 the better the model senses the positives.

The next metric is called recall, which is used when it is unacceptable to miss the positives, therefore it is calculated by dividing the true positives by the number of all the positives in the dataset.

F1-score is on the other hand combination of precision and recall, which is a more robust metric in the determination of the general performance of the classifier. Therefore, F1-score is used as a performance metric to compare the output of the model by feeding it with the same datasets with different initial noise ratios.

3.4 Experiments

I tested the program with varying percentages of the missing data. The clean input dataset was processed synthetically to delete values from certain columns in proportions of 10%, 30%, 40%, and 60%. Then the missing values were imputed by KNN, regression and other statistical methods. After this, the cleaned dataset was fed to the random forest classification model to predict customer churn. The performance of the different inputs will be compared using the accuracy metrics.

Furthermore, the imputation accuracy will be tested based on the different missing data ratios across the dataset features. Specifically, the R2 score is used to compare the predicted imputed value with the actual value. To do this the n random values were deleted from the randomly selected dataset attributes. The original dataset contained zero missing attributes, which allowed it to have a baseline value to which the model's missing value prediction can be compared. This was repeated four times with different ratios per column. The missing values were imputed by using KNN imputation techniques, which is experimentally one of the best techniques for missing data imputation.

The next part of our experiment was the comparison of the performance of the model when the input dataset is cleaned manually, automatically or used as it is without using the semi-automated data cleansing tool. The experiment helped us to determine the effectiveness of the automation and the chosen pipeline for data cleansing and to understand the importance of the data preprocessing steps before performing any machine learning modelling for classification, clustering, and regressive prediction.

The model was tested with two different datasets obtained from Kaggle. This helped us to understand and estimate the reproducibility of the semi-automated data cleansing tool. Reproducibility is an important factor in machine learning modelling because one technique may work extremely well and show high performance with one technique and pipeline and might not produce the same performance with another dataset. Moreover, even the same dataset might be fed to two different machine learning models and cleaned and preprocessed the same for each of the models, but the performance of one model might be a lot less than the other one. Although the difference in performance between the models may be also due to the architectural design and mathematical base of the algorithm, the format and structure of the dataset also pave the way for the varying accuracies in models. Therefore, in our case,

we compared the same datasets, with different cleansing techniques to the same model and compared the same metrics qualities in each of the produced outputs.

Another metric that we experimented with to measure the effectiveness and success of the semi-automated data cleansing tool during the research was the time effectiveness of the automated data cleansing tool. As it is known manual data cleaning and preprocessing requires writing a script to read the data, visualize and observe the statistical features of the dataset. In addition to this, the user is required to find, research, write and experiment with each of the pre-processing techniques to observe which one of them works better in terms of the modelling performance. It is also a known fact that experimenting with different functions requires understanding the input, syntax, and output of each of them, which is an extremely tedious process and time-consuming process. Therefore, we measured the final project in terms of the average time required from uploading the original dataset to the semi-automated data cleansing tool to the generation and saving the cleaned dataset that is ready to feed to any machine learning model.

Overall, the experiments tested a semi-automated data cleansing tool's effectiveness in preparing datasets for machine learning models by comparing the performance of different cleaning techniques on two Kaggle datasets. The experiment involved synthetically deleting values from certain columns in different proportions, imputing the missing values using KNN and other statistical methods, and using a random forest classification model to predict customer churn. The R2 score was used to compare predicted and actual values. The experiment also measured the time effectiveness of the automated data cleansing tool.

4 RESEARCH RESULTS

The initial project outputted a cleaned dataset which demonstrated unexpected results. The first table below Table 1 shows how the average R2 score of the imputed values varies with respect to the percentage of missing data per column.

Table 1: R2 score values by the proportion of the missing values per column

Missing Data % per Column	R2 Score
10%	**0.878**
30%	0.643
40%	0.524
60%	0.284

Table 2 compares different missing data handling techniques with respect to the missing data percentage per column based on the accuracy scores of the machine learning model trained by them. There were compared four most popular techniques namely, mean, KNN and Iterative Imputation as well as deletion.

Table 2: The accuracy of the model based on the missing value ratio and the imputation method

Missing Data % per Column	Mean Imputation	KNN Imputation	Iterative Imputation	Deletion
10%	0.860	**0.861**	0.860	0.855
30%	0.834	0.838	0.835	**0.870**
40%	0.841	0.840	0.842	**0.854**
60%	0.835	0.835	0.836	**0.858**

Table 3 compares the R2 scores of the different numerical data imputation techniques depending on the varying missing value ratios over the columns. There are mentioned six techniques which are used for the missing data imputation manually and automatically. Even though in auto mode Linear Regression is used, there are compared other models as well both to support the auto mode decision and to give flexibility to the users in terms of the choice for the selection of different methods.

Table 3: R2 score of numerical imputation techniques by the missing data ratio

Missing Data % per Column	Mean	Mode	Linear Regression	KNN	Iterative	Median
10%	0.90	0.75	**0.91**	0.90	0.90	0.89
30%	0.70	0.26	**0.73**	0.70	0.70	0.67
40%	0.70	0.26	**0.73**	0.70	0.70	0.67
60%	0.40	(0.52)	**0.46**	0.40	0.40	0.35

Table 4 compares the F1 scores of the different missing data imputation techniques by the proportion of the missing values per column. Due to the fact the missing values were in a categorical and textual format other types of missing data imputation techniques were not included in this experiment. Therefore, only three of the missing data imputation models are compared in the table, namely, Mode, KNN and Iterative.

Table 4: F1 score of the categorical imputation techniques by missing data ratio

Missing Data % per Column	Mode	KNN	Iterative
10%	**0.95**	0.93	0.93
30%	**0.85**	0.78	0.78
40%	**0.80**	0.70	0.70
60%	**0.70**	0.55	0.55

The next table, which is Table 5 evaluates the performance of the developed application by the noisiness of the imported dataset. The processing time is measured by using the timer() method of Python, which started running after the Apply button is pressed until the cleaned dataset is saved to the local computer of the user. It is measured in seconds and calculated as the difference between the start and end timestamps of the timer.

Table 5: Processing time of the application by the noisiness ratio of data

Missing Data % per Column	Duplicates Ratio, %	Processing Time, Seconds
10%	1.3	0.40
30%	0.4	1.60
40%	0.3	0.08
60%	0.1	0.08

The results in Table 6, which is deciding factor in the success of this project, is the comparison of the Auto and Manual modes in terms of the model performance that is trained by these datasets. The performance metric for this comparison was the accuracy of the random forest classifier. The dataset was first cleaned and preprocessed by only using the auto mode in the application without user interference. Then, the same noisy datasets were cleaned by using the data processing techniques that are chosen manually from the GUI. The obtained cleaned datasets were fed to the random forest classification model for training and testing.

Table 6: Accuracy of the model by the Auto and Manual mode

Missing Data % per Column	Auto Mode	Manual Mode
10%	**0.862**	0.856
30%	**0.878**	0.848
40%	**0.848**	0.842
60%	0.834	**0.835**

The program produces the log file listing the changes done to the input dataset with a timestamp as in shown Figure 11. There are listed types of logs which are INFO and DEBUG, their timestamp and

information that the log carries about the modifications to the dataset. The log files are saved in the same directory as the input file with .log extension which can be seen from the screenshot.

datacleaner.log ×

```
08-12-2022 00:31:52.06 - INFO - Started validation of input parameters...
08-12-2022 00:31:52.06 - INFO - Completed validation of input parameters
08-12-2022 00:31:52.07 - INFO - Started handling of duplicates... Method: "AUTO"
08-12-2022 00:31:52.15 - DEBUG - Deletion of 6000 duplicate(s) succeeded
08-12-2022 00:31:52.15 - INFO - Completed handling of duplicates in 0.075088 seconds
08-12-2022 00:31:52.15 - INFO - Started handling of missing values...
08-12-2022 00:31:52.16 - INFO - Found a total of 30000 missing value(s)
08-12-2022 00:31:52.19 - INFO - Started handling of NUMERICAL missing values... Method: "AUTO"
08-12-2022 00:31:58.77 - DEBUG - KNN imputation of 6000 value(s) succeeded for feature "credit_score"
08-12-2022 00:32:03.12 - DEBUG - KNN imputation of 6000 value(s) succeeded for feature "age"
08-12-2022 00:32:06.03 - DEBUG - KNN imputation of 6000 value(s) succeeded for feature "balance"
08-12-2022 00:32:08.92 - DEBUG - KNN imputation of 6000 value(s) succeeded for feature "active_member"
08-12-2022 00:32:11.78 - DEBUG - KNN imputation of 6000 value(s) succeeded for feature "estimated_salary"
08-12-2022 00:32:11.78 - INFO - Completed handling of missing values in 19.62981 seconds
08-12-2022 00:32:11.78 - INFO - Started handling of outliers... Method: "WINZ"
08-12-2022 00:32:15.79 - DEBUG - Outlier imputation of 3980 value(s) succeeded for feature "credit_score"
08-12-2022 00:32:19.62 - DEBUG - Outlier imputation of 3826 value(s) succeeded for feature "age"
08-12-2022 00:32:24.21 - DEBUG - Outlier imputation of 4000 value(s) succeeded for feature "balance"
08-12-2022 00:32:24.29 - DEBUG - Outlier imputation of 60 value(s) succeeded for feature "products_number"
08-12-2022 00:32:26.33 - DEBUG - Outlier imputation of 1975 value(s) succeeded for feature "active_member"
08-12-2022 00:32:31.09 - DEBUG - Outlier imputation of 4000 value(s) succeeded for feature "estimated_salary"
08-12-2022 00:32:33.14 - DEBUG - Outlier imputation of 2037 value(s) succeeded for feature "churn"
08-12-2022 00:32:33.14 - INFO - Completed handling of outliers in 21.363953 seconds
08-12-2022 00:32:33.14 - INFO - DataCleaner process completed in 41.131446 seconds
```

Figure 11: The initial log file generated by the automatic data cleansing program

Figure 12 shows an initial dashboard plotted to demonstrate the detected inconsistencies in the dataset such as the number of missing values, duplicates, and outliers.

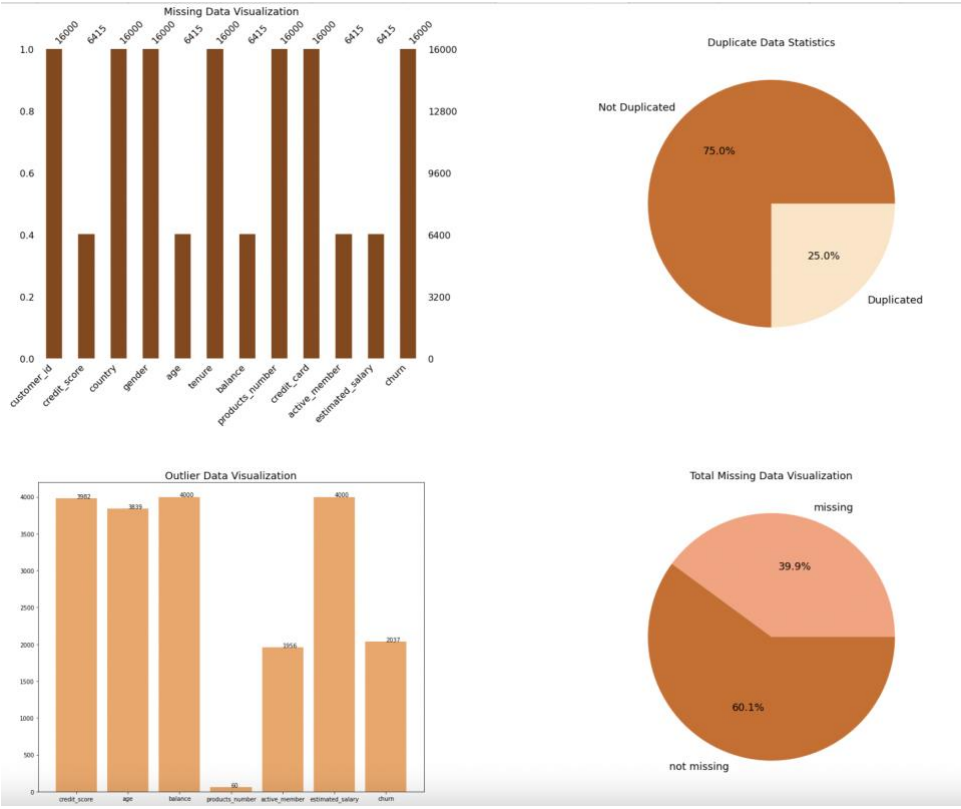


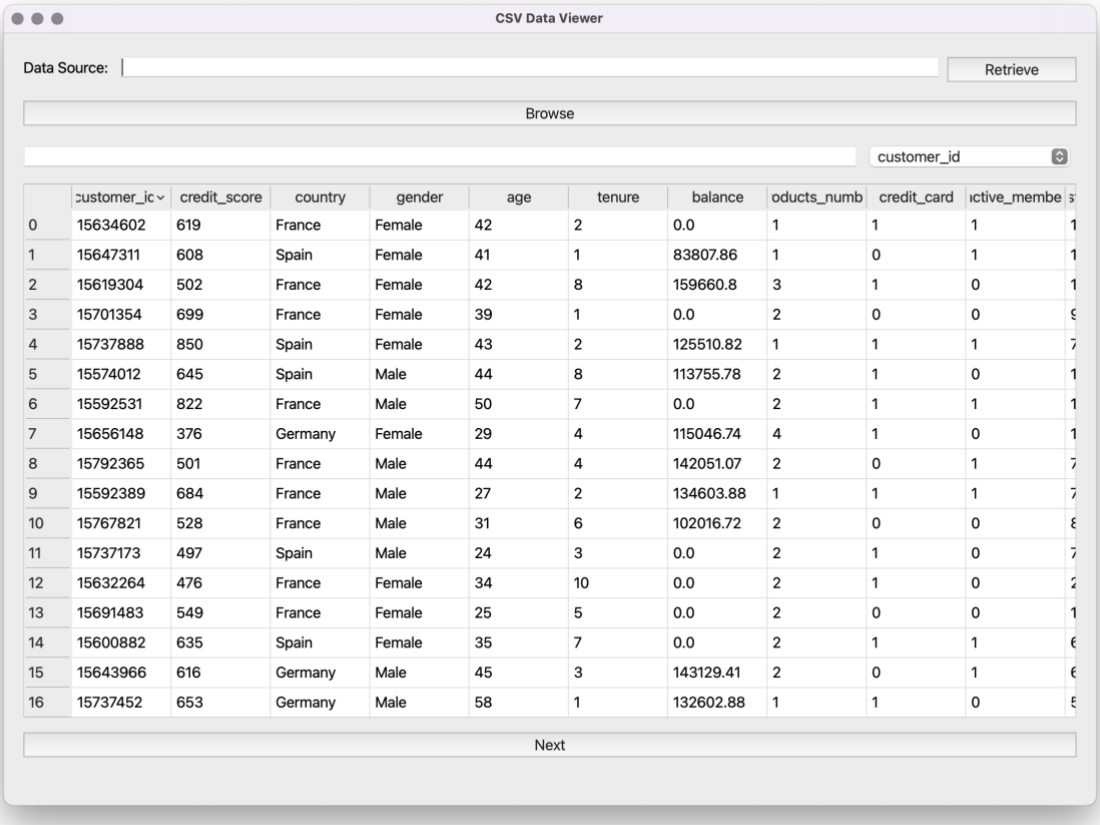
Figure 12: The initial design of subplot visualization of the dataset inconsistencies

4.1 Semi-automated Data Cleansing Application

The final project is the desktop application developed in Python and written in Jupyter Notebook. The application is created using Python's PyQt5 package and its widgets. Additionally, I used pandas package to read and perform initial observations on the dataset.

The Python code for a GUI application that was developed as part of a master's thesis project starts with importing necessary modules like sys, pandas, PyQt5, and QIcon. Then it defines a custom model class called "PandasModel" that inherits from the QAbstractTableModel class of PyQt5. This custom model class is used to display pandas data frames in the table view of the GUI application.

Figure 13 demonstrates the screenshot of the first page of the semi-automated data cleansing tool developed for this master's project. The users can browse the input dataset by either typing the path to the file into the Data Source field or by browsing them through the Browse button. In case the user would like to type the path to the dataset through the Data Source field he or she needs to press either Enter button on the keyboard or the Retrieve button in the right upper corner of the GUI page.



	customer_ic	credit_score	country	gender	age	tenure	balance	oducts_numb	credit_card	ıctive_membe	s
0	15634602	619	France	Female	42	2	0.0	1	1	1	1
1	15647311	608	Spain	Female	41	1	83807.86	1	0	1	1
2	15619304	502	France	Female	42	8	159660.8	3	1	0	1
3	15701354	699	France	Female	39	1	0.0	2	0	0	9
4	15737888	850	Spain	Female	43	2	125510.82	1	1	1	7
5	15574012	645	Spain	Male	44	8	113755.78	2	1	0	1
6	15592531	822	France	Male	50	7	0.0	2	1	1	1
7	15656148	376	Germany	Female	29	4	115046.74	4	1	0	1
8	15792365	501	France	Male	44	4	142051.07	2	0	1	7
9	15592389	684	France	Male	27	2	134603.88	1	1	1	7
10	15767821	528	France	Male	31	6	102016.72	2	0	0	8
11	15737173	497	Spain	Male	24	3	0.0	2	1	0	7
12	15632264	476	France	Female	34	10	0.0	2	1	0	2
13	15691483	549	France	Female	25	5	0.0	2	0	0	1
14	15600882	635	Spain	Female	35	7	0.0	2	1	1	6
15	15643966	616	Germany	Male	45	3	143129.41	2	0	1	6
16	15737452	653	Germany	Male	58	1	132602.88	1	1	0	5

Figure 13: The screenshot of the first page of the GUI for the semi-automated data cleansing tool

The next step after retrieval of the original dataset the user can see their dataset on the sheet created in the middle of the CSV Data Viewer screen. This screen has additional functionalities such as search, and sort based on the selected column selected on the middle right side of the screen. To perform a search on a specific column of the dataset, the users first need to choose it from the drop menu in the middle right side of the screen and then press Enter on their keyboard. The users can also sort the values according to the desired column by pressing the column name in the sheet demonstrated in the middle of the screen. The sheet page of the CSV Data Viewer screen has also both vertical and horizontal scroll features. In contrast to the Python data frame printing feature which trims and hides the middle parts of the column features and records, this screen gives the user full capability on browsing through the dataset and scrolling the feature names. This is an important step for the Data Scientist to understand

the logical meaning behind the dataset features as well as the timely see any noisy data that are unexpected such as text inside of the numerical features or symbols that appear in the dataset due to human error and need to be removed in a custom format. These kinds of inconsistencies, unfortunately, cannot be cleaned and removed by using predefined machine-learning techniques and one size fits all pipelines. Therefore, when they are detected should be deleted or replaced with custom values defined by the user and which are usually specific to the real-life scenario.

Overall, the first-page code defines a custom model class for displaying pandas data frames and creates a GUI application for data cleaning that allows users to search and sort different data based on the selected column options of a given data frame. This allows users not only to input the dataset in a fast and easy manner but also visually observe their dataset. Moreover, searching and sorting through the dataset will help the users understand their dataset and give insights about which data cleansing procedures and techniques may be better to use instead of others. After the pre-processing and the observation of the dataset is complete the users should press the Next button at the bottom of the GUI to go to the next page of the semi-automated data cleansing tool. This will open a new screen with the visuals such as charts and plots about the detected inconsistencies in the dataset shown in Figure 14. It consists of a subplot divided into four distinctive parts and a button placed in the bottom middle part of the window.

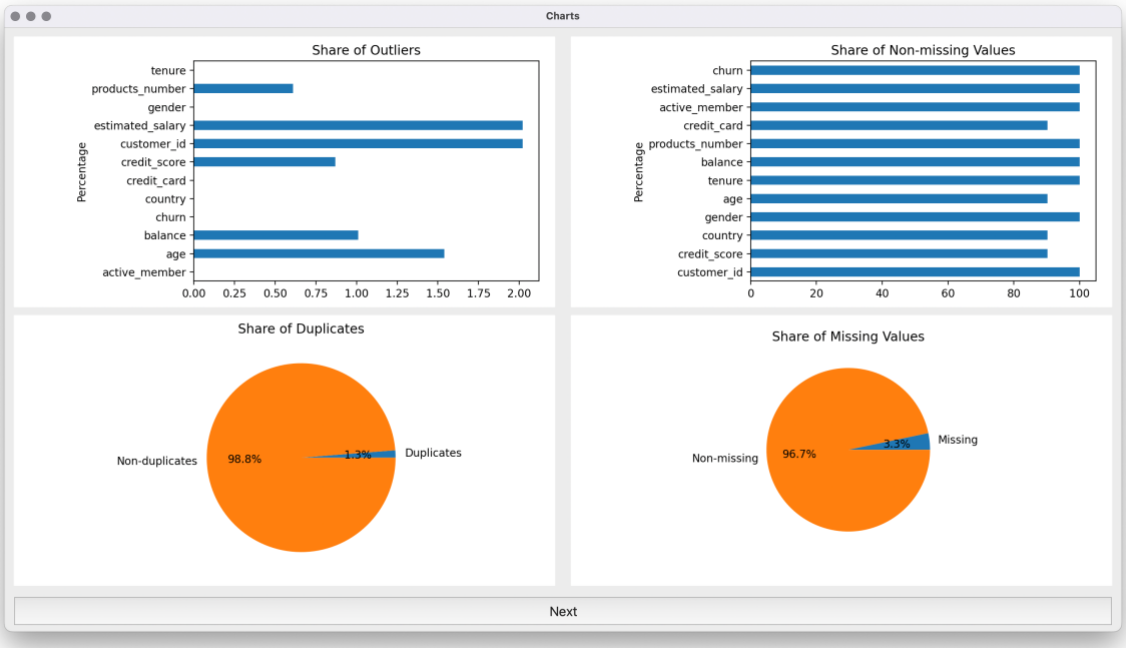


Figure 14: Dashboard about the dataset noisiness

The final screen shown in Figure 15 is the data processing form which is the most important part of the semi-automated data cleansing tool. It consists of many buttons, drop-downs, texts, and selections, which might frighten the user at the first sight. However, it has an extremely user-friendly scrollable layout to allow the users to scroll through each of the attributes in the dataset and apply specific data cleansing techniques selected from the drop-down menus. Otherwise, the users can directly press Apply button at the bottom of the screen and fully rely on the semi-automated data cleansing tool. In that case, the program defines all the modes, techniques, and methodologies for data cleansing. After which the log file and the cleaned dataset are saved as output.

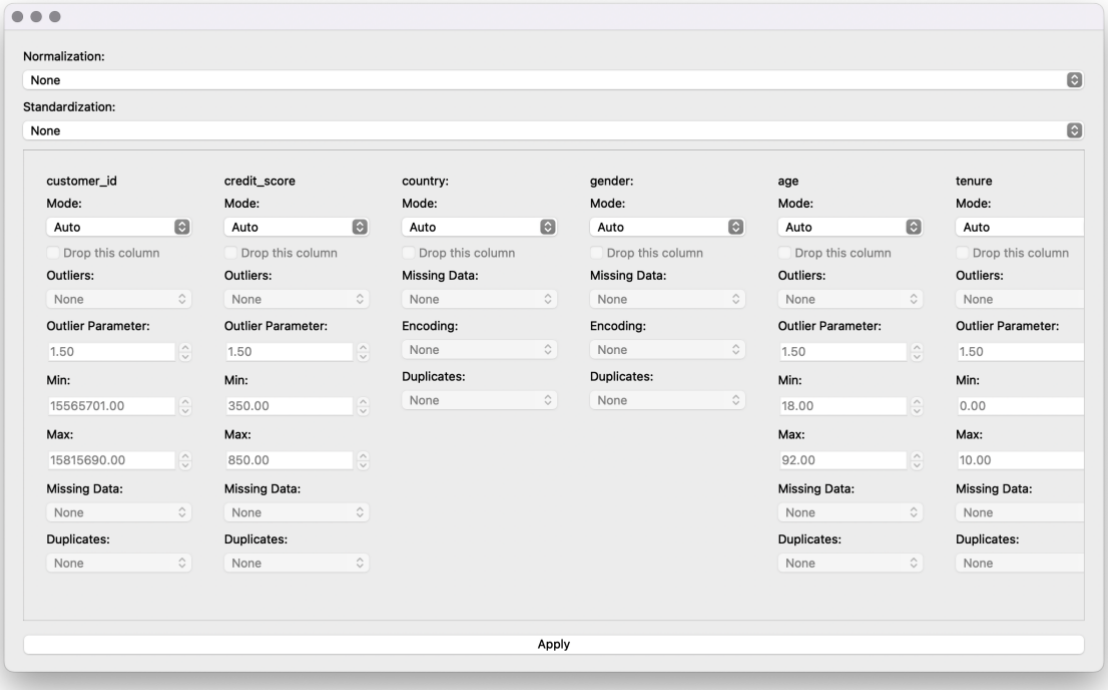


Figure 15: Data cleaner window of the semi-automated data cleansing tool

This is also the largest module of all the other screens due to its extremely large decision and rule-based algorithm. It consists of several classes such as MissingCatValues(), MissinNumValues(), Adjust(), Duplicates(), EncodeCateg(), Outliers(), Normalization(), Standardization(), ColumnOptions() and DataCleaningWindow().

In Figure 16, the screenshot from the output screen is demonstrated, which appears after the Apply button is pressed in the data cleaner window, discussed above. It consists of three parts of the text, the first one of which is the text letting the user know that the processing is done and the amount of time it took for the completion. The second part of the text in the dialogue window is the location paths, where the output files are saved, namely the cleaned dataset and the log file. This will help the user further navigate to the location easily and faster. The last part is the content of the log file showing the user every step of processing that was implemented in the original dataset. Each line of the log file starts with the date of the modification, the time, the type of information the log contains and the information itself. The information is repeated over each of the columns in the dataset and the processing technique such as outlier handling, and missing data imputation. Additionally, the output repeats the noisiness of the original data in terms of the number of missing values in a column and highlights the number of the imputed ones as well as the method with which the data was imputed. The window is scrollable, which allows the user to navigate easily through the list of logs of the application.

The dialogue window contains the OK button in the middle bottom of the screen, which closes the processing window. After the OK button is pressed the user can import another dataset for pre-processing and cleaning.

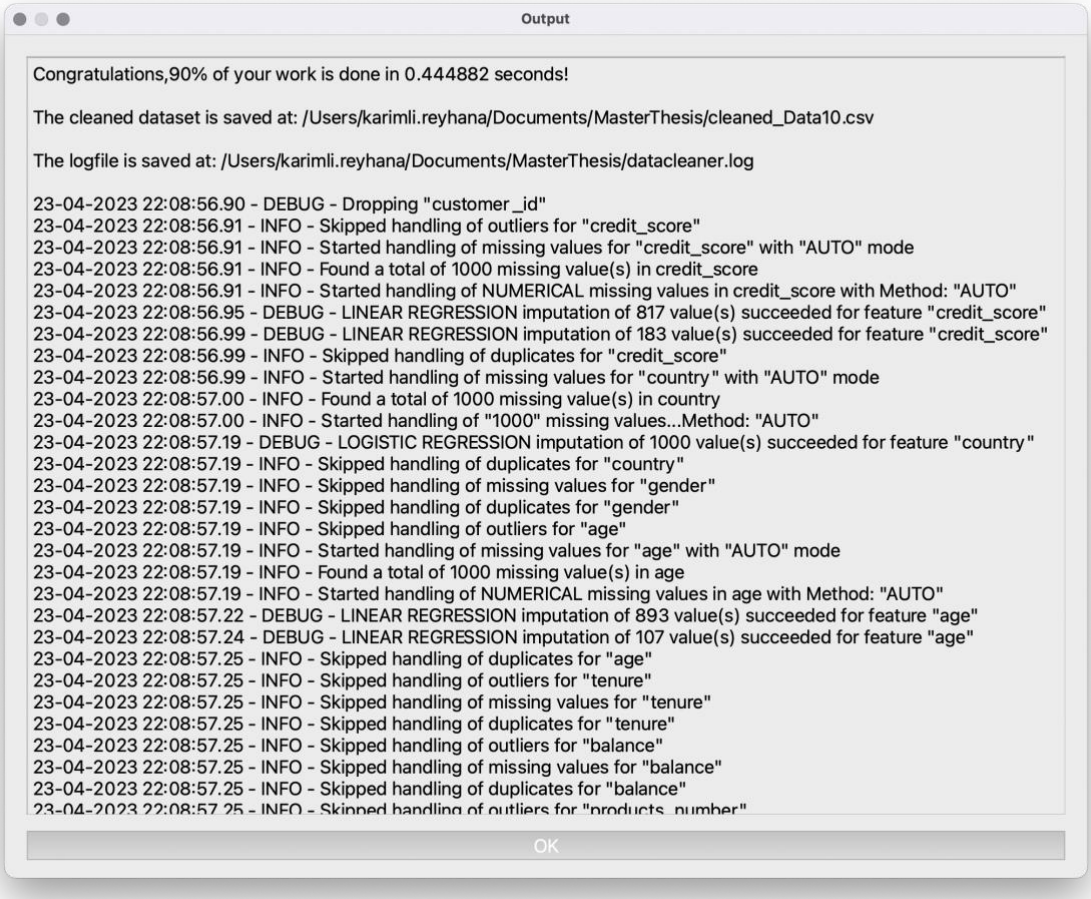


Figure 16: Output dialog window

4.1.1 Missing Categorical Values

The first class is a Python class called MissingCatValues that handles missing categorical values in a pandas dataframe. The class uses the numpy and scikit-learn libraries to perform imputation of missing categorical data.

The MissingCatValues class is initialized with the arguments input_data, column_name, and missing_mode_cat. The input_data parameter is a pandas dataframe that contains the data to process. column_name specifies the column or function in the DataFrame to process and missing_mode_cat specifies the method to use to handle missing categorical data.

The Handle method of the class is the main function that does the actual processing of the data. This method takes the parameters of the initialized class and performs the imputation of missing categorical data based on the specified method. The class method uses conditional statements to check whether the value of the missing_mode_cat argument is None or not. If it is None, the method skips processing of missing values. If not, it handles missing values by checking the method specified in missing_mode_cat.

The method of handling missing categorical values depends on the value of missing_mode_cat. If the value is Auto, the method uses logistic regression imputation and ANN imputation to impute missing categorical data. If the value is logistic regression, the method uses logistic regression to conditionally fill in the missing categorical data. If the value is KNN, the method uses KNN imputation to impute missing categorical data. If the value is the Mode, the method uses the most common column value to impute missing categorical data. If the value is Iterative, the method uses iterative imputation

to impute missing categorical data. If the value is None, the method simply returns a DataFrame with no missing values.

The _impute method is a private method used by the Handle method to impute missing values. This method accepts DataFrame, df, imputer, imputer, and type, type as arguments. If the type is numeric, the method performs a numeric imputation using the specified deductor. If the type is categorical, the method performs a categorical imputation with the specified imputer.

The class also contains a mapping dictionary that is used to map imputed values back to their original values. Additionally, the class uses a logging library to log various messages at various stages of the imputation process.

4.1.2 Missing Numerical Values

The MissingNumValues class represents a data preprocessing step to handle missing values in each dataset. The class constructor takes three parameters: input_data, a dataset containing the missing values; columnname, the name of the function containing the missing values; and missing_mode_num, a method for handling missing values in numeric data.

The Handle method is the main method of the class and takes no arguments other than the object instance itself. The method handles missing values in the record and returns a cleaned record. The method first checks whether the away mode is set to "None". In this case, it logs a message and returns the original dataset. Otherwise, it counts the number of missing values in the given function and removes rows with missing values using the dropna method. If the missing_mode_num parameter is set to a numeric handling method, the method calls the appropriate private _impute method to handle missing values in the function. Otherwise, it logs a message and returns the original dataset.

The private method _impute takes three arguments: df, the dataset containing missing values; imputer, the imputation method to use for handling missing values; and type, the data type of the feature containing missing values. The method first checks the data type of the feature and applies the corresponding imputation method to handle the missing values. It then logs a message indicating the number of missing values that were successfully imputed and returns the imputed dataset.

Compared to the previous version of the MissingNumValues class, this version adds support for additional imputation methods for handling missing numeric values, including mean, median, mode, iterative, and KNN imputation. In addition, the class now treats integer and floating-point values differently and matches categorical values before imputing missing categorical values.

The Duplicates class, as the name suggests, is designed to handle repetitive data. The class has an initialization method "init" that accepts three arguments: input data, column name, and duplicates. These arguments are assigned to the appropriate instance variables: data, functions, and duplicates.

The class also has a method called "handle" that does the actual handling of duplicates. The method first checks if there are any duplicates to be processed based on the duplicate argument. If the argument is None or Auto, the processing is skipped, and the original data is returned. If duplicate handling is required, the method removes duplicate rows based on the column specified in "feature" and resets the index. This method also counts the number of duplicate rows removed and logs the information using a logger. A warning message is written to the log if errors occur while processing duplicates.

4.1.3 Categorical Values

The following EncodeCateg class is a Python class used to encode the categorical features in each record. The class takes three parameters: input_data, column_name and encode_categ.

The input_data parameter refers to the dataset to be encoded, the column_name parameter refers to the name of the categorical column to be encoded, and the encode_categ parameter specifies the type of encoding to be performed on the categorical column should.

The constructor method (__init__) initializes the three parameters as class instance variables. The handle method is used to handle the encoding of a categorical feature. It takes a Pandas DataFrame as input and returns an encoded DataFrame.

If encode_categ is not set to No, the descriptor method continues encoding the categorical feature. If the first element of encode_categ is set to "Auto", the method performs an automatic encoding, where the method checks the number of unique values in the categorical column and performs a one-time encoding if the number of unique values is less than or equal to 10. If the number of unique values is less than or equal to 20, the method performs label encoding. Otherwise, the method skips encoding.

If "Label" is set, the method performs the label encoding. The _to_onehot method is a private class method that performs one-time encoding on a given categorical column. The _to_label method is also a private method that performs label encoding. In the _to_onehot method, the categorical function is one hot encoded using the get_dummies method from the pandas library. The method also checks the number of new features created by the encoding and logs a warning if the number of new features exceeds the specified limit. In the _to_label method, the categorical feature is first encoded using the LabelEncoder class from the sklearn.preprocessing module. The method then creates a mapping between the original values and the encoded values. The method also checks if any of the original values are NaN and replaces them with the appropriate encoded values.

Overall, the EncodeCateg class provides a convenient and efficient way to encode categorical features in each dataset.

4.1.4 Outliers

The Outliers class is designed to handle outliers in each dataset based on the specified mode of outlier handling. The class takes several input parameters including input_data, which represents the dataset, column_name, which is the name of the column to be analyzed for outliers, outliers_mode, which specifies the method to be used for outlier handling, and outlier_param, which is an optional parameter that specifies the threshold value for outlier detection.

The __init__ method initializes the class variables and sets the input parameters as class attributes. It also sets the default value for outlier_param as 1.5 if it is not provided. The processing method is responsible for processing outliers based on the selected processing mode. If the outlier setting is set to None or Automatic, it means that no outlier processing is required, and the original data is returned. When a particular outlier handling method is selected, it invokes the appropriate outlier handling method. Supported outlier processing methods are Winsorize, Drop, and Trim.

The _trim method truncates data by removing values that fall outside the range specified by the trim_min and trim_max parameters. It calculates the number of outliers removed and returns a truncated data set.

The _detect_outliers method calculates the upper and lower data bounds for the specified feature column based on the specified outlier_param. It then iterates over each value in the feature column and checks if it is out of bounds. If a value is found to be an outlier, it is replaced with an appropriate limit. The number of outliers that are replaced is counted and the function returns a record of outliers that have been replaced.

The _winsorization method replaces outliers with the corresponding bound using the Winsorization technique. It calculates the bounds based on the specified outlier_param and replaces each outlier with its corresponding bound. The function then returns the dataset with replaced outliers.

The _delete method deletes cases containing outliers based on the computed bounds for the specified feature column. It iterates over each value in the feature column and checks if it's out of bounds. If an observation is determined to be an outlier, it is removed. The number of outliers removed is counted and the function returns a record of outliers removed.

The _compute_bounds method calculates the upper and lower bounds for the specified feature column based on the specified outlier_param. It calculates the data's first and third quartiles and the interquartile range. The bounds are then calculated as the first quartile minus outlier_param multiplied by the IQR for the lower bound and the third quartile plus outlier_param multiplied by the IQR for the upper bound. The method returns the calculated bounds.

In general, the Outliers class provides a flexible and customizable approach to dealing with outliers in a dataset. The user can select the outlier processing method and specify the appropriate parameters to achieve the desired result.

4.1.5 Normalization

The first step in the Normalization class is to define the init() method, which is the constructor method that initializes the object when it is created. The constructor method takes two arguments: input_data and normalization_mode. The input_data parameter is the dataset that will be normalized, while the normalization_mode parameter determines the method that will be used to normalize the data.

Next, the handle() method is defined. This method takes the input_data as its parameter and returns the normalized dataset. If normalization_mode is set to "None", the method simply returns input_data without normalization. However, if normalization_mode is set to Auto, MaxAbs Scaler, or MinMax Scaler, the procedure continues with the normalization process.

The first step in the normalization process is to define the columns in input_data that contain numeric data. This is done using the pandas select_dtypes() method, which selects columns based on their data type. In this case, we select columns with a numeric data type and exclude any columns with a "timedelta" or "datetime" data type.

Once the numeric columns are defined, the normalization process begins. If normalization_mode is set to MaxAbs Scaler or Auto and the minimum record value is less than 0, the method jumps to the MaxAbsScaler normalization method. This method scales the data between -1 and 1 by dividing each value by the largest absolute value in the data set. The MaxAbsScaler method is implemented using the MaxAbsScaler class from the sklearn.preprocessing library.

If normalization_mode is set to MinMax Scaler or Auto and the minimum record value is greater than or equal to 0, the method jumps to the MinMaxScaler normalization method. This method scales the data between 0 and 1 by subtracting the minimum value from each value in the dataset and then dividing it by the range of the dataset (that is, the difference between the maximum and minimum values). The MinMaxScaler method is implemented using the MinMaxScaler class from the sklearn.preprocessing library.

Finally, the method returns a normalized dataset. If the normalization was successful, a message is written to the log indicating which method was used to normalize and that the normalization was successful. Otherwise, an error message is logged indicating that normalization failed.

So the normalization class takes a data set and a normalization method as input and returns a normalized data set. It first identifies the columns in the dataset that contain numeric data, and then normalizes them using the MaxAbsScaler or MinMaxScaler method, depending on the specified normalization method. The result record is returned along with a log message indicating the normalization method used and the success of the normalization.

4.1.6 Standardization

The Standardization class is used to perform the standardization of input data. Standardization is the process of scaling numerical data to produce a zero mean and unit variance. The class takes two inputs: input_data, which is the data to be standardized, and standardization_mode, which is a string specifying the method to use for standardization.

In the class's __init__ method, these two inputs are stored as class attributes, self.standardization_mode and self.data. The descriptor method is then used to perform the actual standardization. It first creates a copy of the input and assigns it to the variable df.

Next, the method checks if the standardization_mode is set to 'None'. If it is, the method logs a message indicating that standardization has been skipped and returns the original input data. If standardization_mode is not set to 'None', then the method proceeds to standardize the data using the specified method.

To determine which method to use, the method first selects all columns in the input data that are of numeric data types. It excludes any columns that are of the timedelta or DateTime data types. The select_dtypes method of a pandas DataFrame is used to accomplish this.

If standardization_mode is set to 'Auto' or 'Robust Scaler', the method logs a message indicating that data standardization with the robust scaler method has started. The RobustScaler class from scikit-learn is then used to perform the standardization on the numeric columns of the input data. The transformed data is then stored back in the input data object, df. The method logs a message indicating that standardization with the robust scaler method has succeeded.

If standardization_mode is set to 'Standard Scaler', the method logs a message indicating that data standardization with the standard scaler method has started. The StandardScaler class from scikit-learn is then used to perform the standardization on the numeric columns of the input data. The transformed data is then stored back in the input data object, df. The method logs a message indicating that standardization with the standard scaler method has succeeded.

Finally, the method returns the standardized input data object. Overall, this class provides a convenient way to perform data standardization on input data using different standardization methods.

4.1.7 GUI Logic

The next code part that needs to be explained is binding between the user input and the program logic. It consists of a series of functions aimed at performing automated data cleaning on a given dataset. The functions are contained in a script file and are imported into other Python programs using the 'import' statement.

The DataCleaner function is the main function that performs automated data cleaning. It takes four arguments: 'data' which is the input dataset, 'column_name' which is the name of the column to be cleaned, 'outlier' which is a list that contains the outlier cleaning method and its associated parameters, 'missing' which is a string that indicates the missing value cleaning method to be used, and 'duplicate' which is a string that indicates the duplicate cleaning method to be used. The function calls the 'OutlierHelper', 'MissingHelper', and 'DuplicateHelper' functions to perform the cleaning.

The 'DuplicateHelper' function takes three arguments: 'data' which is the input dataset, 'column_name' which is the name of the column to be cleaned, and 'duplicate' which is a string that indicates the duplicate cleaning method to be used. It checks the value of 'duplicate' and calls the appropriate function to perform the cleaning. The 'Duplicates' function is called to handle the duplicate cleaning and returns the cleaned dataset.

The 'MissingHelper' function takes three arguments: 'data' which is the input dataset, 'column_name' which is the name of the column to be cleaned, and 'missing' which is a string that indicates the missing value cleaning method to be used. It checks if the column is numeric or categorical and calls the appropriate function to perform the cleaning. The 'MissingNumValues' function is called to handle the missing value cleaning for numeric columns while the 'MissingCatValues' function is called to handle the cleaning for categorical columns.

The 'OutlierHelper' function takes three arguments: 'data' which is the input dataset, 'column_name' which is the name of the column to be cleaned, and 'outlier' which is a list that contains the outlier cleaning method and its associated parameters. It checks if the column is numeric and calls the appropriate function to perform the cleaning. The 'Outliers' function is called to handle the outlier cleaning and returns the cleaned dataset.

4.1.8 GUI Design

The next important class that creates the data cleaning window and the button logic behind it is class called ColumnOptions that subclasses the QWidget class from the PyQt5.QtWidgets module. The initializer method (init) takes two arguments: column_name and data.

The first thing the initializer does is call the superclass's initializer method (super().init()) to set up the QWidget instance. It then calls a method called _initialize_logger() to set up logging, passing in True for the verbose and logfile parameters.

Next, it sets the instance variables column_name and data to the values passed in as arguments. It then gets a list of all numeric columns in the data frame excluding datetime and timedelta columns and a list of all data columns in the data frame.

If the column specified by column_name is a numeric column, it creates several Qt widgets that allow the user to specify how to handle outliers, missing data, and duplicates. Specifically, it creates a QComboBox widget to select the outlier handling method, a QDoubleSpinBox widget to specify the outlier parameter, and two QDoubleSpinBox widgets to specify the minimum and maximum outlier values. It also creates a QComboBox widget to select the missing data handling method, a QComboBox widget to select the duplicate handling method, a QComboBox widget to select the encoding method, and a QCheckBox widget to specify whether to drop the column. All these widgets are disabled by default.

If the column specified in column_name is a date column, a similar set of Qt widgets is created to handle missing dates and duplicates. However, it does not provide options for handling outliers or encoding the column.

If the column specified in ColumnName is neither a numeric nor a date column, the same Qt widget set is created as for the date column, but a QComboBox widget is added to select the encoding method.

Finally, the initializer instantiates the QVBoxLayout and adds all generated widgets to it in the appropriate order. This QVBoxLayout is set as the layout for the QWidget instance.

This next class defines a graphical user interface window for deleting data. The class is called DataCleaningWindow and derives from the QtWidgets.QWidget class, which is part of the PyQt5 library used to develop GUIs in Python. The DataCleaningWindow class takes three arguments in its constructor: num_columns, column_names, and data.

The num_columns argument specifies the number of columns in the data set that the user is deleting. The column_names argument is a list of strings containing the names of the columns in the dataset. The data argument is pandas dataframe that contains the data to delete.

The constructor initializes some instance variables that are used throughout the class. The num_columns argument is stored as the instance variable self.num_columns. The data argument is stored as an instance variable of self.original and self.data. These two variables are the same at the start of the cleanup process, but self.data is updated when the user clears the data.

The constructor then uses the QtWidgets.QScrollArea class to create a scrollable area that allows the user to scroll through the deleted columns. A scroll area is created by instantiating a QtWidgets.QScrollArea object and setting it as the parent widget. A QtWidgets.QWidget object is created to serve as the child widget of the QScrollArea. A QtWidgets.QHBoxLayout object is created to hold the widgets that represent the columns. The QtGui.Qt.AlignTop flag is set to ensure widgets align to the top of the layout.

The constructor then creates an empty list named self.column_options. This list contains instances of a custom class called ColumnOptions that represents the options the user can choose for each column in the dataset. A loop is used to iterate through each column in the dataset. A new instance of the ColumnOptions class is created for each column and added to the self.column_options list. An instance representing the scrollable area is also added to the QtWidgets.QHBoxLayout object.

The constructor then creates two combo boxes using the QtWidgets.QComboBox class. These combo boxes allow the user to select normalization and standardization options for the dataset. The Normalization combo box contains the options None, Auto, MinMax Scaling, and MaxAbs Scaling. The Standardization combo box is populated with the options None, Auto, Standard Scaler, and Reliable Scaler.

The constructor then creates a QtWidgets.QVBoxLayout object that contains all the widgets in the GUI window. The layout is populated with a normalization combo box label, the normalization combo box itself, a standardization combo box label, the standardization combo box itself, a scrollable area, and an Apply button.

Finally, the handle_apply method is defined. This method is called when the user clicks the Apply button. It is responsible for processing the user-selected options for each column in the dataset, cleaning

the data according to those options, and then applying the user-selected normalization and standardization options.

The method first identifies the numeric columns in the dataset using pandas' select_dtypes method. It then iterates over each instance of the ColumnOptions class that was previously created and prints the options that the user selected for each column. If the user has selected the Delete Column option, Pandas' delete method will remove the column from the dataset.

Next is the Adjust class, which is used to preprocess a data set before applying other data cleaning routines. The class has an init method that takes an input_data parameter and initializes the class's data attribute with the input data.

4.1.9 Automatic Adjustment of Data

The Handle method of the Adjust class is the main method that processes the record. It first removes all duplicate rows in the dataset using the pandas drop_duplicates() method. It then calls the convert_datetime method, which converts all string-encoded objects in the dataset to a DateTime type. Finally, it returns the processed record.

The convert_datetime method is responsible for extracting date and time values from the data. It first checks if the extract_datetime attribute is set to true. If so, the method uses the changeset operation to determine which columns of the record are non-numeric. For each of these columns, it attempts to convert the column to datetime format using the pandas to_datetime() method. If successful, the function logs a debug message indicating that the conversion was successful; Otherwise, it silently fails. After processing all non-numeric columns, the method returns the changed record.

Overall, the Adjust class provides a useful first step in the data cleansing process. It discards duplicate strings and converts all string-encoded functions to the DateTime data type. By performing these pre-processing steps, the Adjust class ensures that the dataset is in a clean and consistent format before applying additional data cleansing procedures.

4.2 Analysis of Study

During the experiments, different techniques and performance metrics were implemented to evaluate objectively the success of the project. The first experiment was the average accuracy of the missing data imputation techniques depending on the missing data ratio in each column. Table 1 shows that the R2 score objective over 0.7 is achieved when the missing data proportion per column is less than 30%. The accuracy of the imputed values missing values is decreasing with the increase of the proportion of the missing values. This is a good indicator that it is not recommended to impute the missing values when a large proportion of them is missing. The imputation would lead to inaccurate data and wrong decisions and erroneous predictions.

Additionally, deletion methods produce better accuracy results in the modelling than imputation in cases when the proportion of missing values per column is higher. The data in Table 2 represents the percentage of missing data per column and the performance of different imputation techniques, namely Mean Imputation, KNN Imputation, Iterative Imputation, and Deletion. The performance of these techniques is measured based on the accuracy of the machine learning model achieved after imputing missing values in the dataset. The results indicate that as the percentage of missing data increases, the accuracy of Mean Imputation decreases, while the accuracy of KNN Imputation and Iterative Imputation remains relatively stable. The Deletion technique also shows a decrease in accuracy as the percentage of missing data increases. In the case of 10% missing data, all imputation techniques achieve similar accuracies, with Mean Imputation and KNN Imputation achieving slightly higher accuracies compared to Iterative Imputation and Deletion. However, as the percentage of missing data increases to 30%, the performance of Mean Imputation and KNN Imputation decreases, while the accuracy of Iterative Imputation remains stable. The Deletion technique shows the highest accuracy in this scenario, which may be due to the removal of highly corrupted columns from the dataset. At 40% and 60% missing data, the accuracy achieved by all imputation techniques decreases further, with Iterative Imputation performing the best among the imputation techniques. The Deletion technique still shows a

higher accuracy compared to Mean Imputation and KNN Imputation at 40% missing data, but its accuracy decreases significantly at 60% missing data. Finally, the results suggest that Mean Imputation and KNN Imputation may not be suitable for datasets with a high percentage of missing data. Instead, Iterative Imputation can provide more reliable results in such cases. The Deletion technique may show a higher accuracy in some scenarios, but it may lead to a significant reduction in the sample size and may not be ideal for all datasets.

After observing the variations in the performance of the machine learning model depending on the missing value imputation technique, I evaluated separately the performance of each of the missing data imputation methods. Table 3 provides information on the performance of different methods for handling missing data, including Mean imputation, Mode imputation, Linear Regression imputation, KNN imputation, Iterative imputation, and Median imputation. The evaluation is also based on the percentage of missing data per column, ranging from 10% to 60%. The obtained results suggest that as the percentage of missing data per column increases, the accuracy of the imputation methods decreases. At the lowest level of missing data, which is 10%, all the methods perform reasonably well, with the lowest accuracy rate of 0.89 and the highest of 0.91, which is in Linear Regression Imputation. However, as the missing data percentage increases to 30%, the accuracy rates decrease to between 0.67 and 0.73, indicating that imputation methods are less effective in handling more missing data. At 40% missing data, the accuracy rates remain low, with all methods performing similarly, with rates ranging from 0.67 to 0.73. At the highest level of missing data, which is 60%, the accuracy rates drop significantly, particularly for mean, mode, and iterative imputation methods. Even though this suggests that these methods are less effective in handling a high percentage of missing data, Linear Regression Imputation still performs better than other models.

While Linear Regression Imputation performs better in all levels of missing data ratio, according to the produced results from the experiments, I modified the algorithm of the automatic missing value handling such that after a threshold of 30% the missing values would be dropped instead of the imputation as this not only produces low accuracy results but also leads to the erroneous predictions.

Table 4 provides information on the performance of different imputation methods for handling missing categorical data, including Mode imputation, KNN imputation, and Iterative imputation. The evaluation is based on the percentage of missing data per column, ranging from 10% to 60%. In this experiment as well, the results suggest that as the percentage of missing data per column increases, the accuracy of the imputation methods decreases. At the lowest level of missing data (10%), all the methods perform reasonably well, with accuracy rates ranging from 0.93 to 0.95, with the highest at Mode imputation. However, as the missing data percentage increases to 30%, the accuracy rates decrease to between 0.78 and 0.85, indicating that imputation methods are less effective in handling more missing data. At 40% missing data, the accuracy rates remain low, with all methods performing similarly, with rates between 0.70 and 0.80. At the highest level of missing data (60%), the accuracy rates drop significantly, particularly for KNN and Iterative imputation methods. This suggests that these methods are less effective in handling a high percentage of missing categorical data. It is also worth noting that Mode imputation appears to be the most effective method for handling categorical missing data, with consistently higher accuracy rates than KNN and Iterative imputation methods. This can be due to the not proportional distribution of the categorical values over the dataset and with one of the values dominating over the others. Therefore, the mode of the variable produces the highest F1 score.

However, the choice of imputation technique should depend on the specific characteristics of the dataset and the research question at hand. Therefore, it is better to drop the missing values with an increase in the proportion of the missing values above a certain threshold.

Next, I experimented with the processing time of the application by the noisiness of the imported dataset. Table 5 provided information about the results of this experiment, which suggest that the processing time of the program is the lowest ranging from 0.08 to 1.6 seconds when the missing ratio is the highest. Additionally, the processing time of the application also increases subtly when the duplicate ratio increases in the dataset. It is also worth mentioning that the processing time is the highest when the missing data ratio is 30%, whereas it is the lowest when the missing data ratio is the highest

which is 60%. This is due to the reason that with an increase in the proportion of the missing values the program drops them instead of imputing them, which is faster in terms of processing time.

The last experiment I conducted was about measuring the accuracy of the model depending on the cleaning mode of the program, namely Auto and Manual. The results in Table 6 demonstrate that the Auto mode mostly performs better than the Manual mode across different ratios of the missing values ranging from 10% to 60%. Even though in low ratios of the missing values the difference between the performances of the mode is subtle, with the increase in the missing data ratio from 10% to 30% it increases significantly, with the accuracy for the Auto mode reaching 0.878 whereas for the Manual mode dropping to 0.848. Although with a further increase in the missing data proportion, the difference is reduced, the Auto mode still produces better results than the Manual mode, with 0.848 for the Auto mode and 0.842 for the Manual mode. The situation slightly changes in favour of Manual mode when the missing data ratio increases to the maximum, which is 60%. In this case, Manual mode outperforms Auto mode by a small amount, with the accuracy for the Manual mode being 0.835 compared to 0.834 in Auto mode.

The obtained results suggest, the developed Auto mode logic considers the small details and captures the nature of the dataset and noise than the user would do in a manual mode. While the last experiment suggested that the performance difference is subtle between the two modes, considering the processing time and the easiness of the implementation it would be more beneficial for the user to apply the automatic data pre-processing and cleansing.

5 SUMMARY

This project is about developing a semi-automated data cleansing tool that is based on the research, experiments and reviewed literature that defines a functional pipeline on which data cleansing techniques to perform automatically on the noisy dataset with the minimum user prompt and achieve high accuracy on the performed machine learning model. The project was developed as a desktop application that can be installed and used everywhere without the need for installing any other dependencies, packages, and Python modules. The user-friendly desktop app allows the users to upload their datasets, observe the inconsistencies and perform the data cleansing techniques on their datasets by simply clicking and selecting the desired options. The users can track the modifications applied to the dataset by just reading the log file generated by the program in case they would like to change the performed techniques with other ones the next time. The cleaned dataset is produced separately and saved on the local computer of the user without changing the original one.

The research performed an analysis of the results with the datasets with varying percentages of noisiness and datasets to measure the reproducibility and the performance of the program. Moreover, I experimented with other performance metrics such as the accuracy of the imputed values, the amount of time it takes for the program to process the dataset as well as the accuracy of the model trained by using the cleaned dataset generated as output from the application.

The semi-automated data cleansing tool produced successful results in terms of the generated clean dataset, however, there are a wide variety of real-world scenarios which would need the custom solutions because the pre-defined pipeline and architecture might not consider and fit all the cases that the inconsistencies in the dataset were created either by human errors or system malfunctions.

Overall, the project is successful and meets the preset goals of the semi-automated data cleansing tool and can be further developed in terms of the increase of automation and reductions of human input. The final project partially automates data cleansing and pre-processing steps and reduces human labour and therefore, reduces the errors that are produced by a human due to the inability to choose the best algorithms or lack of knowledge about the modelling and the original dataset itself.

5.1 Future Work

The main aim of the project was to create and research the adequacy of the developed desktop application for the automatic preprocessing and cleansing of datasets. The application is semi-automated which is the main downside of it. It should be further researched and automated in terms of

efficient pre-processing and cleansing of the datasets before the modelling. Additionally, the application should be further developed to handle all other different real-life scenarios which might occur in the life of the data scientist. The automatic model should be able to capture in its decision process the wide variety of reasons and types of noisiness and inconsistencies occurring in the datasets.

Additionally, the project should be developed to cover not only the structured numeric and categorical datasets but also textual and image datasets, which are unstructured. This is because most of the data in real life is in unstructured form and all of it needs to be pre-processed and cleaned before modelling.

Finally, the application should be developed and deployed further to be experimented with by other data scientists for the implementation of their real-life tasks. This is currently impossible as the application is currently running on a local machine only as a part of the master thesis project, but it can be further developed and deployed for public use.

REFERENCES

[1] F. Ridzuan and W. M. Nazmee, "A review on data cleansing methods for big data," in *The Fifth Information Systems International Conference*, 2019.

[2] D. Mertz, Cleaning Data for Effective Data Science, Packt Publishing, 2021.

[3] H. J. H. a. Y. B. Xin Wang, An Ontology-Based Approach to Data Cleaning, Regina, Saskatchewan, 2005.

[4] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez and L. Kaiser, "Attention is All You Need," in *31st Conference on Neural Information Processing Systems*, Long Beach, 2017.

[5] Y. Li, A. Anastasopoulos and A. W. Black, "Towards Minimal Supervision BERT-Based Grammar Error Corection," in *The Thirty-Fourth AAAI Conference on Artificial Intelligence*, 2020.

[6] N. Parmar, A. Vaswani, J. Uszkoreit, L. Kaiser, N. Shazeer, A. Ku and D. Tran, "Image Transformer," in *Proceedings ofthe 35th Internartional Conference on Machine Learning*, Stockholm, 2018.

[7] J. Devlin, M.-W. Chang, K. Lee and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," 2019.

[8] J. E. Robles-Alcaraz, A review of techniques for handling missing data in data science, 2021.

[9] S. Zhang, H. Huang, J. Liu and H. Li, "Spelling Error Correction with SOft-Masked BERT," in *58th Annual Meeting of the Association for Computational Linguistics* , 2020.

[10] M. Tan, D. Chen, Z. Li and P. Wang, "Spelling Error Correction with BERT based on Character-Phonetic," in *IEEE Xplore*, 2020.

[11] R. Fakhitah and W. M. N. Wan Zainon, "A Review on Data Cleansing Methods For Big Data," in *The Fifth Information Systems International Conference*, 2019.

[12] Y. Bai, "Data cleansing method of talent management data in wireless sensor network based on data mining technology," *EURASIP Journal on Wireless Communications and Networking,* vol. 33, 2019.

[13] Z. Hu and D. Du, "A New Analutical Framework For Missing Data Imputation and Classification with Uncertainty: Missing Data Imputation and Heart Failure Readmission Prediction," *PLOS ONE,* vol. 15, no. 9, 2020.

[14] A. Petrozziello, I. Jordanov and C. Sommeregger, "Distributed Neural Networks for Missing Big Data Imputation," in *IEEE*, 2018.

[15] A. Folch-Fortuny, F. Arteaga and A. Ferrer, "Assessment of Maximum Likelihood PCA Missing Data Imputation," *Chemometrics,* vol. 10, no. 1002, pp. 386-393, 2016.

[16] Z. Y. Khayyat, "Scaling Big Data Cleansing," King Abdullah University of Science and Technology, Thuwal, 2017.

[17] G. Topre, "www.kaggle.com," 31 July 2022. [Online]. Available: https://www.kaggle.com/datasets/gauravtopre/bank-customer-churn-dataset. [Accessed 20 September 2022].