PRAGUE UNIVERSITY OF ECONOMICS AND BUSINESS

FACULTY OF ACCOUNTING AND FINANCE

Department of Banking and Insurance

Application of Machine Learning Algorithms within Credit Risk Modelling

Master's thesis

Author: Bc. Petr Nguyen

Study program: Banking and Insurance | Data Engineering

Supervisor: prof. PhDr. Petr Teplý, Ph.D.

Year of defense: 2023

Declaration of Authorship

I, as an author, hereby declare that I wrote and compiled the Master's thesis *"Application of Machine Learning Algorithms within Credit Risk Modelling"* independently, using only the resources and literature listed in bibliography.

May 25, 2023, Prague

Petr Nguyen

Abstract

The abstract should concisely summarize the contents of a thesis. Since potential readers should be able to make their decision on the personal relevance based on the abstract, the abstract should clearly tell the reader what information he can expect to find in the thesis. The most essential issue is the problem statement and the actual contribution of described work. The authors should always keep in mind that the abstract is the most frequently read part of a thesis. It should contain at least 70 and at most 120 words (200 when you are writing a thesis). Do not cite anyone in the abstract.

Keywords: machine learning, data science, credit risk, probability of default, loans, mortgages

Abstrakt

Nutnou součástí práce je anotace, která shrnuje význam práce a výsledky v ní dosažené. Anotace práce by neměla být delší než 200 slov a píše se v jazyce práce (tj. česky, slovensky či anglicky) a v překladu (tj. u anglicky psané práce česky či slovensky, u česky či slovensky psané práce anglicky). Anotace práce by neměla být delší než 200 slov a píše se v jazyce práce (tj. česky, slovensky či anglicky) a v překladu (tj. u anglicky psané práce česky či slovensky, u česky či slovensky psané práce anglicky). V abstraktu by se nemělo citovat.

Klíčová slova: machine learning, data science, kreditní riziko, pravděpodobnost defaultu, úvery, hypotéky

Acknowledgments

I, as an author, would like to express my deepest gratitudes and thanks to my supervisor prof. PhDr. Petr Teplý, Ph.D.for his help and significant advices throughout my thesis. Last but not least, I would like to also thank to my family for an enormous support during my studies.

Contents

List of Tables

List of Figures

Acronyms

ML Machine Learning

PD Probability of Default

AUC Area Under the Curve

LR Logistic Regression

RF Random Forest

GB Gradient Boosting

MLP Multi-Layer Perceptron

DT Decision Tree

ADASYN Adaptive Synthetic Sampling

Chapter 1

Introduction

TBD

This document serves two purposes. First, it is a template and example for a master's thesis. Second, the text in all sections contains some useful information on structuring and writing your thesis.

The introduction should consist of three parts (as paragraphs, not to be structured into multiple headings): The first part deals with the background of the work and describes the field of research. It should also elaborate on the general problem statement and the relevance. The second part should describe the focus of the thesis, typically the paragraph starts with a phrase like "The objective of this thesis is ...." The last part should describe the structure of the thesis, for instance in the following manner. The thesis is structured as follows: Chapter 2 cites some formal requirements of the faculty and the frequently asked questions about the template, Chapter 3 gives some hints on basic formatting features and covers also acronyms, figures, boxes and tables. Chapter 4 gives a recommendation on the usage of hyphens in English language in LaTeX and explains how to use the itemize and quote environments and shows a few enumerate-based environments. Chapter 5 presents a checklist of common mistakes to avoid. **??** contains numerous hints. Chapter 6 summarizes our findings.

Chapter 2

Credit Risk Modelling

TBD

Chapter 3

Machine Learning

TBD

3.1 Terminology

TBD

3.2 Algorithms

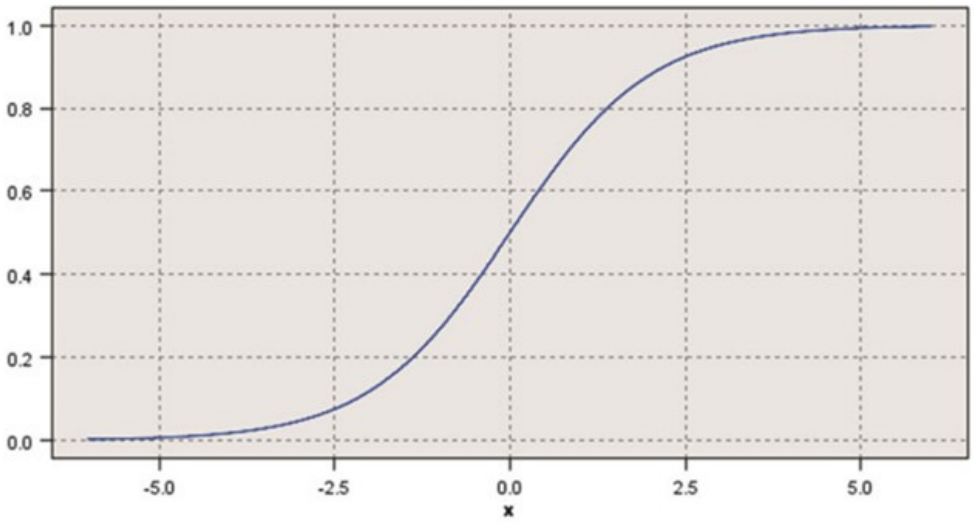
In this section, several algorithms, which are used in the machine learning implementation, are going to be described. Since the goal is to predict whether or not given client will default, henceforth only (binary) classification algorithms as a part of the supervised learning are described. In other words, regression models and unsupervised learning algorithms are out of the scope of this thesis.

3.2.1 Logistic Regression

TBD

Despite the algorithm's name, it is actually not a regression but rather a classification model. In contrast, a linear regression's target variable is continuous whereas regarding a logistic regression, the target variable is binary or dichotomous. For the probability estimation it is using a logistic, or so-called sigmoid function, which maps any real value within the range of 0 to 1 and takes a S-shaped curve as can be seen in Figure 3.1.

Figure 3.1: Logistic function



Source: (Wendler & Gröttrup 2021)

The linear form of the logistic regression with n features can be written as:

$$\ln\left(\frac{P}{1-P}\right) = \beta_0 + \sum_{i=1}^{n} \beta_i X_i \tag{3.1}$$

where P is the probability of the occurred event, conditional on the set of given features. Let us denote $Y = 1$ as an observed target instance where the event occurred (e.g., defaulted), then:

$$P = \Pr(Y = 1 \mid X_1, X_2, \ldots, X_n) \tag{3.2}$$

Therefore, the term within the natural logarithm are the odds or more particularly, the ratio of the probability of the event with respect to the probability of non-event, both conditional on the same set of given features.

$$\begin{aligned} \frac{P}{1-P} &= \frac{\Pr(Y = 1 \mid X_1, X_2, \ldots, X_n)}{1 - \Pr(Y = 1 \mid X_1, X_2, \ldots, X_n)} \\ &= \frac{\Pr(Y = 1 \mid X_1, X_2, \ldots, X_n)}{\Pr(Y = 0 \mid X_1, X_2, \ldots, X_n)} \end{aligned} \tag{3.3}$$

Referring to the previous equations, solving for P, henceforth we get a final equation for computing the probability of occurred event with usage of logistic regression:

$$P = \frac{1}{1 + e^{-\left(\beta_0 + \sum_{i=1}^{n} \beta_i X_i\right)}} \tag{3.4}$$

3.2.2 Decision Tree

TBD

3.2.3 Naive Bayes

TBD

Naive Bayes is a classification and probabilistic machine learning algorithm which is based on the Bayes theorem:

$$\Pr(C = c \mid E) = \frac{\Pr(C = c) \times \Pr(E \mid C = c)}{\Pr(E)} \tag{3.5}$$

where:

- $\Pr(C = c \mid E)$ is the posterior probability which is the probability that the target variable C takes on the class of interest c after taking the evidence E.
- $\Pr(C = c)$ is the prior probability of the class c is the probability we would assign to the class c before seeing any evidence E.
- $\Pr(E \mid C = c)$ is the probability of seeing the evidence E conditional on the given class c.
- $\Pr(E)$ is the probability of the evidence E.

With regards to the binary classification, we can substitute Y as a target variable instead C, and set of features X which will refer to the set of evidence E. Assuming that $Y = 1$ refers to the occurrence of given event (e.g., default), henceforth the probability of default using the NaĂŻve Bayes algorithm can be mathematically expressed as:

$$\Pr(Y = 1 \mid X) = \frac{\Pr(Y = 1) \times \Pr(X \mid Y = 1)}{\Pr(X)} \tag{3.6}$$

One of the assumptions of this algorithm is the conditional probabilistic independence among the features. Therefore, instead of computing the probability of all features together, conditional on the class event, for each feature X we will compute its probability, conditional on the class event. Hence:

$$\Pr(Y = 1 \mid X) = \prod_{i=1}^{n} \Pr(X_i \mid Y = 1) \tag{3.7}$$

With regards to the conditional independence, we can also derived the probability of evidence E or set of features X respectively, as a sum of the probability of given set of features, conditional on class one (event), and of the probability of given set of features, conditional on one class two (non-event). Therefore:

$$\Pr(X) = \Pr(X \mid Y = 1) \times \Pr(Y = 1) + \Pr(X \mid Y = 0) \times \Pr(Y = 0) \tag{3.8}$$

Therefore:

$$\begin{aligned} \Pr(X) = & \prod_{i=1}^{n} \Pr(X_i \mid Y = 1) \times \Pr(Y = 1) + \\ & \prod_{i=1}^{n} \Pr(X_i \mid Y = 0) \times \Pr(Y = 0) \end{aligned} \tag{3.9}$$

Finally, we can derive the final formula for NaĂŻve Bayes the posterior probability as:

$$\frac{\prod_{i=1}^{n} \Pr(X_i \mid Y = 1)}{\prod_{i=1}^{n} \Pr(X_i \mid Y = 1) \Pr(Y = 1) + \prod_{i=1}^{n} \Pr(X_i \mid Y = 0) \Pr(Y = 0)} \tag{3.10}$$

3.2.4 K-Nearest Neighbors

TBD

3.2.5 Random Forest

ratatatat (Rigatti 2017)

TBD

3.2.6 Gradient Boosting

TBD

3.2.7 Support Vector Machine

TBD

3.2.8 Neural Networks

TBD

3.3 Evaluation Metrics

TBD

This section focuses on particular measures through which it is possible to determine a predictive power of model in terms of its performance. The are many ways, how to evaluate the model's performance, therefore, only the most common ones and the most relevant are further described. Note, since default prediction regards classification tasks, therefore regression's evaluation metrics are omitted.

3.3.1 Confusion Matrix

TBD

Confusion matrix is a table which summarizes the classification model's performance with respect to the actual classes and predicted classes. It is a square $n \times n$ matrix, where n determines number of classes within the target variable. Let us denote the confusion matrix as $C(f)$ for classification algorithm f. Its elements can be denoted as $c_{i,j}$ where i and j refer to the row and column indices, respectively, or more particularly, i refers to the actual class and j to the class predicted by the classifier f. Each element of the confusion matrix refers to the number of instances corresponding to actual class i and predicted class j. For instance, the element $c_2, 1$ would refer to the number of instances which

have the actual class 2 but have been classified as class 1. Mathematically, the confusion matrix can be written as following:

$$C = c_{i,j} = \sum_{l=1}^{m} [(y_l = i) \wedge (f(x_l) = j)] \tag{3.11}$$

Or either in matrix form as:

$$C_{i \times j} = \begin{bmatrix} c_{1,1} & c_{2,1} & \cdots & c_{1,j} \\ c_{2,1} & c_{2,2} & \cdots & c_{2,j} \\ \vdots & \vdots & \ddots & \vdots \\ c_{i,1} & c_{i,2} & \cdots & c_{i,j} \end{bmatrix} \tag{3.12}$$

From the given matrix, the diagonal elements represent the numbers of correctly classified instances, whereas the non-diagonal elements represent the numbers of misclassified instances. Further, let us consider a binary classification - hence, the confusion matrix will have a form of 2×2.

$$C_{2 \times 2} = \begin{bmatrix} c_{1,1} & c_{1,2} \\ c_{2,1} & c_{2,2} \end{bmatrix} \tag{3.13}$$

We can this rewrite confusion matrix as:

$$C_{2 \times 2} = \begin{bmatrix} TP & FN \\ FP & TN \end{bmatrix} \tag{3.14}$$

where:

- TP is the True Positive which refers to the number of instances which correspond to the actual class $True$ and indeed have been correctly classified as class $True$.
- FP is the False Positive which refers to the number of instances which correspond to the actual class $True$, but have been incorrectly classified as class $False$. In the statistics and hypothesis–testing terms, it can be also called as Type 1 Error.
- FN is the False Negative which refers to the number of instances which correspond to the actual class $False$, but have been incorrectly classified as class $True$. In the statistics and hypothesis–testing terms, it can be also called as Type 2 Error.

- TN is the True Negative which refers to the number of instances which correspond to the actual class $False$ and indeed have been correctly classified as class $False$.

3.3.2 Accuracy

TBD

$$Accuracy = \frac{TP + FN}{TP + TN + FP + FN} \tag{3.15}$$

3.3.3 Recall

TBD

$$Precision = \frac{TP}{TP + FN} \tag{3.16}$$

3.3.4 Precision

TBD

$$Precision = \frac{TP}{TP + FP} \tag{3.17}$$

3.3.5 F1 Score

TBD

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall} = \frac{2 \times TP}{2 \times TP + FP + FN} \tag{3.18}$$

3.3.6 AUC

TBD

In order to derive Area Under the Curve (AUC), first we need to define Receiver Operating Characteristics (ROC) curve. ROC curve is two-dimensional visualization of the model performance as a probability curve in terms of True

Positive Rate (TPR) and False Positive Rate (FPR) based on varying the given threshold.

Briefly, it can be construct as following: First, we need to sort the instances by the predicted probability and based on the given probability, we set a threshold - what will be above the threshold will be classified as $True$ instance and what is below the threshold will be classified as $False$ instance. Based on these classified instances, the confusion matrix can be constructed and via which we can compute the TPR and FPR values. Thus, if the probability is 1, the threshold will be 1 as well and hence:

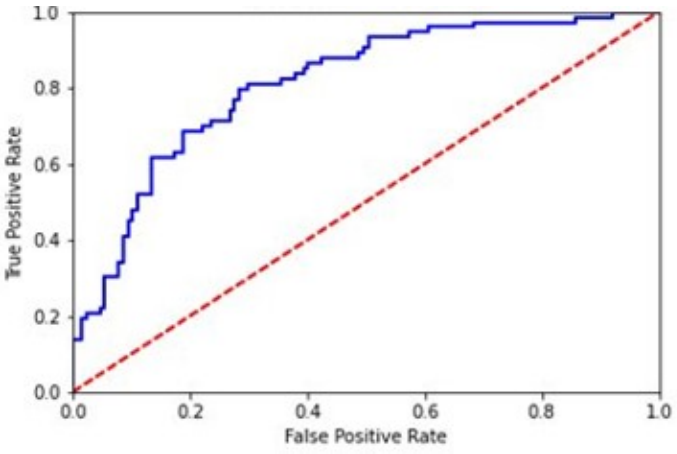
- TPR will be 0 because there is no probability which is higher than 1 and hence, everything will be classified as $False$ which will result into TP of 0, and subsequently into TPR of 0 as well.
- FPR will be 0, too â€“ since everything will be classified as $False$, therefore FP will be 0 which implies FPR to be 0, too.

On the other hand, if the probability is 0, the threshold will be 0 as well and hence:

- TPR will be 1 because there is no probability which is lower than 0 and hence, everything will be classified as $True$ which will result into FN of 0, and subsequently into TPR of 1.
- FPR will be 1, too â€“ since everything will be classified as $True$, therefore TN will be 0 which implies FPR to be 1.

Thus, based on each threshold, the TPR and FPR will be to coordinates for single point within the graph and based on such points, we can construct the ROC curve. Such visualization on the following Figure 3.2. Note the diagonal line represents a random model which randomly and correctly predicts the $True$ and $False$ classes in such way, that FPR and TPR are the same. Logically, a decent model should perform better than the random model, thus it the ROC curve should be above the diagonal line. Intuitively, the best possible theoretical model would have TPR of 1 and FPR of 0, meaning that all the $True$ actual classes should be predicted as $True$ and all the $False$ actual classes should not be classified as $True$ Within the ROC curve, the given curve reaches the left top corner which corresponds to the coordinates of TPR and FPR.

Figure 3.2: ROC Curve



Source: Author's results in Python.

AUC is basically the representation of ROC curve as a single number as it aggregates the performance on all possible thresholds. AUC can be interpreted as the probability that the randomly chosen actual $True$ instance is ranked higher than the randomly chosen actual $False$ instance. Since ROC curve is a probability curve, thus it is considering distribution curve of TP and distribution class of TN, separated by particular threshold â€“ hence, TP would have probability scores above the given thresholds, whereas TN would have probability scores below the threshold. If these curves do not overlap, meaning the model can perfectly distinguish between the $True$ and $False$ values, therefore the AUC would be 1 and the ROC curve would reach the left top corner. However, this idealistic situation does not occur in the practice at all, but rather the two distributions are overlapping since the misclassification of the classes takes the place. The bigger overlap, the lower AUC is. If the distributions are completely overlapping, it implies the AUC of 0.5, meaning that the model cannot distinguish between the $True$ and $False$ classes, which is the worst scenario. On the other hand, if the distributions are totally opposite (meaning that the TP instances would have probability scores below the given threshold, whereas the TN instances would have probability scores above the given threshold), the AUC would be 0 since the model is predicting the $True$ actual classes instead of $False$ and vice versa.

As the AUC is an area present underneath the ROC curve, mathematically, it can be computed with the definite integral where x is the given threshold:

$$AUC = \int_0^1 TPR\left(FPR^{-1}(x)\right) dx \tag{3.19}$$

3.3.7 Kolmogorov-Smirnov Distance

TBD

$$KS = \max_{x \in \mathbb{R}} \left| \hat{F}_n(x) - F(x) \right| \tag{3.20}$$

3.3.8 Somer's D

TBD

$$SD = \frac{\tau(X, Y)}{\sqrt{\tau(X, X)\,\tau(Y, Y)}} \tag{3.21}$$

3.3.9 Matthews Correlation Coefficient

TBD

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{3.22}$$

3.3.10 Brier Score Loss

TBD

$$BSL = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{3.23}$$

3.3.11 Jaccard Score

TBD

$$JC = \frac{|y \cap \hat{y}|}{|y \cup \hat{y}|} \tag{3.24}$$

3.3.12 Zero-One Loss

TBD

$$ZOL = \frac{1}{n}\sum_{i}^{n} \delta_{y_{i=1} \neq \hat{y}_i} \text{ , where } \delta_{y_i \neq \hat{y}_i} = \begin{cases} 1, & \text{if } x < 0. \\ 0, & \text{otherwise.} \end{cases} \tag{3.25}$$

3.4 Hyperparameter Tuning

TBD

3.4.1 Grid Search

TBD

3.4.2 Random Search

TBD

3.4.3 Bayesian Optimization

TBD

3.5 Imbalanced Class Distribution

TBD

3.5.1 Random Oversampling

TBD

3.5.2 SMOTE Oversampling

TBD

3.5.3 ADASYN Oversampling

TBD

3.6 Optimal Binning

TBD

3.6.1 Weight of Evidence Encoding

TBD

Chapter 4

Machine Learning Implementation

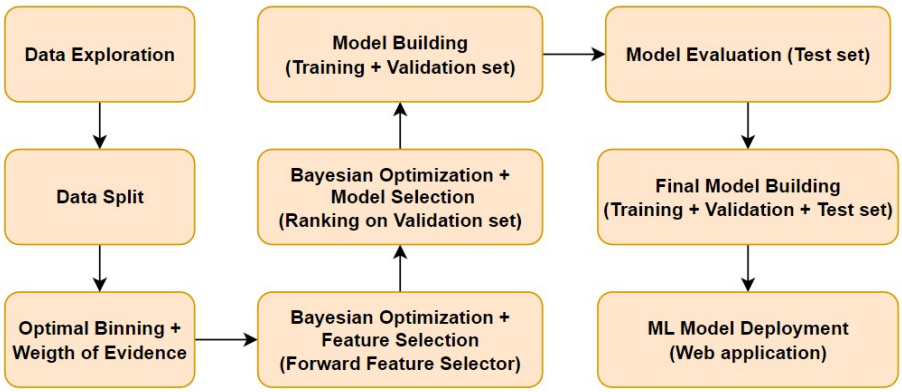
This chapter focuses on the main part of this thesis, particularly on the practical example of machine learning implementation. The machine learning framework deployed in this thesis is shown in Figure 4.1.

- **Data Exploration** - this part of the framework is focused on the exploration of the data in order to infer some insights about the data quality, distribution of the variables, statistical testing or association analysis.
- **Data Split** - this part of the framework is focused on the splitting of the data which are used separaterly in diferrent tasks such as model training, model selection and model evaluation.
- **Optimal Binning and WoE Encoding** - this part of the framework is focused on the optimal binning and the WoE encoding of the features as the main part of the feature preprocessing.
- **Feature Selection** - this part of the framework is focused on the feature selection in order to reduce the dimensionality of the data and to improve the performance of the machine learning models - each input model estimator is tuned with Bayesian Optimization.
- **Model Selection** - this part of the framework is focused on the model selection in order to find the best model based on the ranking - each input model is tuned with Bayesian Optimization on the subsets of selected features.
- **Model Building (Evaluation)** - this part of the framework is focused

on the recalibration of the final model by re-training it on the joined training and validation sets, which will be further evaluated.

- **Model Evaluation** - this part of the framework is focused on the evaluation of the final model on the unseen data from test set.
- **Model Building (Deployment)** - this part of the framework is focused on the final recalibration of the final model by re-training it on the joined training, validation and test sets, which will be further deployed into a production.
- **Model Deployment** - this part of the framework is focused on the deployment of the final model into a production as a web application.

Figure 4.1: Machine Learning Framework



Source: Author's results

4.1 Repository and Environment Structure

The whole machine learning implmenetation as the scope of this thesis is done mainly using Python Programming Language and further with collaboration of Git and HTML. The whole repository can be found in the separate appendix or is available on the GitHub repository `https://github.com/petr-ngn/FFU_VSE_Masters_Thesis_ML_Credit_Risk_Modelling`. The repository structure is shown in Figure 4.2.

Figure 4.2: Repository Structure

```
|--- data
|  |--- interim_dat.csv
|  |--- preprocessed_data.csv
|  |--- raw_data.csv
|
|--- flask_app
|  |--- inputs
|  |  |--- inputs_flask_app_dict.pkl
|  |
|  |--- templates
|  |  |--- index.html
|  |  |--- results.html
|  |
|  |--- static
|  |
|  |--- app.py
|
|--- models
|  |--- feature_preprocessing
|  |--- feature_selection
|  |--- model_selection
|  |--- objects_FINAL
|
|--- plots
|--- Masters_Thesis.ipynb
|--- README.md
|--- requirements.yml
```

Source: Author's results at GitHub

- `data` - directory containing the raw data, partially preprocessed data (`interim`) and the final preprocessed data.
- `flask_app` - directory containing the Flask application which is used for the deployment of the model. Particularly, it contains the `app.py` file which is the main back-end file of the application, the `templates` and `static` subdirectories which contain the front-end HTML files for the application, and the `inputs` subdirectory which contains the input dictionary for the application (such as the trained model, threshold, final features etc.).
- `models` - directory containing the the subdirectories of the trained and fitted objects for features preprocessing, feature selection and model selection, including the final objects used in deployment.

- `plots` - directory containing the plots generated within the main Python notebook.
- `Masters_Thesis.ipynb` - main Python notebook containing the main part of the machine learning Implementation, such as exploratory analysis, data preprocessing, training and evaluation of the models.
- `README.md` - README file containing the description of the repository.
- `requirements.yml` - file containing the list of the required packages and their specific versions used in this project.

This particular solution is developed in Python version 3.10.9 and these are the main packages and modules used in this project:

- `NumPy`, `Pandas` - for data manipulation and analysis.
- `Matplotlib`, `Seaborn` - for data visualization.
- `Scipy` - for statistical analysis.
- `OptBinning` - for optimal binning of features with respect to the target.
- `ImbLearn` - for handling imbalanced data using oversampling.
- Scikit-learn - for feature selection, model selection and model evaluation.
- Scikit-optimize - for more advanced hyperparameter optimization.
- Flask - for deployment of the model as web application.

To replicate this solution, one may download this repository as a zip file or either can clone this repository using Git to the local repository. Before running any files or scripts, it is important to set the environment for such project using the file `requirements.yml`. This can be done by running the following command in the Anaconda terminal which will create the new environment with the name `FFU_VSE_Masters_Thesis` and install all the required packages:

```
>> conda env create -n FFU_VSE_Masters_Thesis -f requirements.yml
```

Be aware of your current path directory in your terminal. In order to install the file `requirements.yml`, you need to define a path to the directory, where such file is located. To achieve this, the user has to either change the path in the terminal using `cd` command, insert the path directory before the

`requirements.yml` in terminal, or to copy the file `requirements.yml` to the current path directory. The following code shows the former approach:

```
>> cd C:\Users\ngnpe\FFU_VSE_Masters_Thesis_ML_Credit_Risk_Modelling
>> conda env create -n FFU_VSE_Masters_Thesis -f requirements.yml
```

To preserve the reproducibility of this solution and conistency of the results, the random seed is instantiated to `42`, so for instance data split, model optimization or training would be deterministic and not totally random everytime when replicating the solution.

Some `Scikit-learn` or `Scikit-optimize` objects have optional argument `n_jobs` which utilizes the number of CPU cores used during the parallelizing computation. Such argument was set to `-1`, hence all the processors are used in order to speed up the training or optimization process.

4.2 Data Exploration

This section is focused on exploration of the analyzed loan dataset, particularly on dataset description, distribution analysis and association analysis, in order to infer potential valuable insights and hypotheses which can be used in the preprocessing or modelling part.

4.2.1 Dataset Description

The analyzed dataset pertains to the HMEQ dataset which contains loan application information and default status of 5,960 US home equity loans. Such dataset was acquired from Credit Risk Analytics platform. Since this dataset regards the loan application scoring data, using macroeconomic or other external data is omitted due to the dataset characteristics as well as modelling with behavioral scoring. Thus our goal is to predict whether the loan applicant will or would default based on provided information from the loan application.

As can be seen in Table 4.1, the dataset contains 13 columns, 12 features and 1 target variable `BAD` indicating whether the loan was in default (`1`) or not (`0`). Amongst the 12 features, there are 10 numeric features and 2 categorical features, namely `REASON` which contains 2 categories - Debt consolidation

(DebtCon) and Home improvement (HomeImp), and JOB which fontains following categories - Administration (Office), Sales, Manager (Mgr), Professional Executive (ProfExe), Self-employed (Self), and Other.

Table 4.1: Dataset columns

Columns	Description	Data type
BAD	Default status	Boolean
LOAN	Requested loan amount	numeric
MORTDUE	Loan amount due on existing mortgage	numeric
VALUE	Value of current underlying collateral property	numeric
REASON	Reason of loan application	categorical
JOB	Job occupancy category	categorical
YOJ	Years of employment at present job	numeric
DEROG	Number of derogatory public reports	numeric
DELINQ	Number of delinquent credit lines	numeric
CLAGE	Age of the oldest credit line in months	numeric
NINQ	Number of recent credit inquiries	numeric
CLNO	Number of credit lines	numeric
DEBTINC	Debt-to-income ratio	numeric

Source: http://www.creditriskanalytics.net/datasets-private2.html

After the initial data inspection, data does not contain any duplicates but does contain missing values, which are summarized in Table 4.2. Most of the missing values contains the feature DEBTINC with 1,267 missing observations, whereas columns indicating default status (BAD) or requested loan amount (LOAN) do not contain any missing values, which is expected as the bank should have the available information about their loans whether they have defaulted or not, and since this dataset pertains to the application scoring, when applying for a loan, an applicant should always fill out the requested loan amount.

Table 4.2: Missing Values Summary

Columns	# NA's	% NA's
BAD	0	0.00 %
LOAN	0	0.00 %
MORTDUE	518	8.69 %
VALUE	112	1.88 %
REASON	252	4.23 %
JOB	279	4.68 %
YOJ	515	8.64 %
DEROG	708	11.88 %
DELINQ	580	9.73 %
CLAGE	308	5.17 %
NINQ	510	8.56 %
CLNO	222	3.72 %
DEBTINC	1267	21.26 %

Source: Author's results in Python

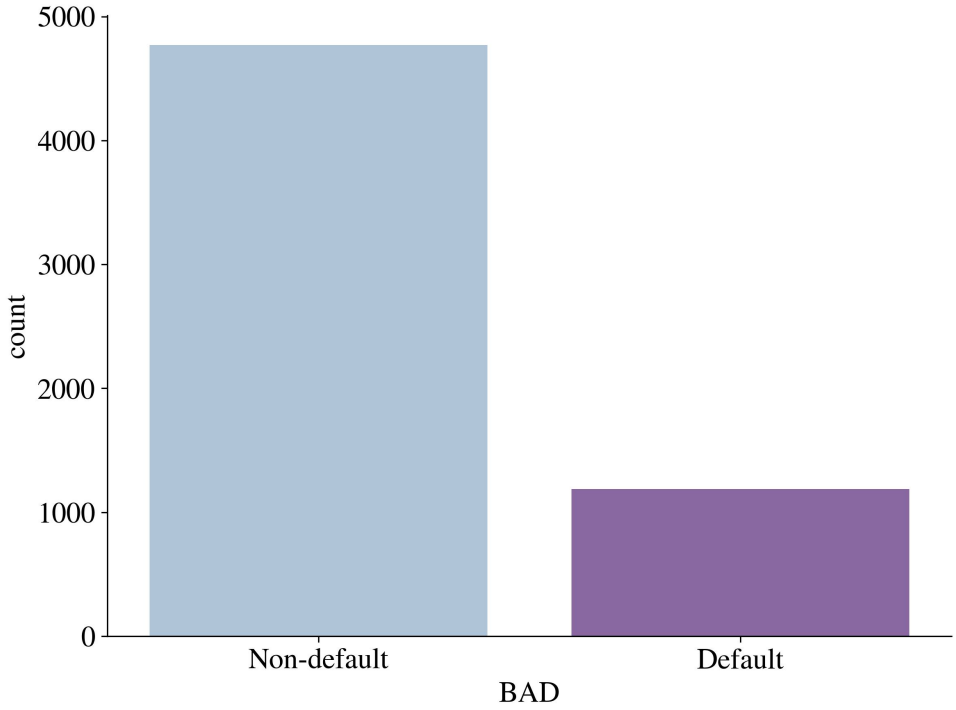
4.2.2 Distribution Analysis

In this subsection, we inspect the distribution of our variables, including the target variable and the features. Such distribution inspection may help us to identify potential outliers, missing values, and other potential issues with the dataset.

Default Distribution

Regarding the the target variable distribution, from the Figure 4.3 we can observe that the default status distribution is heavily imbalanced, as most of the loans have not defaulted yet. Particularly, 80.05% of the observations have been labelled as non-default (4,771 observations) and 19.95% observations labelled as default (1,189 observations). This may cause problems in the modelling part, as the model may be biased towards the majority class, i.e., the non-default class. Such imbalanced class issue will be further treated in Subsection 4.3.1.

Figure 4.3: Default status distribution



Source: Author's results in Python

Numeric Features' Distribution

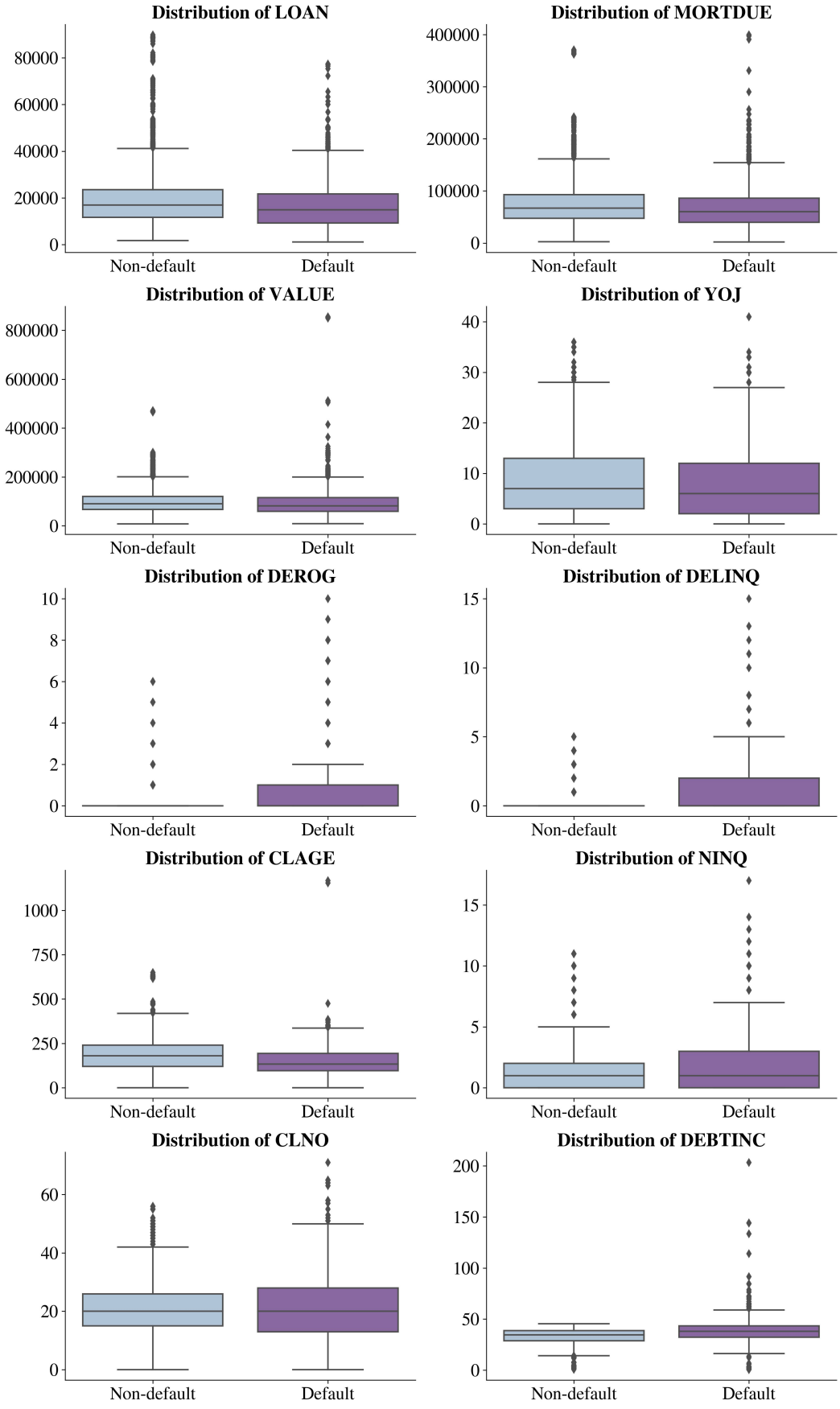
Regarding the numeric features, it can be observed that most of them exhibit a positive skewness and contain outliers, as illustrated in Figure 4.4, which depicts the conditional distribution of the numeric features with respect to the default status via boxplots.

All the outliers appear to be valid, indicating that they have not arisen due to data entry errors. This can be attributed to the non-negative nature of all the numeric features, which makes it impossible to have negative values for features such as the number of years at present job or the number of delinquent credit lines, among others. Additionally, the maximum values of the given features are not unrealistically high, further corroborating the validity of the outliers. However, it is necessary to treat these outliers as they can bias a model's weights or coefficients, particularly in the case of logistic regression or neural networks. Outliers can also jeopardize distance calculations in the case of KNN, or in general, affect the position and orientation of the decision boundary. Such factors can lead to overfitting and inaccurate and biased predictions. A detailed explanation of the outlier treatment is provided in Subsection 4.3.2.

Concerning the target variable, it can be observed that there are some dif-

ferences in the distribution shapes of DEROG and DELINQ, which exhibit less skewness and lower dispersion for non-default cases as compared to default cases. Since both features indicate negative information about delinquency, it is expected that a higher value for these features would increase the likelihood of loan default. Referring to the feature DEBTINC, it does not exhibit any extreme values for non-default cases, but some extreme values are present for default cases. From this, it can be inferred that if the debt-to-income ratio is too high, indicating that the applicant's income is not sufficient to cover their debt, the loan is more likely to end in default. The association between the default status and the numeric features is further investigated in Section 4.2.3.

Figure 4.4: Conditional distribution of numeric features



Source: Author's results in Python

Due to the fact that the boxplots do not capture the missing values occurred in given features, it is also important to inspect the numbers and proportions of missing values in each feature, conditional on the default status. As can be seen in Table 4.3, n_0 refers to the number of missing values in given feature for non-default cases, n_1 refers to the number of missing values in given feature for default cases. N_0 and N_1 refer to the total number non–default cases and default cases respectively, therefore n_0/N_0 refers to the proportion of missing values in given feature for non-default cases, and n_1/N_1 refers to the proportion of missing values in given feature for default cases.

Pertaining to the feature `DEBTINC`, we can observe a significant difference in the number of missing values between the default and non-default cases. Out of all defaulted loans, 66.11 % had missing debt–to–income ratio, whereas only 10.08 % out of all non-defaulted loans had missing debt–to–income ratio. Therefore, there could be a strong association between the missing debt–to–income ratio and the default.

Similarly, the table depicts a significant difference with respect to `VALUE` as 0.15 % had missing collateral property value out of all non–defaulted loan, and 8.92 % defaulted loans had missing collateral property value. It can be inferred that loan applicants who withhold information on their collateral property value or debt-to-income ratio are more likely to default on their loans. This may be due to negative information that they are trying to conceal, such as an excessively high debt or low income, or a low collateral property value. Such associations are further investigated in Section 4.2.3.

Table 4.3: Numeric features NA's table

Feature	n_0	n_1	n_0/N_0	n_1/N_1
LOAN	0	0	0 %	0 %
MORTDUE	412	106	8.64 %	8.92 %
VALUE	7	105	0.15 %	8.83 %
YOJ	450	65	9.43 %	5.47 %
DEROG	621	87	13.02 %	7.32 %
DELINQ	508	72	10.65 %	6.06 %
CLAGE	230	78	4.82 %	6.56 %
NINQ	435	75	9.12 %	6.31 %
CLNO	169	53	3.54 %	4.46 %
DEBTINC	481	786	10.08 %	66.11 %

Source: Author's results in Python

Categorical Features' Distribution

Regarding the distribution of categorical features, the dataset includes 2 nominal features, namely `REASON` and `JOB`. The conditional distribution of categorical features on the default status is visualized using barplots in Figure 4.5. The plot indicates that most loan applicants applied for debt consolidation, while most job occupancies were labeled as `Other`.

With respect to the default status, there appears to be no significant difference between the default and non-default cases in terms of the relative distribution of the `REASON` feature. However, a slight difference is observed between the default and non-default cases in terms of the relative distribution of the `JOB` feature. Specifically, the categories `Office`, `ProfExe`, and `N/A` exhibit a relatively higher proportion of non-default cases than default cases. Hence, a moderate association between the `JOB` feature and the default status is possible, as further investigated in Section 4.2.3.

Figure 4.5: Conditional distribution of categorical features

Source: Author's results in Python

4.2.3 Association Analysis

In this subsection, we aim to examine potential relationships between the variables by analyzing their associations. Firstly, we investigate the association between the default status and the features. Subsequently, we explore the association among the features themselves.

Association between default status and numeric features

To measure the association between the target variable and the numeric features, we use the Point-Biserial correlation coefficient, which is the Pearson's product moment correlation coefficient between a continuous variable and a dichotomous variable (Kornbrot 2014). This coefficient ranges from -1 to +1

and can be used to assess the strength and direction of the relationship between a continuous variable and a binary variable. The formula for computing this coefficient is as follows:

$$r_{pb,X} = \frac{\mu\left(X|Y=1\right) - \mu\left(X|Y=0\right)}{\sigma_X} \sqrt{\frac{N\left(Y=1\right) \times N\left(Y=0\right)}{N\left(N-1\right)}} \tag{4.1}$$

Here, $\mu\left(X|Y=1\right)$ and $\mu\left(X|Y=0\right)$ represent the means of the given numeric feature X conditional on the default status and non-default status, respectively, while σ_X denotes the standard deviation of X. The values of $N\left(Y=1\right)$ and $N\left(Y=0\right)$ indicate the number of observations with default status and non-default status, respectively, and N represents the total number of observations within the feature X.

The following Table 4.4 displays the computed Point-Biserial coefficient for each numeric feature with respect to the default status, along with its statistical significance. The results show that features such as `DEROG`, `DELINQ`, and `DEBTINC` are moderately and positively associated with the default status at the 1% statistical significance level. These findings support the observations made in Section 4.2.2 regarding the positive associations of these features with the default status. It can be inferred that these features may serve as important predictors in the model.

Table 4.4: Point–Biserial Correlation table

Feature	**Coefficient**	**Significance**
LOAN	-0.075	***
MORTDUE	-0.048	***
VALUE	-0.030	**
YOJ	-0.060	***
DEROG	0.276	***
DELINQ	0.354	***
CLAGE	-0.170	***
NINQ	0.175	***
CLNO	-0.004	
DEBTINC	0.200	***

*: $p < 0.10$, **: $p < 0.05$, ***: $p < 0.01$

Source: Author's results in Python

Association between default status and categorical features

In order to measure the strength of the relationship between the dichotomous default status and categorical variables, we employ Cramer's V, which ranges from 0 to 1 and is defined as:

$$CV_X = \sqrt{\frac{\chi^2}{N\,(k-1)}} \tag{4.2}$$

As noted in Section 4.2.2, the association between the default status and `REASON` is weak, as evidenced by the Cramer's V value being close to zero. Conversely, the association between the default status and `JOB` is slightly stronger, as the categories `Office`, `ProfExe`, and `N/A` exhibit a higher proportion of non-default cases than default cases. Both `REASON`'s and `JOB`'s associations with default status are statistically significant at the 1% significance level.

While statistical significance is important, it does not necessarily indicate that a feature is a strong predictor of the target variable. Ultimately, the usefulness of a feature in predicting the target variable is determined by the performance metrics of the model.

Table 4.5: Cramer's V Association table

Feature	**Coefficient**	**Significance**
REASON	0.038	***
JOB	0.120	***

*: $p < 0.10$, **: $p < 0.05$, ***: $p < 0.01$

Source: Author's results in Python

Association between default status and missing values

Given that the loan dataset contains missing values, it is necessary to examine whether the missingness is associated with the default status. One possible approach is to encode the feature with missing values as a binary variable, where 1 indicates the presence of a missing value and 0 otherwise.

To quantify the strength of association between the two binary variables, the Phi coefficient is used, which is defined as:

$$\phi_X = \sqrt{\frac{\chi^2}{n}} \tag{4.3}$$

In line with the finding regarding the `DEBTINC` and `VALUE` in Section 4.2.2, there is a strong and statistically significant association between the missing debt-to-income ratio and default status, and a moderate and statistically significant association between the missing collateral property value and default status, as shown in Table 4.6. Therefore, we can anticipate that these features will be crucial indicators in default prediction. Further details on feature selection are presented in Subsection 4.4.2.

Table 4.6: Phi Correlation Coefficient table

Feature	**Coefficient**	**Significance**
LOAN	0.000	
MORTDUE	0.003	
VALUE	0.254	***
REASON	0.004	
JOB	0.064	***
YOJ	0.056	***
DEROG	0.070	***
DELINQ	0.061	***
CLAGE	0.030	**
NINQ	0.039	***
CLNO	0.018	
DEBTINC	0.547	***

*: $p < 0.10$, **: $p < 0.05$, ***: $p < 0.01$

Source: Author's results in Python

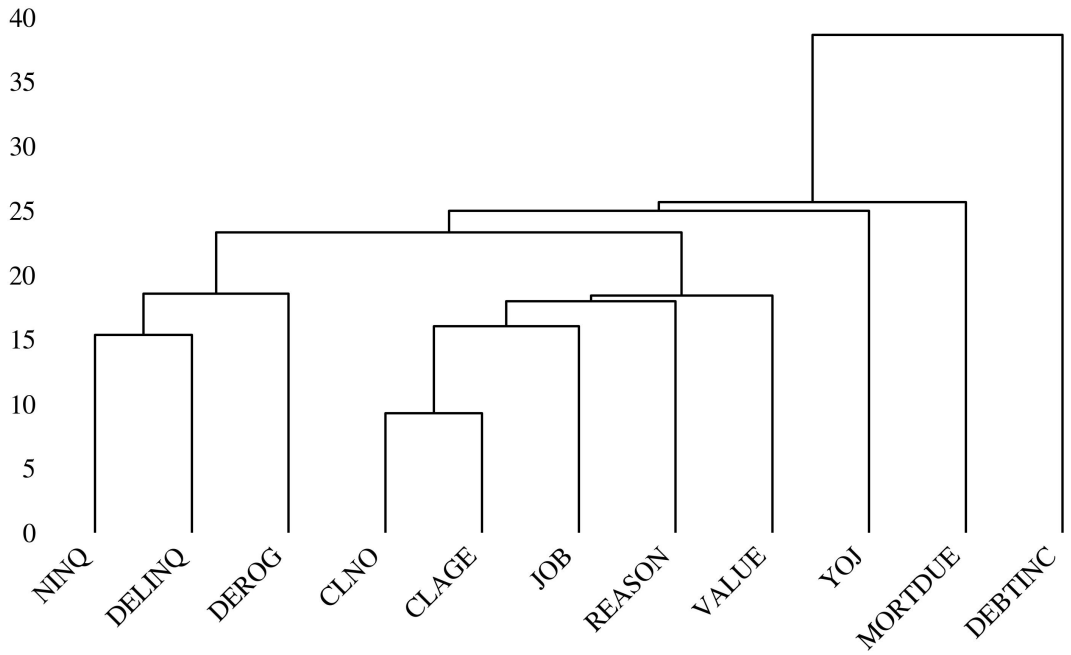
Missing Values Association

Additionally, it is imperative to investigate the relationship between missing values and default status, as well as the interrelationship between the missing values themselves. A common approach to identifying patterns of missing data in a dataset is through the use of a dendrogram, which clusters variables hierarchically based on the occurrence of missing values. This method groups variables into clusters based on the similarity of their missing value patterns, such that variables with comparable patterns of missingness are clustered to-

gether. Conversely, variables with dissimilar patterns of missingness are placed in separate clusters. The dendrogram is constructed by merging the two closest clusters iteratively until all variables are in the same cluster. The distance between the clusters at each step of the merging process is shown on the y-axis of the dendrogram, and the order in which the variables are merged is displayed on the x-axis.

In Figure 4.6, the hierarchical clustering of the dataset's variables is illustrated, excluding the default status and requested loan amount feature `LOAN`, as these variables do not contain any missing values. As depicted in the dendrogram, the `CLNO` and `CLAGE` features have the most similar patterns of missing values occurrences. Therefore, it can be inferred that a significant number of loan applicants tend to omit information regarding their number of credit lines (`CLNO`) and the age of their most recent credit line (`CLAGE`) when submitting their loan applications.

Figure 4.6: Nullity dendrogram



Source: Author's results in Python

Multicolinearity Analysis

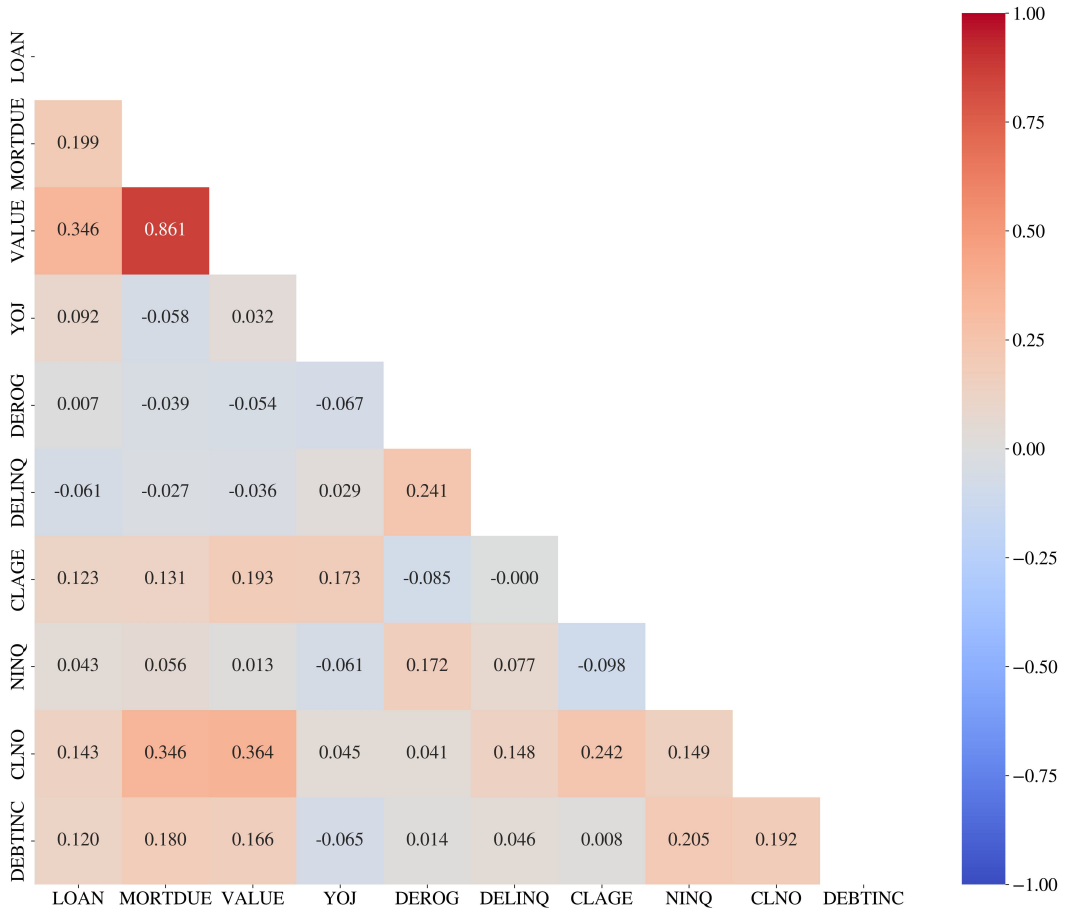
To quantify the association between the numerical features, Pearson correlation coefficient is often used. However, it is highly sensitive to outliers and makes assumptions regarding the normal distribution and linear relationship

between variables. Consequently, Spearman correlation coefficient is utilized as an alternative, as it is a non-parametric measure that does not make any assumptions regarding the distribution of variables or the linearity of their relationship. The Spearman correlation coefficient is defined as follows:

$$\rho_{spearman} = 1 - \frac{6 \sum_{i=1}^{n} d_i^2}{n\left(n^2 - 1\right)} \tag{4.4}$$

In the Figure 4.7, we can observe a very strong correlation between the `MORTDUE` and `VALUE` features. Such multicolinearity can cause problem in predictions na model's overfitting. Therefore, a feature selection is recommended - such selection is further described in Subsection 4.4.2.

Figure 4.7: Spearman Correlation Matrix



Source: Author's results in Python

4.3 Data Preprocessing

In this section, the process of preprocessing data is described as the crucial step in the machine learning modelling. Particularly, the following is described:

- Splitting the data
- oversampling,
- discretization,
- Weight–of–Evidence encoding.

4.3.1 Data Split and ADASYN Oversampling

To ensure appropriate model training and unbiased evaluation, it is necessary to split data into separate sets for various purposes. Specifically, the data was split into three sets with the ratio of 70:15:15:

- Training set for model training, feature selection, and hyperparameter tuning;
- Validation set for selecting the best final model;
- Test set for the final evaluation of the best final model.

The data is split using stratified split to preserve the default status distribution, which was highly imbalanced. Stratification ensures that the distribution of defaults and non-defaults remains the same across all sets, thereby avoiding overfitting and data leakage. Using stratification, each set had 80 % non-defaults and 20 % defaults. This method enables accurate prediction since the model is trained and evaluated on the same population (Igareta 2021).

However, stratification alone may not be sufficient for dealing with imbalanced classes. Undersampling or random oversampling may lead to loss of information or overfitting due to the minority instances' duplication. Therefore, the Adaptive Synthetic Sampling (ADASYN) technique is used for oversampling the minority class. ADASYN generates synthetic instances of the minority class based on the nearest neighbors of the minority class instances. It is more effective than SMOTE, as it generates more synthetic samples in regions where the density of the minority class is lower and fewer synthetic instances in regions

where the density is higher, thereby focusing on the difficult-to-learn minority instances. In other words, ADASYN focuses on difficult–to–learn minority instances, i.e., instances having lower density, therefore it generates more synthetic instances for such instances (He *et al.* 2008). Thereby, it makes easier for the machine learning model to learn the decision boundary between the minority and majority classes and boost the model's performance. The oversampling is performed on the training set only after the split to avoid data leakage and biased evaluation.

The following Table 4.7 shows the default distribution of the individual sets before and after oversampling. The training set after ADASYN oversampling was balanced, while the default distribution remained the same across the validation and test sets before and after oversampling, which is desirable due to stratification.

Table 4.7: WoE distribution

Set	**# instances**	**% defaults**	**% non–defaults**
Training	4,171	19.95 %	80.05 %
Training (oversampled)	6,437	48.13 %	51.87 %
Validation	895	20.00 %	80.00 %
Test	894	20.00 %	80.00 %

Source: Author's results in Python

In Python, the data are divided and oversampled using a custom function `data_split()`. This function first employs the `train_test_split()` function from the `scikit-learn` module to split the data into training, validation, and test sets with a stratification technique. Next, the `ADASYN()` class from the `imblearn` module is used to oversample the training set. This is achieved by generating synthetic instances of the minority class based on the five nearest neighbors and Euclidean distance.

However, the `ADASYN()` class from `imblearn` is not designed to handle missing values or categorical features encoded as character. To overcome this limitation, the following approach is taken:

1. Separate the categorical and numeric features;
2. Impute the missing values with arbitrary values:
    - Categorical features: string `'N/A'`;

- Numeric features: number `99999999999999` - such value is chosen since it is highly unlikely to be present in the dataset.

3. Convert the categorical features into dummy variables;
4. Join the numeric features with the dummy variables;
5. Perform the oversampling on the joined dataset;
6. Convert the dummy variables back into categorical features;
7. Retrieve back the missing values:
   - Categorical features: replace string `'N/A'` with `np.nan`;
   - Numeric features: for each feature X if its value exceeds the original maximum value, then replace it with `np.nan`;[1]

4.3.2 Optimal Binning and WoE Encoding

In the context of data preprocessing, it is crucial to consider the most appropriate feature transformation method that optimizes the performance of machine learning models. Although common approaches such as dummy encoding, standardization, logarithmic transformation, and normalization are widely used, they may not always be suitable for a given dataset due to the presence of certain characteristics. For instance, dummy encoding may not be suitable for categorical features with a large number of categories as it could lead to the curse of dimensionality. Standardization may not be appropriate for features with a large number of outliers as it may result in a loss of information. Similarly, logarithmic transformation may not be appropriate for features with a large number of zeros, and normalization may not be suitable for features with a significant number of outliers.

Therefore, alternative approaches such as discretization or binning are increasingly being used. This approach enables the identification of outliers within bins, which is not feasible with standardization or normalization. Additionally, discretization can capture missing values without requiring the removal or imputation of such values. As a result, binning is a more flexible and versatile feature transformation method that can effectively handle different types

[1]The theory behind this ADASYN will be described later.

of datasets and is particularly useful in cases where other methods may not be appropriate.

In this thesis, we employ the `BinningProcess` from the `optbinning` module in Python for an optimal binning of both numeric and categorical features. This approach involves grouping the values of a continuous variable into discrete intervals, or "bins", based on their relationship with the target variable. Similarly, for categorical features, the approach involves grouping the categories based on their relationship with the target variable. The optimal binning is performed with the objective of achieving maximum separation between the classes of the target variable within each bin. This is done to ensure that the bins are highly informative with respect to the target variable, and that each bin contains a meaningful range of values. Furthermore, the resulting bins are encoded into numeric values using the Weight-of-Evidence (WoE) approach. The WoE is a commonly used measure of the strength of association between a binary target variable and an independent variable. It is calculated as

$$WoE_{X,b} = \ln\left(\frac{\Pr\left(X = b \mid Y = 0\right)}{\Pr\left(X = b \mid Y = 1\right)}\right) \tag{4.5}$$

he following Figure 4.8 depicts the distribution of Weight-of-Evidence (WoE) bins for each feature. It can be observed that binning captures either linear, non-linear, monotonic, or non-monotonic relationships between the default status and the numeric features in terms of WoE. Regarding the `DELINQ` feature, a monotonic relationship can be observed, where the higher number of delinquent credit lines, the lower the WoE coefficient, indicating a larger distribution of defaults with respect to non-defaults in the given bin. Thus, the higher the number of delinquent credit lines, the higher the likelihood of defaulting in terms of WoE.

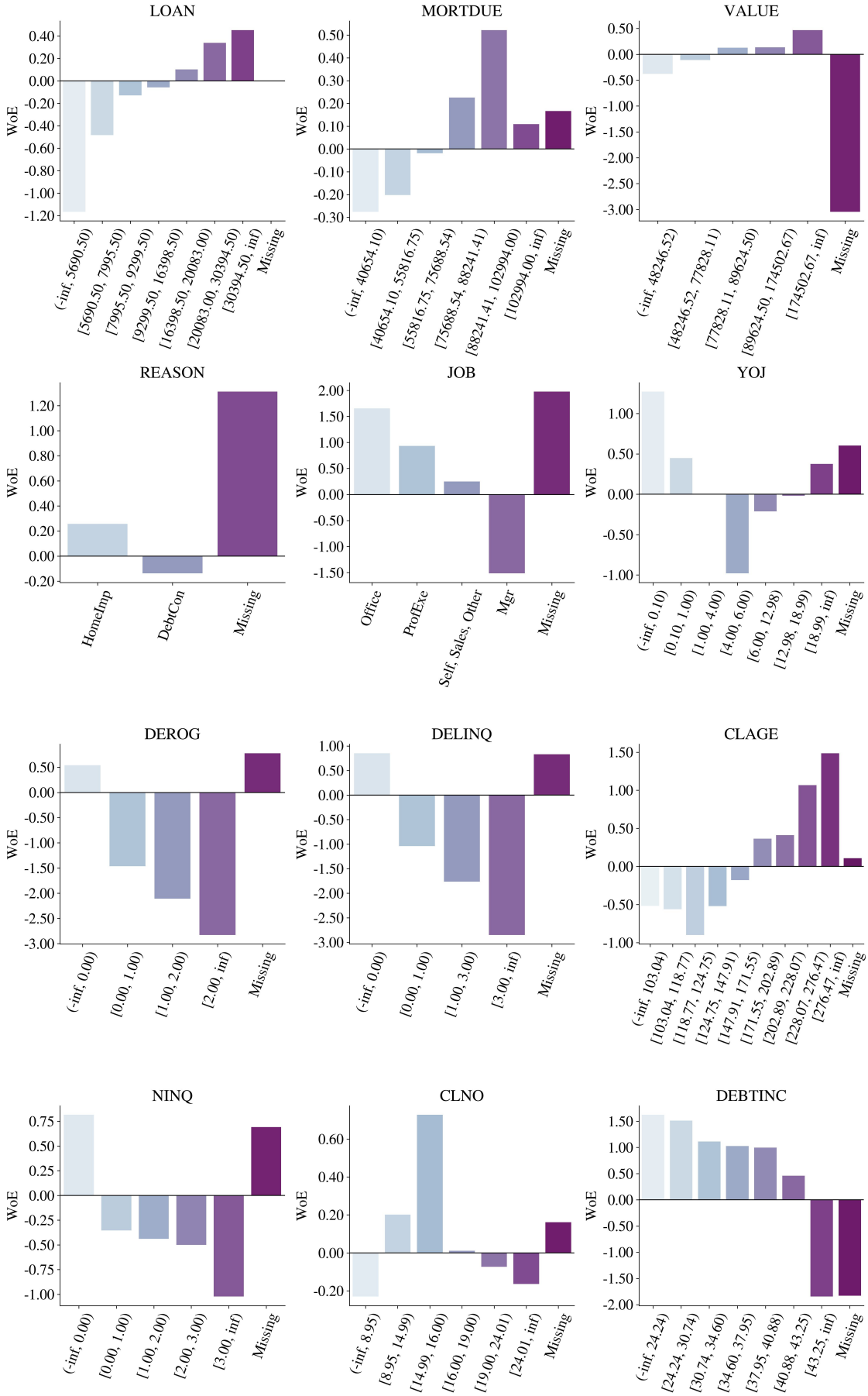
A non-linear relationship can be observed with respect to the `YOJ` feature, where the WoE coefficient is positive for applicants who have recently started working at their new job (i.e., number of years at the present job is less than 1) and for applicants who have been working at their current job for a relatively long time (i.e., number of years at the present job is higher than 19). Thus, applicants who have been working for a longer time have stable income and are more creditworthy and less likely to default. Regarding applicants who have recently started working at their new job, it is possible that they are less likely to default since the `YOJ` feature does not capture the applicant's total number

of years of work experience, but only the number of years at the present job. Thus, in the given dataset, applicants who have recently started working at their new job have a relatively higher total number of years of work experience, making them more creditworthy and less likely to default. On the other hand, for applicants who have been working at their present job between 1 and 19 years, the WoE coefficient is negative. This relationship seems to be complex and can be influenced by other factors not present in the dataset, such as the applicant's age, total number of years of work experience, education, etc.

Both numeric and categorical features contain a separate bin capturing missing values, which can be a useful indicator when training a model. This is evident in the `DEBTINC` feature, where the bin capturing missing values has the most negative WoE coefficient, indicating that there is a larger distribution of defaulters compared to non-defaulters. This finding was already raised in Section 4.2.3 in terms of the strong and statistically significant association between the default status and the missing values in `DEBTINC`.

However, inconsistencies can also be observed in terms of distributions from exploratory analysis and the WoE coefficient distributions. One example pertains to the `Mgr` category in the `JOB` feature, where the WoE coefficient is substantially negative, indicating that among managers, there is a larger distribution of defaulters compared to non-defaulters. However, this was not observed in the distribution analysis of categorical features (Section 4.2.2), where the relative distributions conditional on the default status do not differ too much in terms of the `Mgr` category. This is a result of ADASYN oversampling, which synthetically replicates minority instances, especially those instances that are hard-to-learn. Hence defaulted loans that managers have applied for are difficult to learn, and ADASYN balances default distribution by generating default instances having `JOB` equal to `Mgr`, which results in an increase in the number of instances having `JOB` equal to `Mgr`, i.e., a larger distribution of defaulters in the `Mgr` bin, and therefore a negative WoE coefficient.

Figure 4.8: WoE Bins Distribution



Source: Author's results in Python

4.4 Modelling

Once the data are finally preprocessed, the next step regards the modelling part which includes hyperparameter tuning, feature selection, model selection and model building.

In Python, 8 different machine learning models from `Scikit-learn` module are used for the default status prediction, which are:

- Logistic Regression - `LogisticRegression()`,
- Decision Tree - `DecisionTreeClassifier()`,
- Gaussian Naive Bayes - `GaussianNB()`,
- K-Nearest Neighbors - `KNeighborsClassifier()`,
- Random Forest - `RandomForestClassifier()`,
- Gradient Boosting - `GradientBoostingClassifier()`,
- Support Vector Machine - `SVC()`,
- Neural Network - `MLPClassifier()`.

4.4.1 Hyperparameter Bayesian Optimization

In order to enhance a model's performance, it is recommended to select optimal hyperparameter values instead of relying on default hyperparameters. Grid Search and Random Search are commonly used methods for hyperparameter tuning, however, they are computationally expensive and do not guarantee to find the global optimum. Additionally, they do not consider any information from previous iterations and rather check all possible hyperparameter combinations or randomly select hyperparameter combinations, respectively.

To overcome these limitations, Bayesian Optimization is used as a hyperparameter tuning approach. This method employs a probabilistic model using a Gaussian Process to approximate the objective function of interest. By utilizing Bayesian inference, the approach updates the prior distribution over hyperparameter values based on the results of previous iterations and uses the resulting posterior distribution to guide the selection of the next set of hyperparameters to evaluate. In summary, Bayesian Optimization provides a more efficient and effective approach to hyperparameter tuning compared to Grid Search and Random Search, by utilizing prior information and Bayesian inference to guide the

selection of hyperparameters to evaluate. *Theory behind Bayesian Optimization will be described later.*

In Python, a custom function, `bayesian_optimization()`, is implemented to perform hyperparameter tuning using Bayesian Optimization. This function utilizes `BayesSearchCV` class from the `Scikit-optimize` module, with a 10-fold stratified cross-validation scheme and 50 iterations, while maximizing the F1 score. For each model, the Bayesian Optimization algorithm performs 50 iterations, searching for the best hyperparameters values that maximize the F1 score. Within each iteration, a 10-fold stratified cross-validation is conducted to evaluate the model's F1 score.

The use of `BayesSearchCV` with stratified cross-validation in the hyperparameter tuning process provides a robust and reliable approach to selecting the optimal hyperparameters for the model, while the incorporation of Bayesian Optimization enables the efficient exploration of the hyperparameter space. By maximizing the F1 score, the hyperparameters selected through this process will result in a model with improved performance for the classification task at hand.

The hyperparameter space is defined for each model as follows(*The individual hyperparameters will be later described in the theory part*):

Logistic Regression

Table 4.8: Logistic Regression - Hyperparameter Space

Hyperparameter	**Space**
Intercept	True, False
C factor	$<1 \times 10^{-6}$, 5>
Penalty	L1, L2, Elastic Net, None
Solver	lbfgs, liblinear, newton-cg, sag, saga
Class weight	None, balanced
L1 ratio	<0, 1>
Intercept scaling	True, False

Source: Author's results in Python

Decision Tree

Table 4.9: Decision Tree - Hyperparameter Space

Hyperparameter	**Space**
Criterion	Gini, Entropy
Max depth	<1, 10>
Max features	<1, `len(X.columns)`>

Source: Author's results in Python

Gaussian Naive Bayes

Table 4.10: Gaussian Naive Bayes - Hyperparameter Space

Hyperparameter	**Space**
Variance smoothing	$<1 \times 10^{-9}, 1 \times 10^{-6}>$

Source: Author's results in Python

K-Nearest Neighbors

Table 4.11: K–Nearest Neighbors - Hyperparameter Space

Hyperparameter	**Space**
# neighbors	<5, 20>
Weights	Uniform, Distance
Algorithm	Ball Tree, KD Tree, Brute, Auto
Metric	Euclidean, Manhattan, Cityblock, Minkowski

Source: Author's results in Python

Random Forest

Table 4.12: Random Forest - Hyperparameter Space

Hyperparameter	**Space**
# estimators	<100, 1000>
Criterion	Gini, Entropy, Log Loss
Max depth	<1, 10>
Max features	<1, `len(X.columns)`>
Class weight	None, balanced, subsample balanced
Bootstrap	True, False
CCP alpha	$<1 \times 10^{-12}$, 0.5>

Source: Author's results in Python

Gradient Boosting

Table 4.13: Gradient Boosting - Hyperparameter Space

Hyperparameter	**Space**
# estimators	<100, 1000>
Criterion	Friedman MSE, Squared Error
Max depth	<1, 10>
Max features	<1, `len(X.columns)`>
Loss	Log Loss, Exponential
Learning rate	<0.0001, 0.2>

Source: Author's results in Python

Support Vector Machine

Table 4.14: Support Vector Machine - Hyperparameter Space

Hyperparameter	Space
C factor	$<1 \times 10^{-6}$, 5>
Kernel	Linear, Poly, RBF, Sigmoid
Degree	<1, 10>
Gamma	scale, auto
Shrinking	True, False
Decision function shape	OVR, OVO
tol	$<1 \times 10^{-9}$, 1×10^{-3}>
Class weight	balanced, None

Source: Author's results in Python

Neural Network

Table 4.15: Multi Layer Percepton - Hyperparameter Space

Hyperparameter	Space
Hidden layer size	<5, 500>
Activation function	Identity, Logistic, Tanh, ReLU
Solver	Adam, SGD, LBFGS
Learning rate	Constant, Adaptive, Invscaling

Source: Author's results in Python

4.4.2 Feature Selection

In subsection, the process of selecting optimal features is described. Instead of using all the features in dataset, only the most relevant are chosen and the noisy ones are eliminated.

For such case, the Forward Sequential Feature Selection is used. Forward Sequential Feature Selection (SFS) is a feature selection technique used in machine learning to select the optimal features from a given dataset. It is a wrapper-based approach that selects features by using a specific model to evaluate the usefulness of each feature in the dataset. The process starts with an empty set of features and iteratively adds one feature at a time that maximizes

the performance of the model. The process continues until the desired number of features is selected or no improvement in model performance is observed (or until the stop criterion is met).

The feature selection algorithm is stated in 1 below:

Algorithm 1 Feature Selection Algorithm

1: **for** *model* ∈ *models* **do**
2: *optimized_model* ← BayesianOptimization(*model*)
3: *best_features* ← ForwardSFS(*optimized_model*)
4: **end for**

In other words, each input model is tuned using the Bayesian Optimization on the training set and on the same set, and then the optimal features are selected using the Forward SFS. Thus, when having n input models, it returns n subsets of optimal features, one per each model.

To implement Forward SFS, the custom function `bayesian_optimization()` is used to tune the model, and the `SequentialFeatureSelector` class from the `Scikit-learn` module is used for feature selection. The `Scikit-learn`'s `SequentialFeatureSelector` class performs feature selection in the forward direction by maximizing the F1 score with a 10-fold stratified cross-validation approach. The `tol` parameter is used to set a threshold for the minimum increase in performance required for adding a new feature to the optimal subset. Such parameter is set to a number which is close to zero, thus the feature selection process stops when the model's F1 score is not increasing between the two consecutive steps.

In Python, the custom function `SFS_feature_selection()` is used to implement the Forward SFS approach. This function iteratively prints the process of the feature selection as can be seen in Figure 4.9, including the current step (Bayesian Optimization or Feature Selection), the execution time, and the selected features for each model.

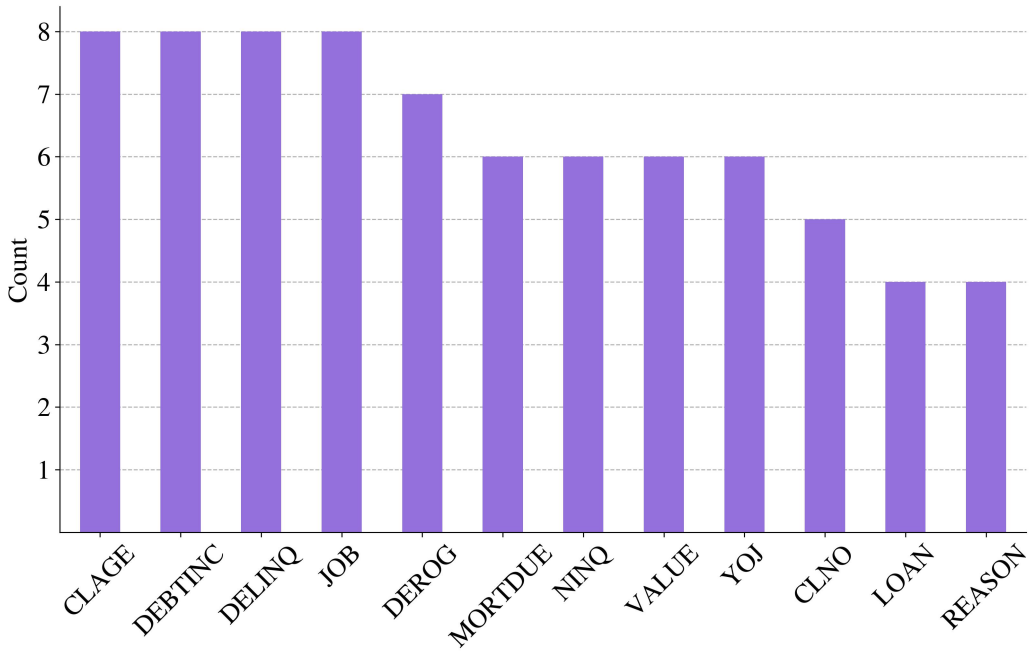
Figure 4.9: Feature Selection Print Statement

```
--------------------------------------------------------------------------------
---------------------------------------- 2/8 -----------------------------------
--------------------------------------------------------------------------------
------------------------------ FEATURE SELECTION WITH DT -------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

    1/4 ... Starting Bayesian Optimization on the whole set of features
    2/4 ... Bayesian Optimization finished
    3/4 ... Starting Forward Sequential Feature Selection
    4/4 ... Forward Sequential Feature Selection with finished

    Execution time: 0.8622 minutes

    9 features selected: VALUE, JOB, YOJ, DEROG, DELINQ, CLAGE, NINQ, CLNO, DEBTINC

--------------------------------------------------------------------------------
--------------------------------------------------------------------------------


--------------------------------------------------------------------------------
---------------------------------------- 3/8 -----------------------------------
--------------------------------------------------------------------------------
------------------------------ FEATURE SELECTION WITH GNB -------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

    1/4 ... Starting Bayesian Optimization on the whole set of features
    2/4 ... Bayesian Optimization finished
    3/4 ... Starting Forward Sequential Feature Selection
    4/4 ... Forward Sequential Feature Selection with finished

    Execution time: 0.4152 minutes

    6 features selected: MORTDUE, JOB, DELINQ, CLAGE, NINQ, DEBTINC

--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
```

Source: Author's results in Python

The following Figure 4.10 depicts the reccurrence of the selected features. As can be seen, features such as `CLAGE`, `DEBTINC`, `JOB` and `REASON` were selected by each model. On the other hand, features such as `LOAN` and `REASON` were selected by only 4 times. Therefore, we can expect that such features which were selected every time will have high importance in predictions. *the figure will be adjusted, it has been incorrectly exported.*

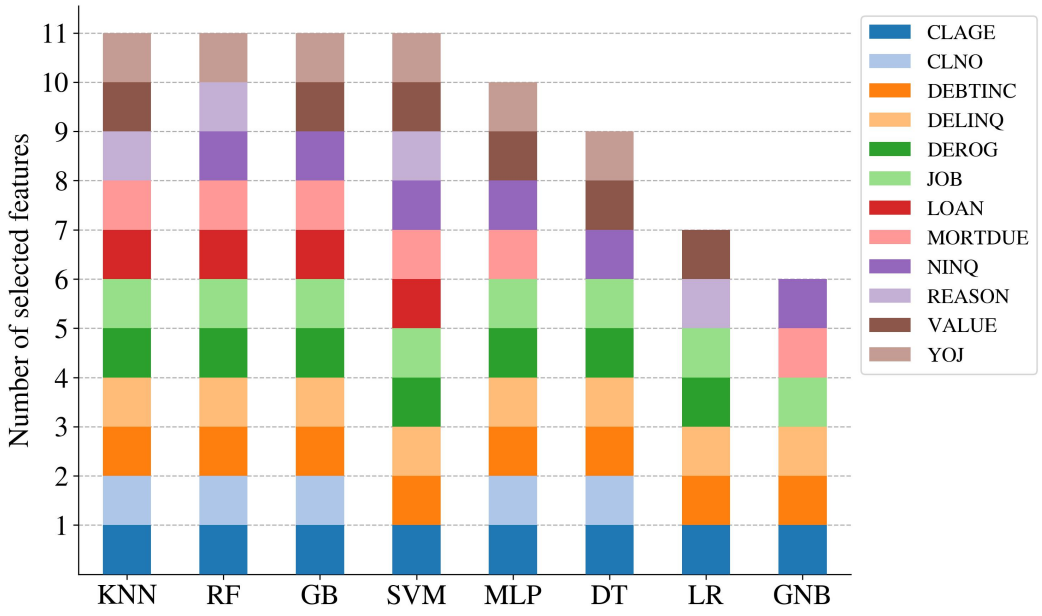
Figure 4.10: Reccurrence of Selected Features



Source: Author's results in Python

According to Figure 4.11, models such as K–Nearest Neighbors, Random Forest, Gradient Boosting and Support Vector Machine chose almost all the features as one feature feature was eliminated. On the other hand, Gaussian Naive Bayes chose only 6. It seems to be that most of the features are important as each model has selected a higher amount of features. *the figure will be adjusted, it has been incorrectly exported.*

Figure 4.11: Distribution of Selected Features per Model



Source: Author's results in Python

4.4.3 Model Selection

In combination with the pre–selected subsets of features, the next step regards the selection of the final model. The algorithm process is described in 2 below:

Algorithm 2 Model Selection Algorithm

```
for model ∈ models do
    for F ⊆ features_subsets do
        optimized_model ← BayesianOptimization(model, F)
        for metric ∈ evaluation_metrics do
            performance ← Evaluation(optimized_model, metric)
        end for
    end for
end for
```

Therefore, each input model is tuned on each subset of features selected within feature selection on the training set and subsequently, the optimized model is evaluated on the validation set. Thus, when having n input models and m subsets of selected features, we get $n \times m$ tuned models. $m \leq n$ because we exclude duplicated subset of selected features which can occur when more than one model choose the same subset(s) of features. Since there are 8 input

models and 8 unique subsets of selected features, the total number of tuned models is 64.

Further, for each model, we evalute it on the validation set by computing following metrics scores and loss functions:

- F1 score,
- Precision,
- Recall,
- Accuracy,
- AUC,
- Somers' D,
- Kolmogorov Smirnov Distance,
- Matthews Correlation Coefficient,
- Jaccard Score,
- Brier Score Loss,
- Zero-One Loss.

When evaluating classification models using class-based metrics such as F1 score, Precision, Recall, Accuracy, Matthews Correlation Coefficient, Jaccard Score, and Zero-One Loss, a default classification threshold of 0.5 is often used. This threshold separates predicted classes based on whether the predicted probability score is higher or lower than 0.5. However, in real-world use cases, the 0.5 classification threshold may not be appropriate. Therefore, it is recommended to calculate an optimal threshold rather than relying on the default one. One approach to finding the optimal threshold is to use the Younden index, which is derived from the Receiver Operating Characteristic (ROC) curve. The Younden index searches for the threshold that maximizes the sum of True Positive Rate and True Negative Rate, decreased by 1, thus:

$$J = TPR + TNR - 1 \tag{4.6}$$

Mathematically, the optimal threshold using Youden index is derived as follows:

$$T_{opt} = \text{argmax}_{t \in [0,1]} (J) \tag{4.7}$$

In Python, the `roc_curve` function from `Scikit-learn` returns False Positive Rate instead of the True Negative Rate. Nevertheless, we can derive the True Negative Rate from False Positive Rate as follows:

$$TNR = 1 - FPR \tag{4.8}$$

Therefore:

$$T_{opt} = \text{argmax}_{t \in [0,1]} \left(TPR + (1 - FPR) - 1 \right) \tag{4.9}$$

In order to ensure a more comprehensive and unbiased evaluation of a model's performance, it is recommended to consider multiple metrics rather than relying on a single metric alone. This approach provides a more generalized overview of the model's performance across different aspects and helps to prevent any bias towards a single metric. To accomplish this, models can be ranked based on their performance on each individual metric, where a higher score or a lower loss indicates a better model, resulting in a higher rank for that metric. Subsequently, for each metric, the ranking of the models is determined, and the final ranking is calculated as a weighted average of these individual rankings. The weights have been set expertly and are summarized in Table 4.16.

Specifically, the highest weight (1.5) is assigned to the F1 score, which provides a balanced measure of a model's performance with respect to both False Positives and False Negatives. This metric is commonly used in classification tasks, particularly in imbalanced datasets, such as the validation set in our case, which has not been oversampled. In addition to the F1 score, higher weight is assigned to the Recall score as well (1.2), which is a metric that penalizes False Negatives. False Negatives occur when the model predicts a negative result (i.e., no default) for an instance that is actually positive (i.e., default). In the context of loan applications, one may prefer to reject a loan applicant who would not have defaulted (False Positive) rather than approving the application of a client who would have defaulted (False Negative). Therefore, it is appropriate to give higher weight to Recall in order to reduce the likelihood of False Negatives. Henceforth, the weights are assigned to different metrics based on their relevance to the models' ranking, with the highest weight given to F1 score and additional weight given to Recall to ensure that False Negatives are minimized.

Table 4.16: Model Ranking Weights table

Metric	Weight
F1 score	1.5
Recall	1.2
Precision	1
Accuracy	1
AUC	1
Somers' D	1
Kolmogorov Smirnov Distance	1
Matthews Correlation Coefficient	1
Jaccard Score	1
Brier Score Loss	1
Zero-One Loss	1

Source: Author's results in Python

The custom function `model_selection()` iteratively prints the process of the model tuning and evaluation on each subset of features, in order to keep the track of such process as it is depicted in Figure 4.12. Particularly, it prints which model on which features is being tuned and evaluated, exeuction time, optimal threshold, F1 score on the validation set and the best hyperparameters.

Figure 4.12: Model Selection Print Statement

```
--------------------------------------------------------------------------------
------------------------------------ 56/64 -------------------------------------
--------------------------------------------------------------------------------
--------------------------- BAYESIAN OPTIMIZATION OF SVM ------------------------
--------------------------- WITH FEATURES SELECTED BY MLP ------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

1/2 ... Starting Bayesian Optimization on the subset of features (10 features):
            MORTDUE, VALUE, JOB, YOJ, DEROG, DELINQ, CLAGE, NINQ, CLNO, DEBTINC
2/2... Bayesian Optimization finished

Execution time: 22.0695 minutes

F1 Score on Validation set: 0.7102272727272726

Optimal classification threshold: 0.6477

Tuned hyperparameters of SVM:

        C: 4.999999999999999
        break_ties: False
        cache_size: 200
        class_weight: balanced
        coef0: 0.0
        decision_function_shape: ovr
        degree: 1
        gamma: scale
        kernel: rbf
        max_iter: -1
        probability: True
        random_state: 42
        shrinking: False
        tol: 1.102507160381566e-09
        verbose: False

--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
```

Source: Author's results in Python

The final output of the function `model_selection()` is table which summarizes the model's computed metrics as depicted in Table 4.17. As can be seen, the best models in terms of ranking are the Gradient Boosting models which in general have the highest score metrics and the lowest loss metrics. On the other hand, the worst-performing models are Gaussian Naive Bayes models.

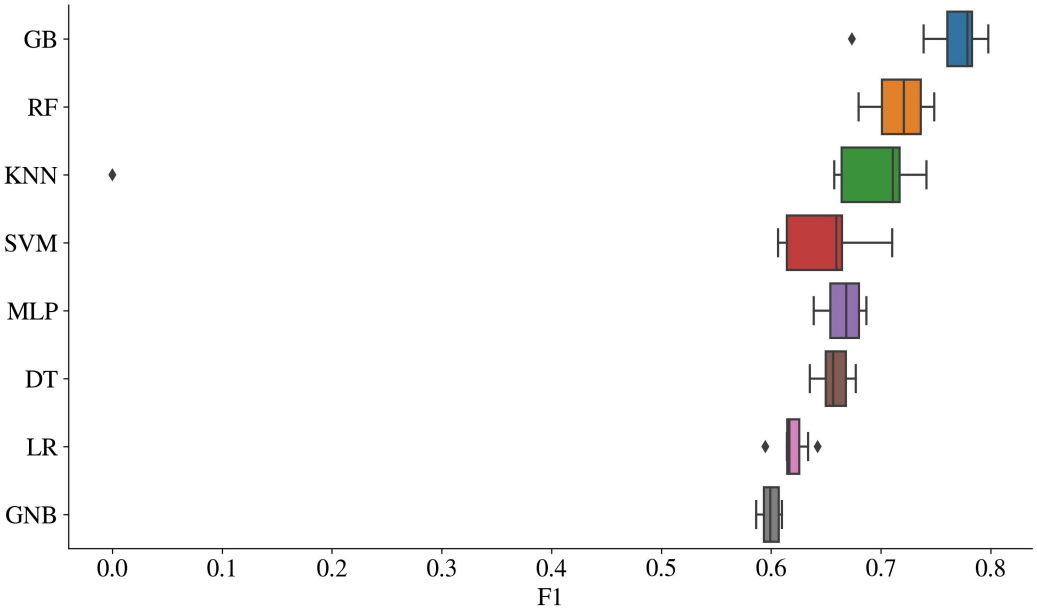
Table 4.17: Model Selection table

Tuned model	**FS model**	# Features	Time	Thres	F1	Prec	Rec	Acc	AUC	SD	KS	MCC	JC	BSL	ZOL	Log Loss	rank
GB	MLP	10	12.30	0.4955	0.7809	0.7853	0.7765	0.9128	0.9515	0.9030	0.7751	0.7265	0.6406	0.0666	0.0872	0.2487	1
GB	KNN	11	12.80	0.5072	0.7978	0.7912	0.8045	0.9184	0.9587	0.9175	0.7989	0.7467	0.6636	0.0687	0.0816	0.4135	2
GB	SVM	11	15.08	0.5053	0.7896	0.8155	0.7654	0.9184	0.9555	0.9109	0.7961	0.7397	0.6524	0.0718	0.0816	0.3486	3
GB	GB	11	11.44	0.4132	0.7799	0.7778	0.7821	0.9117	0.9543	0.9086	0.7989	0.7247	0.6393	0.0703	0.0883	0.3372	4
GB	RF	11	13.38	0.4973	0.7775	0.7841	0.7709	0.9117	0.9541	0.9081	0.7961	0.7225	0.6359	0.0669	0.0883	0.2803	5
RF	KNN	11	5.94	0.4725	0.7486	0.7326	0.7654	0.8972	0.9226	0.8452	0.7109	0.6843	0.5983	0.0923	0.1028	0.3232	6
GB	DT	9	12.00	0.4729	0.7675	0.7697	0.7654	0.9073	0.9407	0.8814	0.7556	0.7096	0.6227	0.0808	0.0927	0.4693	7
RF	GB	11	5.18	0.4517	0.7385	0.7135	0.7654	0.8916	0.9200	0.8400	0.7123	0.6709	0.5855	0.0886	0.1084	0.3135	8
RF	MLP	10	6.62	0.4761	0.7357	0.7181	0.7542	0.8916	0.9179	0.8358	0.7081	0.6679	0.5819	0.0879	0.1084	0.3084	9
GB	LR	7	16.81	0.4362	0.7388	0.7000	0.7821	0.8894	0.9219	0.8438	0.7081	0.6706	0.5858	0.0886	0.1106	0.3925	10
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
GNB	LR	7	0.39	0.2990	0.6099	0.5094	0.7598	0.8056	0.8420	0.6840	0.5866	0.5043	0.4387	0.1340	0.1944	0.6458	55
DT	GNB	6	0.89	0.5000	0.6354	0.6284	0.6425	0.8525	0.8187	0.6375	0.5838	0.5430	0.4656	0.1251	0.1475	2.1679	56
GNB	KNN	11	0.38	0.3588	0.6093	0.5219	0.7318	0.8123	0.8479	0.6957	0.5698	0.5024	0.4381	0.1367	0.1877	0.6686	57
GNB	DT	9	0.39	0.2241	0.6063	0.5095	0.7486	0.8056	0.8467	0.6935	0.5768	0.4991	0.4351	0.1316	0.1944	0.6370	58
GNB	GB	11	0.38	0.1829	0.5969	0.4893	0.7654	0.7933	0.8531	0.7063	0.5810	0.4880	0.4255	0.1310	0.2067	0.6461	59
GNB	MLP	10	0.39	0.2194	0.6013	0.5000	0.7542	0.8000	0.8493	0.6986	0.5768	0.4930	0.4299	0.1319	0.2000	0.6360	60
GNB	GNB	6	0.38	0.4787	0.5950	0.5039	0.7263	0.8022	0.8356	0.6711	0.5559	0.4835	0.4235	0.1470	0.1978	0.5280	61
LR	GNB	6	1.59	0.4561	0.5948	0.5121	0.7095	0.8067	0.8379	0.6758	0.5531	0.4831	0.4233	0.1319	0.1933	0.4180	62
GNB	SVM	11	0.38	0.1991	0.5885	0.4872	0.7430	0.7922	0.8425	0.6850	0.5712	0.4756	0.4169	0.1345	0.2078	0.6721	63
GNB	RF	11	0.39	0.3413	0.5864	0.4943	0.7207	0.7966	0.8288	0.6577	0.5545	0.4720	0.4148	0.1486	0.2034	0.7040	64

Source: Author's results in Python

In order to gain a more detailed understanding of the model selection results, the distribution of computed metrics is plotted. In Figure 4.13, the F1 score distribution is visualized for each input model. An outlier can be observed in KNN where the F1 score is 0.

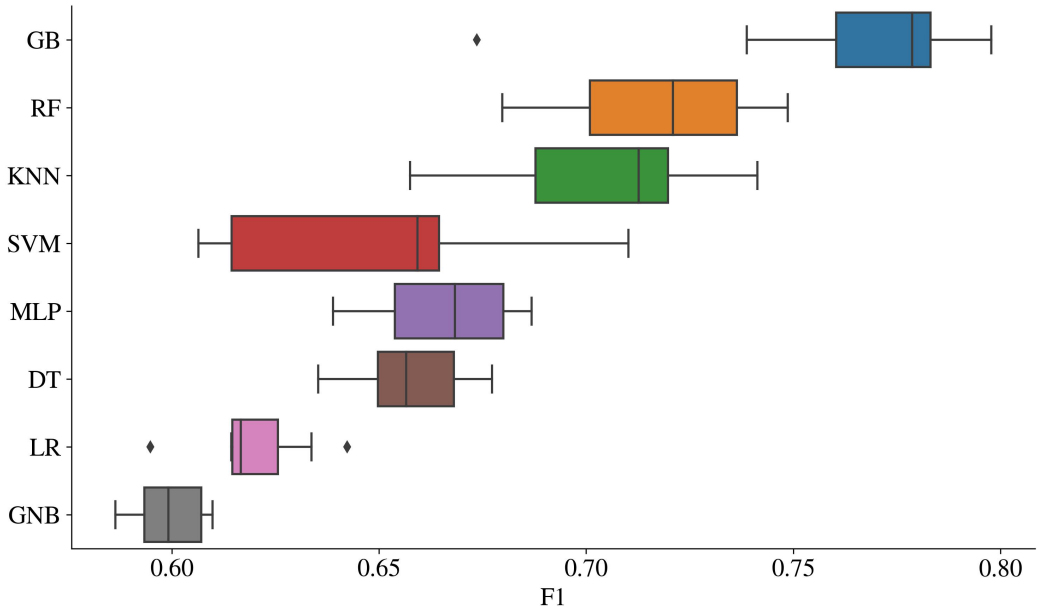
Figure 4.13: F1 score distribution



Source: Author's results in Python

Such outlier is removed in Figure 4.14 to gain a more general insight into the F1 score distribution. It can be observed that Gradient Boosting models have the highest F1 scores of around 80 %. Another tree ensemble model, Random Forest, is performing well as the second best. However, more transparent models such as Logistic Regression and Naive Bayes are performing poorly, having F1 scores around 60 %. Surprisingly, black box models such as Support Vector Machine and Neural Network are outperformed by the less complex KNN model. Nonetheless, given the relatively small sample size, KNN performs better than Neural Network and non-linear SVM on small datasets, whereas NN or non-linear SVM generally outperform KNN when it comes to large datasets.

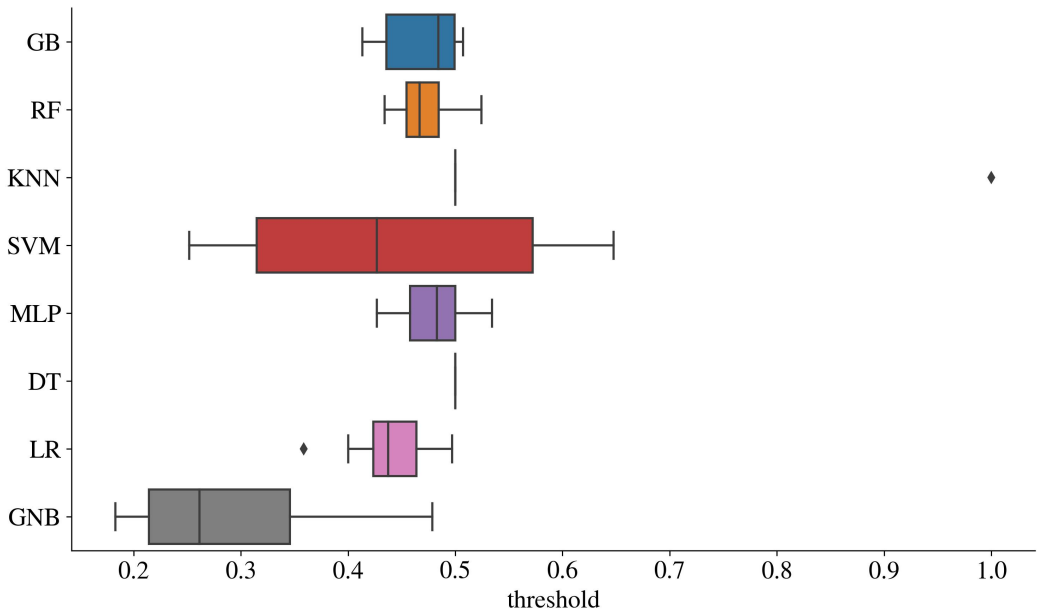
Figure 4.14: F1 score distribution - without outliers



Source: Author's results in Python

The optimal threshold distribution for each base model is presented in Figure 4.15. We can observe an outlier in KNN having a threshold value of 1, which explains the F1 score outlier found previously in Figure 4.13.

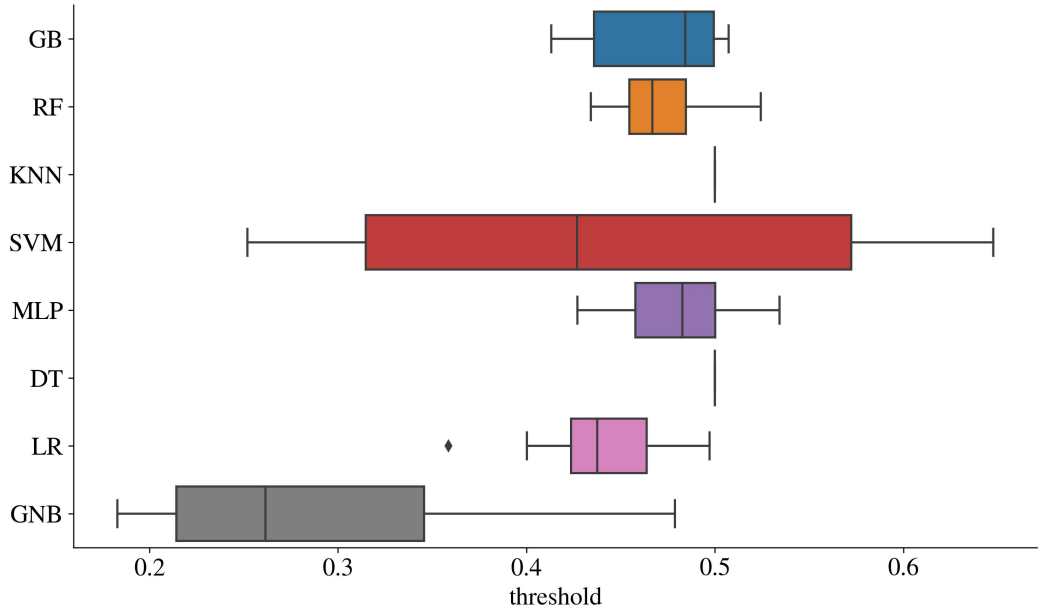
Figure 4.15: Threshold distribution



Source: Author's results in Python

In order to obtain a better insight into the distribution of optimal thresholds, the outlier in KNN is excluded, resulting in the threshold distribution depicted in Figure 4.16. The optimal threshold values are mostly distributed below 0.5, indicating that the models are generally more conservative. The most conservative model is Gaussian Naive Bayes, which has a median threshold value around 0.25.

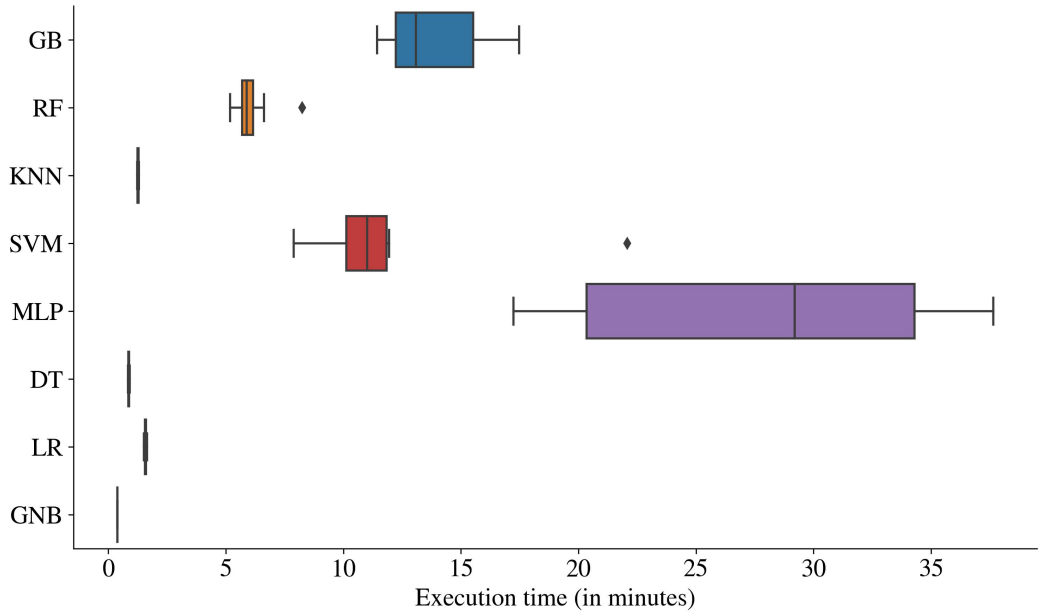
Figure 4.16: Threshold distribution - without outliers



Source: Author's results in Python

Upon examining the optimization time of each model, it can be observed that the transparent and non-complex models such as Logistic Regression, Gaussian Naive Bayes, or even Decision Tree, which take around only 1 minute to optimize themselves, also perform poorly, as already inspected in Figure 4.14. Conversely, the most time-consuming models are undoubtedly the Neural Network models, which take around 30 minutes to optimize themselves. Other time-consuming models include Gradient Boosting and Support Vector Machine, which take around 13 and 11 minutes to optimize themselves, respectively. This finding suggests that longer optimization time does not necessarily lead to better performance, as the Neural Network models are significantly outperformed by several other models.

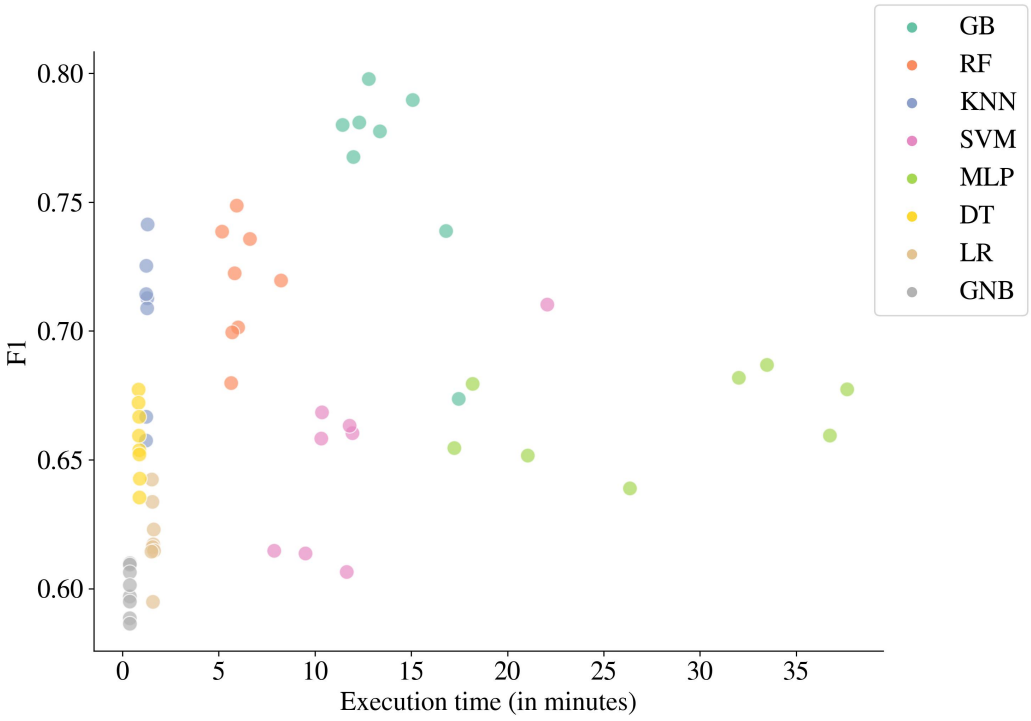
Figure 4.17: Execution time distribution



Source: Author's results in Python

The execution time and the F1 score were inspected together using a scatterplot, as shown in Figure 4.18. A cluster of non-complex and transparent models, such as Logistic Regression, Gaussian Naive Bayes, Decision Tree, and KNN, can be observed around the vertical line near 0 execution time. These models are quick to optimize, but their F1 scores are generally low, except for KNN. Further, their variance in execution time is quite low, regardless of the feature subset they are optimized on. On the other hand, the Neural Network models always perform poorly, regardless of the length of the execution time. Furthermore, the variance of the F1 scores is quite low for these models, indicating that the execution time does not have a significant impact on the F1 score in the case of Neural Networks.

Figure 4.18: Execution time vs. F1 Scatterplot



Source: Author's results in Python

To summarize this subsection, the best and final model is the Gradient Boosting Classifier which was optimized on the subset of features selected by Multi-Layer Percepton - both the model information and its final hyperparameters' values are described in Table 4.18 and Table 4.19, respectively. Such model is then used in the next modelling steps, including the recalibration, evaluation and deployment.

Table 4.18: Final Model Information

Final Model	Gradient Boosting
FS Model	Multi–Layer Percepton
Final Features	MORTDUE, VALUE, JOB, YOJ, DEROG, DELINQ, CLAGE, NINQ, CLNO, DEBTINC
Threshold	0.4955
F1	0.7809
Precision	0.7853
Recall	0.7765
Accuracy	0.9128
AUC	0.9515
SD	0.9030
KS	0.7751
MCC	0.7265
JC	0.6406
BSL	0.0666
ZOL	0.0872
Log Loss	0.2487

Source: Author's results in Python

Table 4.19: Gradient Boosting - Final Hyperparameters

Hyperparameter	**Value**
# estimators	1,000
Criterion	Friedman MSE
Max depth	10
Max features	1
Loss	Log loss
Learning rate	0.0150

Source: Author's results in Python

4.4.4 Model Building

In order to ensure that the final model performs well on unseen data, it is common practice to employ the recalibration approach, which involves retraining the model on both the training and validation sets. By doing so, the sample size used for training is increased, resulting in improved model performance. The recalibrated model is then used to evaluate the performance of the final model on the test set, which is the ultimate measure of a model's performance.

In addition to recalibrating the final model, it is crucial to recalibrate the threshold value for assigning class labels based on predicted probabilities. The optimal threshold value can be determined using the training and validation sets. In this thesis, the optimal threshold value is found to be **0.4511**, which is then used for evaluating the final model's performance on the test set. By recalibrating the threshold value, the model's performance is further improved, resulting in more accurate predictions.

Moreover, the recalibration process helps to mitigate overfitting issues, which occur when the model is only trained on the training set. By incorporating the validation set into the training process, the recalibrated model can better generalize to new data and improve its overall performance on the test set. The inclusion of the validation set during the recalibration process does not cause any data leakage issues since this set was already used during model selection to evaluate each model's performance. Therefore, using the validation data for recalibration is a sound practice that helps to ensure the reliability and accuracy of the final model.

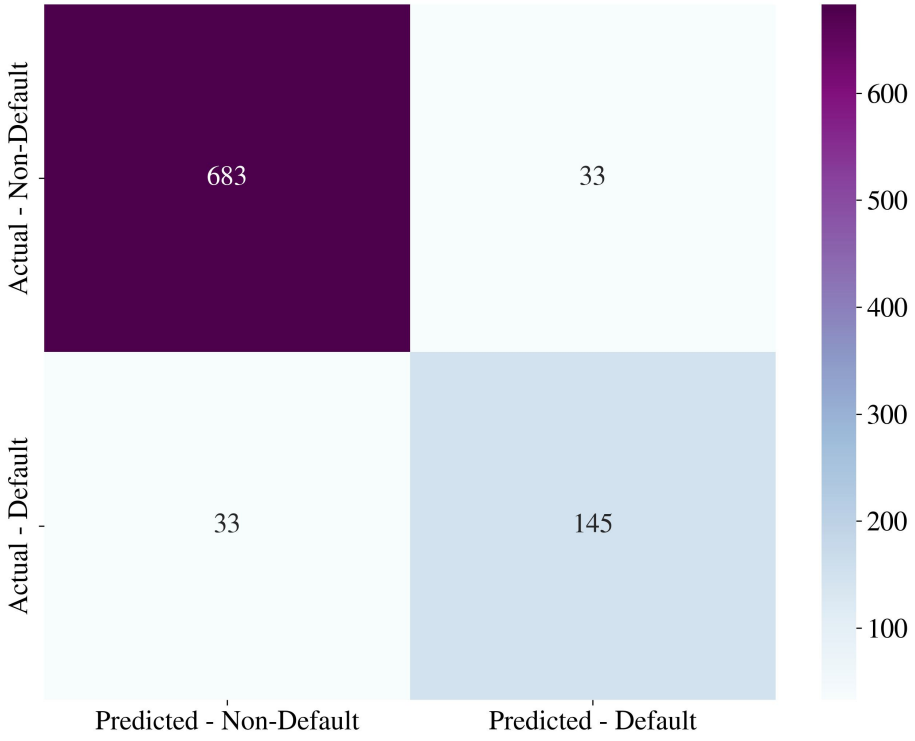
4.5 Model Evaluation

After recalibrating the model and threshold, the final step in evaluating the model's performance is to test it on previously unseen data, specifically the test set. This evaluation is critical to determine whether the model can generalize well to new data beyond the training, feature selection, and model selection phases. During the evaluation, the recalibrated classification threshold of 0.4511, determined in the model recalibration process, is also used.

In Figure 4.19, the confusion matrix for the final model, based on the test set and using the recalibrated threshold, is presented. The matrix shows that

the model is generalizing well, having correctly predicted 140 defaults and misclassified only 38 defaults and further, correctly predicted 686 non–defaults and misclassified only 30 non–defaults. Such a result indicates that the model is a good fit for the data and can provide useful predictions for the problem at hand.

Figure 4.19: Confusion Matrix



Source: Author's results in Python

In order to obtain a better understanding of the model's performance on previously unseen data, we computed several metrics that were used during the model selection process. These metrics are presented in Table 4.20. The results indicate that the model performs well on the unseen data, with most of the scores metrics around 80 % to 90 %. Furthermore, the loss metrics are relatively low, indicating that the model can effectively distinguish between defaults and non-defaults. Overall, these results suggest that the model is performing well and is suitable for predicting defaults. The results suggest that the model has a good balance between correctly identifying defaults and non-defaults, as well as minimizing false positives and false negatives. This further confirms the model's ability to accurately predict defaults.

Table 4.20: Metrics Evaluation

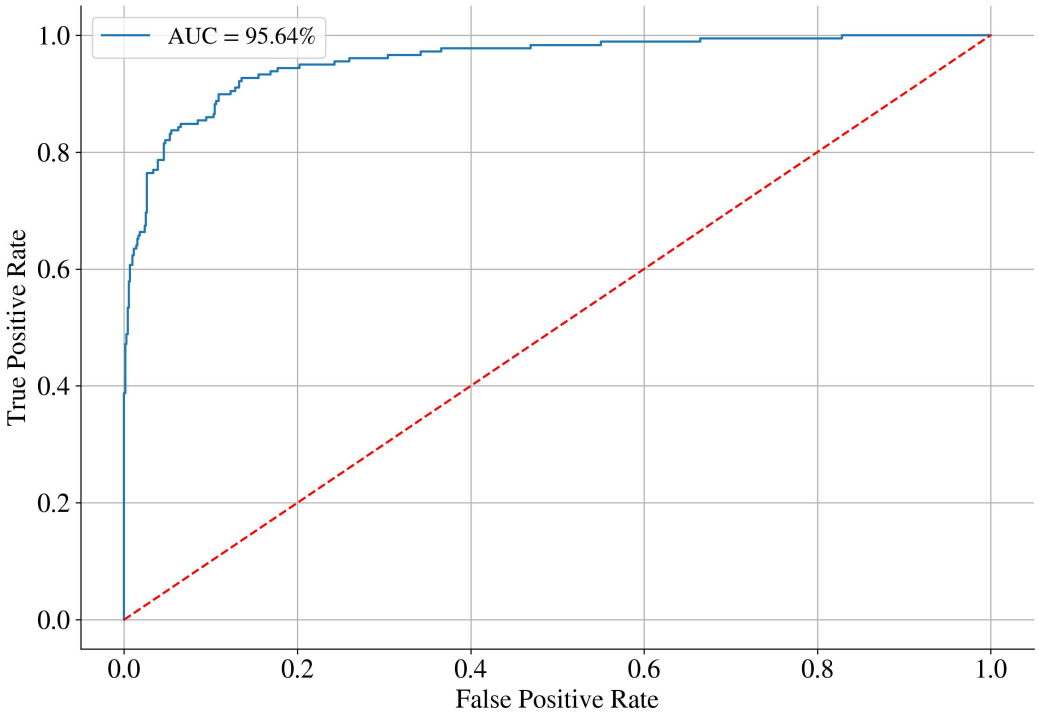
Metric	Value
F1	0.8146
Precision	0.8146
Recall	0.8146
Accuracy	0.9262
AUC	0.9564
Somers D	0.9128
KS	0.7915
MCC	0.7685
Jaccard Score	0.6872
Brier Score Loss	0.0594
Zero-One Loss	0.0738
Log Loss	0.2163

Source: Author's results in Python

To further evaluate the performance of the model, we can visualize the Receiver Operating Characteristic (ROC) curve, as presented in Figure 4.20. The curve illustrates the trade-off between the true positive rate and the false positive rate at various classification thresholds. An ideal ROC curve should have an area under the curve (AUC) value of 100 %, indicating a perfect classifier, while a random classifier would have an AUC of 50 %.

From the curve, we observe that the AUC value of the model is 95.55 %, indicating a high degree of accuracy in distinguishing between defaults and non-defaults. The curve covers most of the area under the diagonal line, indicating that the model is performing well in differentiating the two classes. Therefore, the results suggest that the model is performing well and is capable of accurately identifying potential defaulters.

Figure 4.20: ROC Curve



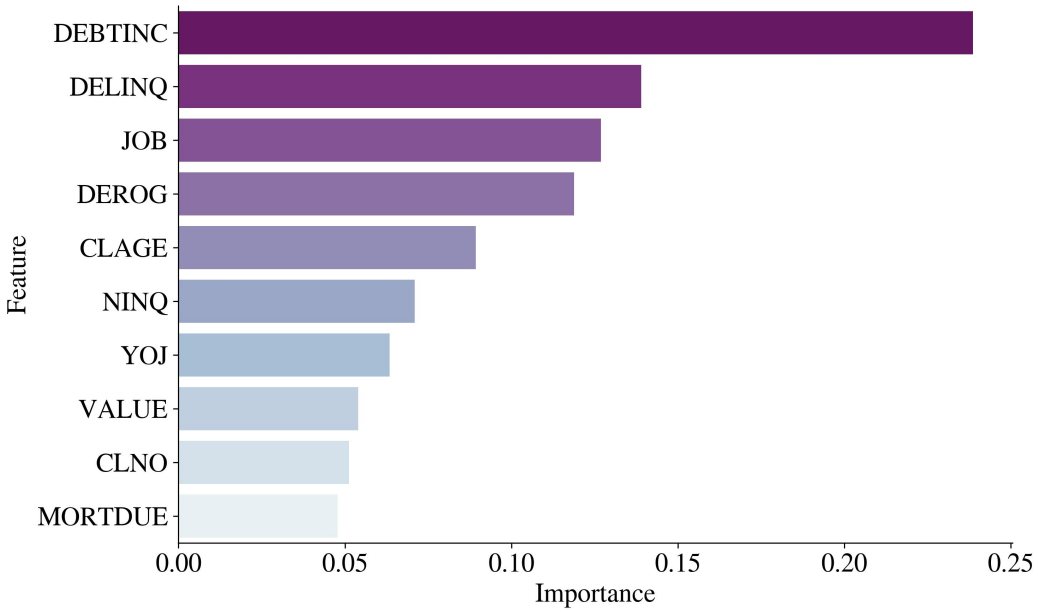
Source: Author's results in Python

To gain insights into the impact of the features used in the final model, a feature importance graph was visualized and is presented in Figure 4.21. Feature importance is a measure of how much a feature contributes to the overall performance of the model, and it is based on the reduction in impurity achieved by splitting the data on that feature. The higher the reduction in impurity, the more important the feature is considered to be. Overall, the feature importance plot provides valuable insights into the factors that are most important in predicting loan defaults. It can be used to identify which features are contributing the most to the model's accuracy and to guide future feature selection efforts. The two most important features used in the final model are `DEBTINC` and `DEROG`, which are crucial delinquency indicators in determining whether a borrower would be able to repay their loan. These two features have a significant impact on the model's ability to accurately predict loan defaults, with high feature importance scores.

Understanding the impact of individual features on model performance can be useful in identifying areas for improvement, as well as in identifying the most significant factors that drive loan defaults. By focusing on these important features, lenders and policymakers can better understand and address the

underlying factors that contribute to default risk, ultimately leading to better lending decisions and improved outcomes for borrowers and lenders alike.

Figure 4.21: Feature Importance



Source: Author's results in Python

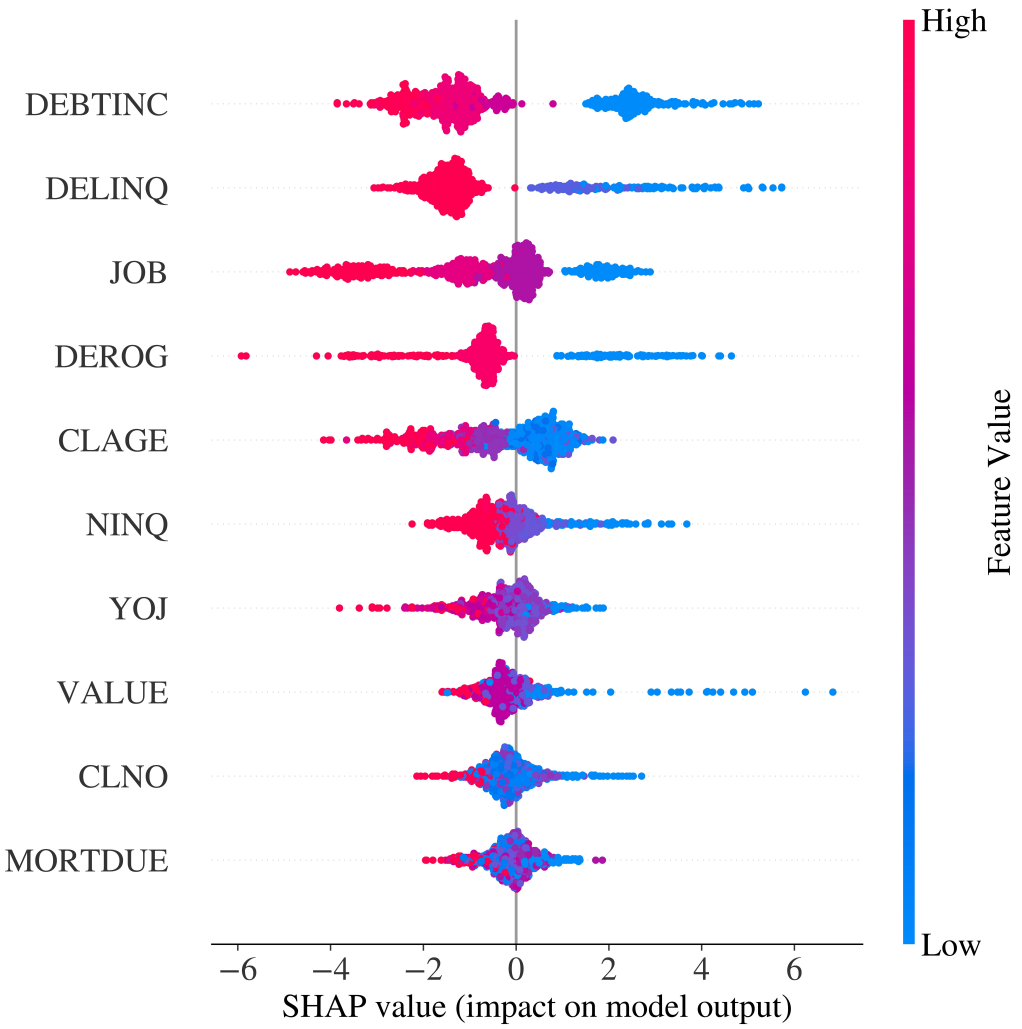
To gain further insights into the impact of the features on the final model's predictions, the SHAP (SHapley Additive exPlanations) summary plot is displayed in Figure 4.22. The SHAP summary plot provides a clear visualization of the contribution each feature makes to a prediction and is used for a black box model explainability. Each dot in the plot represents a feature and its corresponding SHAP value. The color of the dot represents the feature's value, with red indicating high values and blue indicating low values. The position of the dot on the x-axis represents the impact of the feature on the prediction, with features on the right-hand side contributing more positively to the prediction, and features on the left-hand side contributing more negatively.

Since our data points are encoded in WoE values, the higher (the more positive) value, the larger distribution of non-defaulters compared to defaulters, and vice versa, the lower (the more negative) value, the larger distribution of defaulters compared to non-defaulters. For the most important features, we can observe that the blue values (negative WoE values) and red values (positive WoE values) are quite separable. Negative WoE values are positively contributing to the predictions, and the positive WoE values are negatively

contributing to the predictions. This means that the more negative the WoE value, the more likely the borrower is to default, and vice versa.

Overall, the SHAP summary plot provides a valuable tool for interpreting and understanding the complex decision-making process of the final model. The plot enables us to examine the impact of individual features on the model's output and helps us to identify which features are most influential in the model's decision-making process. By understanding the relative importance of each feature, we can gain deeper insights into the creditworthiness assessment process and make more informed decisions in the lending industry.

Figure 4.22: SHAP Summary Plot



Source: Author's results in Python

4.6 Machine Learning Deployment

This section describes the process of taking a trained machine learning model and making it available for use in the real world. It involves taking the model from a development environment and integrating it into a production environment where it can be used to make predictions or decisions based on new data. In this thesis, the machine learning model is deployed as web application using Flask and HTML.

4.6.1 Final Model Building

Before deploying the model in a production setting, a meticulous process is undertaken to ensure optimal performance. This involves conducting a final recalibration of the model using the entire dataset, comprising the training, validation, and test sets. The purpose of this recalibration is to fine-tune the model parameters, thereby maximizing its ability to generalize and yield accurate predictions.

The test set, which has been employed previously during the model evaluation phase, is incorporated into the recalibration process without any risk of data leakage. By including the test set, the sample size for training is expanded, thereby increasing the model's capacity to generalize effectively.

Upon completion of the recalibration, the final classification threshold for deployment is determined to be **0.3358**. This value is derived from a comprehensive analysis of the training, validation, and test sets, which ensures that the model's generalization capabilities are further enhanced for optimal performance in real-world applications.

4.6.2 Flask and HTML Web Application

In this case, the machine learning model was deployed into a web application using Flask and HTML. The application is temporarily deployed on the Cloud server on the **PythonAnywhere** platform and is accessible here: `http://ml-credit-risk-app-petrngn.pythonanywhere.com/`. However, the application will be shut down after the thesis defense and will not be available online anymore due to the budget reasons. Nevertheless, the code for the ap-

plication is available in the GitHub repository, and the application can be run locally using any Python compiler.

Furthermore, prior to deployment, we need to prepare several Python inputs for the web application, including:

- Model - the final model recalibrated on the training, validation and test set.
- Threshold - the final classification threshold recalibrated on the training, validation and test set.
- Features - the final features used in the final model.
- Data Frame - he input data frame used in the web application to store the loan applicant's inputs.
- Optimal Binning Transformator - fitted `BinningProcess` object for binning and WoE-encoding of the loan applicant's inputs.
- WoE Bins - a set of bins and WoE values used for mapping the WoE values to missing values.

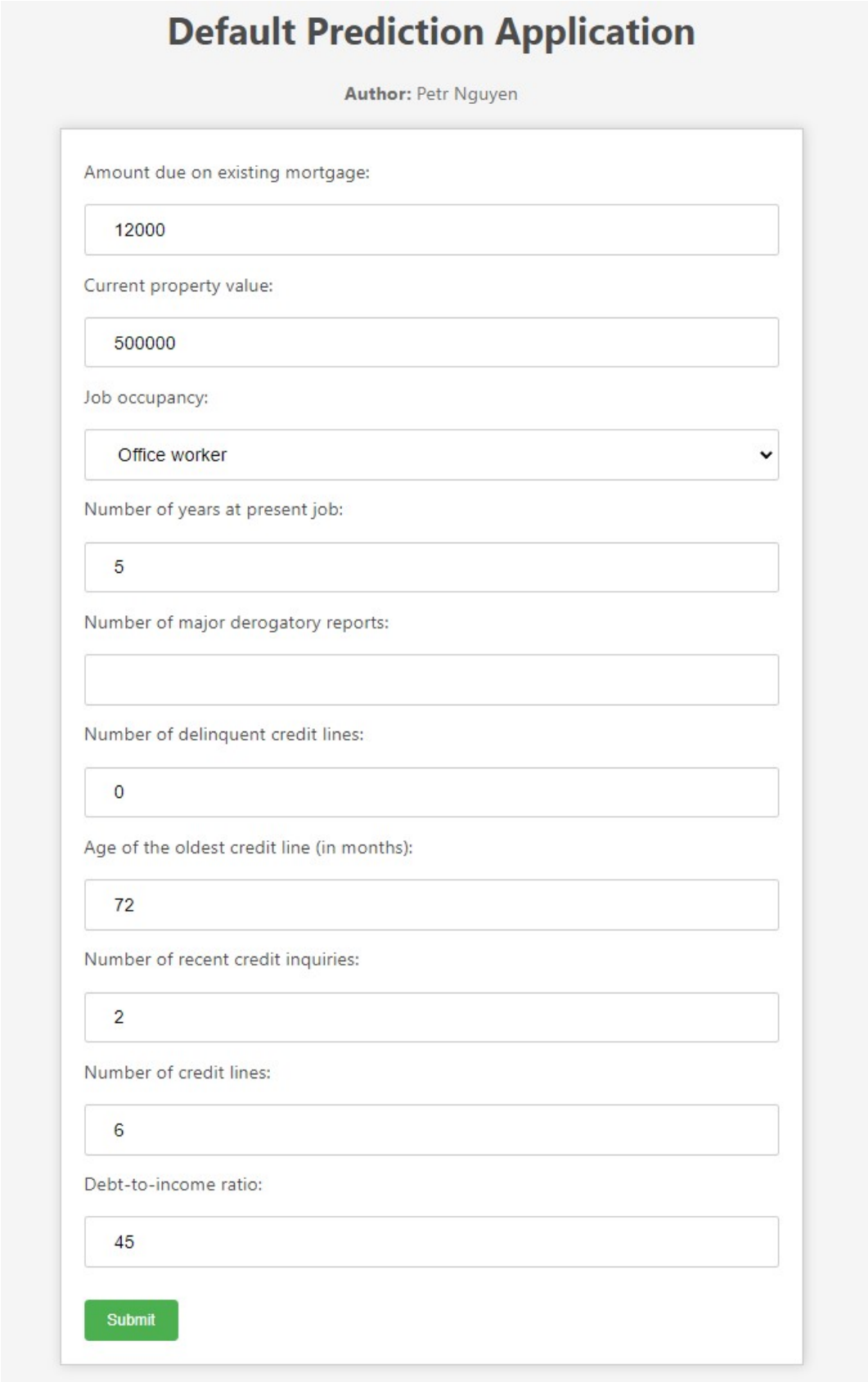
Such inputs required for the machine learning application are exported in the `.pkl` format using the `pickle` module. This format allows for efficient and easy-to-use serialization and deserialization of the inputs. The pickled file is then loaded directly into the Flask application.

For the back-end of the web application, Flask is used to deploy the machine learning model. The Flask application is coded in the `app.py` file, which is stored in the `flask_app` repository. The front-end of the web application is coded in HTML, with CSS and JavaScript elements used to enhance the user interface.

The web application first renders an HTML page, as shown in Figure 4.23. This page contains a loan application form, in which the user or the loan applicant fills in the respective field values that correspond to the features on which the machine learning model was trained. The form is designed to capture the necessary information required for the model to make a prediction about the loan applicant's application. Note that not all fields in the loan application form need to be filled out, except for the requested loan amount. This is because missing values in certain features may indicate a higher risk of defaulting. Conversely, one may choose to impose a restriction on the form,

requiring all fields to be filled out. However, this could result in a lower number of received applications from delinquent clients, as they may not have all the necessary information to complete the form.

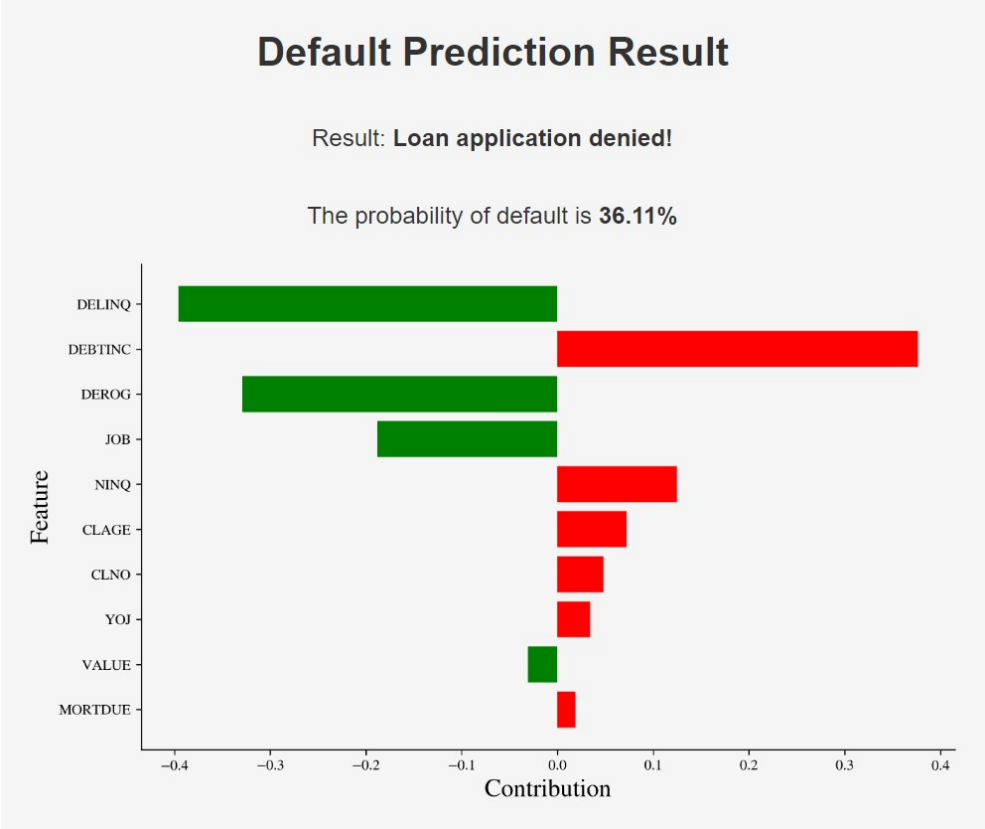
Figure 4.23: Flask Web Application Form



Source: Author's results in Python

Once, the loan application form is submitted, the web application uses the pickled input to transform the data from the loan application form and use it in the recalibrated model in order to get the result, whether the given loan applicant would repay his loan based on predetermined threshold. The result is then displayed in the web application, as shown in Figure 4.24. Particularly, the web application returns whether the loan application would be denied or approved and also the probability score of default.

Figure 4.24: Flask Web Application - Prediction Result



Source: Author's results in Python

Besides the prediction results, it also displays the Local Interpretable Model-Agnostic Explanations (LIME) of the black–box model (Gradient Boosting) with respect to the inputs submitted within the form, as shown in Figure 4.24. LIME focuses on the local explainability of the black box model around the black–box prediction as it generates a new dataset consisting of perturbed samples around the given prediction and then trains a surrogate linear model on the new dataset. Such local interpretable, surrogate model should be a good approximation of the black box model in the vicinity of the given prediction,

i.e., the local interpretable model is then used to explain the prediction of the black box model (Ribeiro *et al.* 2016).

The LIME explanation of input instances x is given as follows:

$$\xi(x) = \arg\min_{g \in G} L(f, g, \pi_x) + \Omega(g) \tag{4.10}$$

where f is the original black–box model, g is the surrogate model, L is the loss functions measuring how far is the explanation $\xi(x)$ far from the prediction produced by black–box model f, and $\Omega(g)$ is the complexity of the surrogate model g.

The explanation is given in terms of the feature importance, which is represented by the magnitude of the feature's coefficient in the local interpretable model. The higher the magnitude of the coefficient, the more important the feature is in the prediction of the black box model. The explanation is displayed in the web application in the form of a bar chart, as shown in Figure 4.24.

Chapter 5

Title of Chapter Five

5.1 Frequently made mistakes

The following checklist should help in avoiding some frequently made mistakes, if any of the following propositions apply for your thesis, there is a problem:

- You have citations in your abstract.
- The introduction does not cover the three parts as described in Chapter 1.
- The introduction contains subheadings.
- You described different aspects than promised in the title.
- You copied some parts of the text from other work without proper referencing and citing.
- You used automatic translation tools to produce text by translating it from another language.
- Your thesis contains many typos and grammatical errors. (Use an electronic spell checker. Please!)
- You used color in your figures and refer to the “blue” line (assume that your readers use a monochrome printer).
- You mainly used websites and other unrefereed material as your sources or you used Wikipedia as your source.

- You refer to something in your conclusion which you have not mentioned before.
- Some forenames in the references are abbreviated, some not.
- Some references miss a publishing date.

5.2 Useful Hints

If you write in English, you might find the following hint useful: The indefinite article a is used as an before a vowel sound—for example an apple, an hour, an unusual thing, an **MNC!** (MNC!) (because the acronym is pronounced Em-En-See). Before a consonant sound represented by a vowel letter a is usual—for example a one, a unique thing, a historic chance. Few more tips to follow:

- Don't give orders—don't write in the imperative mood—unless you are training to be a teacher.
- Avoid the use of questions. You may know the answer: does your reader? It's much safer to tell her, or him.
- Do not become entangled in the problems of 'sexist' language. It is much easier to write in the plural. "Students should check their work" is good English. "A student should check—" is also good English, but now the problems begin: "—her work?" "—his work?" Which? You can write "his or her," but that seems clumsy. Stick to the plural.
- If you must refer to yourself, use the third person such as "The present writer would recommend that . . . " may be useful.
- Use the full forms of words and phrases, not contractions like "he's," "don't," etc. Keep the apostrophe to indicate possession—and use it correctly. Academics really sneer at students who use the "Greengrocer's apostrophe."
- Do not despise short, workmanlike, and effective plain English words. If they mean what you want to say. Accurately.

- Avoid the use of humor in academic writing—unless you are very sure of yourself.
- Even when you are not being funny, avoid the use of irony or sarcasm.
- Paragraphs in academic English should contain more than one sentence. (Short paragraphs look as if you are writing for a tabloid newspaper—or a simple Template!) I guess that the average academic book runs to two or three paragraphs per page. Look at the books in your subject, and get a feel for how long your own paragraphs should be when you are imitating the academic style.
- Develop an academic vocabulary. The 'long words' you learn in the course of your studies are long usually because they have more precise meanings than their less formal equivalents. They are therefore better when you want to be accurate. (Also they allow you to sound like someone who deserves a degree.)
- Use as few words as you can; but use enough words to express your meaning as fully as you can. Your judgment of what is appropriate here is part of what you should learn throughout your course.
- Avoid lazy words such as "nice". It is usually better to say "acquire" or "obtain" than "get;" and it may be better, if you mean "through the use of money," to say "purchase" or—better still—"buy."
- A short word like "buy" is better than a long one like "purchase"—unless the long one is more accurate. A "statutory instrument" is better than a "rule"—to a lawyer, at any rate.
- Proof-read with care. Ask someone else to help—you may be too close to your work to be able to see your mistakes.
- If in doubt, choose the more formal, or possibly just the more old-fashioned, of two words. For example, say quotation rather than quote whenever you mean the use of somebody else's words.
- You will often sound more academic if you include doubts in your work—and qualifications. Within the scope of this thesis, the current writer cannot hope to cover all the possible implications of the question.

- In this context, the use of litotes sounds very academic. This is the construction where a writer uses a negative with a negative adjective, e.g. it is not unlikely that ... This does not mean the same as it is probable that ... It has a shade of meaning and qualification that can be useful to academic writers.

Text text Haufler & Wooton (2006).

Text text text text text text text (see, *inter alia*, Haaparanta 1996, pg. 10).

Special Czech, Slovak, and German letters:

ů, á, š, ď, ť, ř, ô, ß, ö

5.3 Formal requirements of master's thesis

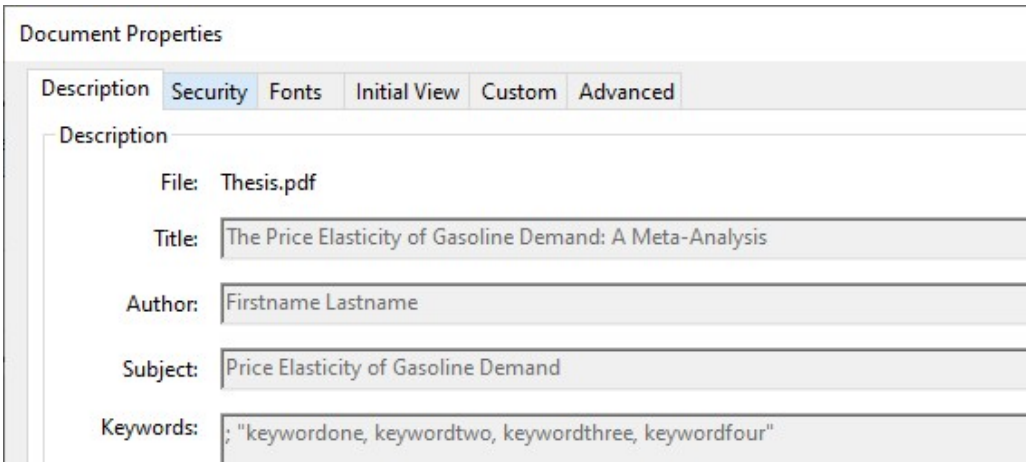
According to Dean's Provision no. 18/2017:

- The minimum extent of master's thesis is 60 standard pages (108 thousand characters including spaces) of the text itself, i.e. without an abstract and appendices and a list of literature. In case the master's thesis is written in English, its minimum extent is 50 standard pages (90 thousand characters including spaces) without an abstract and appendices and a list of literature. When writing a standard text document, the minimum requirement is 60 characters per line and 30 lines per page, i.e. 1,800 characters per page (the so-called standard page). Font size, page layout, margins, and line spacing need to be customized.
- Generally, a standard form of the page of the final thesis applies the fonts of 12 points, the gaps between the paragraphs are recommended to be of the size of 6 points. Notes and footnotes can be written in a 10-point font. The text is aligned on both sides (aligned to a block). Electronic version of the thesis will be entered by a student/applicant for a state examination through the SIS website interface in the archive format of PDF/A version 1.3 or higher. Further details are stipulated by the rector's provision.
- The master's thesis is submitted in the accreditation language of the respective follow-up Master's study program.

Note that due to GDPR, the thesis cannot include any personal information (phone, e-mail) or signatures (neither of the author nor of the supervisor).

5.4 Template adjustments and meta-data

Read README.txt to get a jist of how the template works and how to adjust styles. You can change the properties of your pdf file, such as title, author, keywords, or publisher



in **Thesis.xmpdata** file. The file is editable in any text editor.

5.5 Itemization and Environments

Many people use simple n-dash in many occasions – like this –, where however typographic convention—it looks a bit strange at first sight—requires m-dash. Text text text text text text Haufler & Wooton (2006).

Text text text text text text Wells *et al.* (2001). Let us describe the following animals:

Item 1 Text.

Item 2 Text.

See what Edmund Burke said about the duties of a Member of Parliament (Speech To The Electors Of Bristol At The Conclusion Of The Poll, November 3, 1774):

> It ought to be the happiness and glory of a representative to live in the strictest union, the closest correspondence, and the most unreserved communication with his constituents. Their wishes ought to have great weight with him; their opinion, high respect; their business, unremitted attention. It is his duty to sacrifice his repose, his pleasures, his satisfactions, to theirs; and above all, ever, and in all cases, to prefer their interest to his own. But his unbiased opinion, his mature judgment, his enlightened conscience, he ought not to sacrifice to you, to any man, or to any set of men living. These he does not derive from your pleasure; no, nor from the law and the constitution. They are a trust from Providence, for the abuse of which he is deeply answerable. Your representative owes you, not his industry only, but his judgment; and he betrays, instead of serving you, if he sacrifices it to your opinion.

Text text text text.

(i) The first item, the first item, the first item, the first item, the first item, the first item,

(ii) and the second item.

(a) The first item, the first item, the first item, the first item, the first item, the first item,

(b) and the second item.

TText text text text text text Blomstrom & Kokko (2003).

5.6 Acronyms

Politicians usualy like inward **FDI!** (**FDI!**) and an **MNC!** appreciates **FDI!** subsidies. Are **MNC!**s greedy?

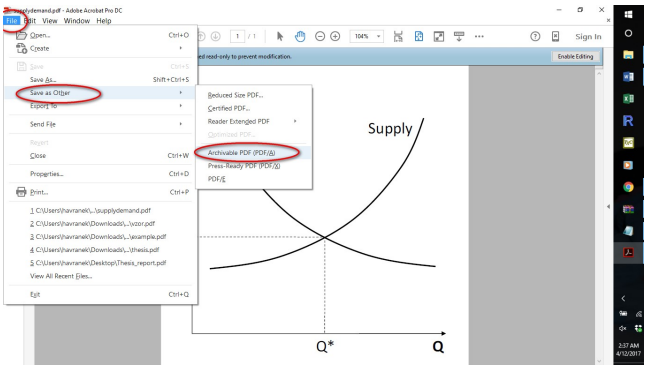
5.7 Figures

To achieve compatibility with PDF/A 2u, your file must not include links to external fonts, audio, video, or scripts. On the other hand, your file must declare each color environment you use, it must include all the pictures/figures either in jpeg or PDF/A 2u format, used fonts compliant under Unicode (your

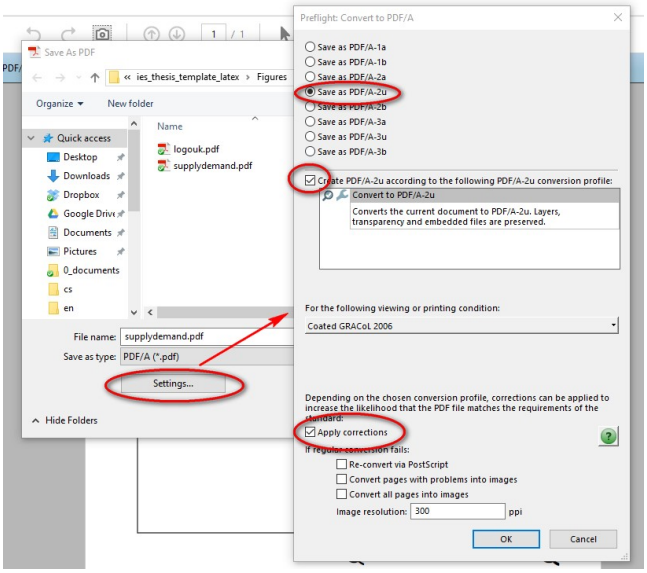
file cannot use any external fonts), and it must include meta-data in XMP format.

Most troubleshooting comes from the conversion of figures to compliant formats. You can convert from simple PDF using Adobe Acrobat:

- Select File » Save as Other » Archivable PDF (PDF/A)



- Save as PDF/A-2u:

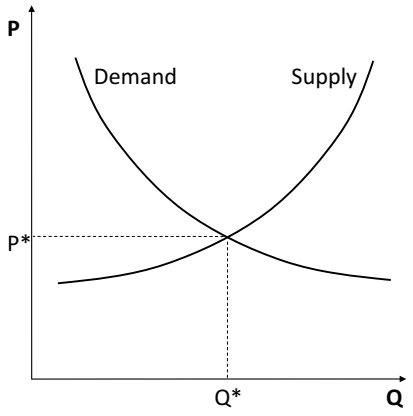


But most of the vector graphics gets distorted to lower quality in Adobe (like pictures in pdfs generated from Stata, unless jpeg is sufficient for you). You can also use GhostScript, the conversion tool is provided by courtesy of the Faculty of Mathematics and Physics at

https://kam.mff.cuni.cz/pdfix/

Text text text text text text.[1] Font of Latin phrases should be consistent: Furthermore, there is no *ex post* price effect, all things being equal (*ceteris paribus*). This is *per se* truth.

Figure 5.1: Market equilibrium



Source: Haufler & Wooton (2006).

Look at the Figure 5.1. Text text text text text text text text text text.

5.8 Tables

If you use Stata, you might want to check the `sutex`, `outtable`, `outtex`, and `estout` tools, which help you with exporting Stata tables to LaTeX.

Table 5.1: Model's predictions

Case	Y_1	Y_2	τ_1	τ_2	a	n
CR—Slovakia	10.9	10	0.24	0.19	1,000	2.16
CR—Poland	13.3	12	0.24	0.19	1,000	0.38
CR—Hungary	10.4	8	0.24	0.16	1,000	1.10

Source: If the source is author himself (like a calculation output), this line is redundant.

Text text text text text text text text text text text text text text text.

[1] Text text text text text text text text text text text text text text text. Text text text text text text text text text text.Text text text text text text.

5.9 Boxes

Text text text text text text text text text text text text text text text text. Let us make a box:

Figure 5.2: Boxy's example

- Welcome to Boxy paragraph. We sincerely hope you will all enjoy the show.
- Welcome to Boxy paragraph. We sincerely hope you will all enjoy the show.
- Welcome to Boxy paragraph. We sincerely hope you will all enjoy the show.

Source: Haaparanta (1996)

Text text text text text text text text text text text text text text text text.

5.10 Theorems, Definitions, ...

Definition 5.1 (My original definition). This is a definition.

Assumption 5.1 (My realistic assumption). This is an assumption.

Proposition 5.1 (My clever proposition). *This is a proposition.*

Lemma 5.1 (My useful lemma). *This is a lemma.*

Example 5.1. This is an example.

Proof. This is a proof. □

5.11 Nonumbered Equations

$$U = \underbrace{\int_0^\infty \frac{1}{1-\sigma}\left(C^{1-\sigma}-1\right)e^{-\rho t}\,\mathrm{d}t}_{\text{meaning of life}}$$

5.12 Numbered Equations

$$U = \int_0^\infty \overbrace{\frac{1}{1-\sigma}\left(C^{1-\sigma}-1\right)}^{\text{instantaneous utility}} e^{-\rho t}\,\mathrm{d}t \tag{5.1}$$

5.13 Matrix Equations

$$\boldsymbol{A} = \boldsymbol{B} + \boldsymbol{C} \tag{5.2}$$

5.14 Cross-references

- to literature (Bjorvatn & Eckel 2006, pg. 10) or Haufler & Wooton (2006, pg. 10),
- to Figure 5.1,
- see Table 5.1,
- to **??**,
- to Definition 5.1, to Proposition 5.1, Example 5.1,
- to equations like this: see (5.1).

5.15 Source codes

You can input a source code like this:

```
omega = 1;
syms zeta;
jmn = [1 2*zeta*omega omega^2];
figure(1);
    for zeta = 1E-5 : 0.2 : 1+1E-12
        G = tf(omega^2,subs([1 2*zeta*omega omega^2]));
        bode(G); hold on;
    end
legend('\zeta = 0','\zeta = 0,2','\zeta = 0,4','\zeta = 0,6',');
```

Should you prefer a different font size, redefine file `Styles/Mystyle.sty`.

5.16 Paragraphs

Usually you should not use the first person singular (I) in your text, write we instead. As a general recommendation, use the first person sparsely, sometimes it can be replaced by a phrase like “This work presents . . . .”

Text text text text text text (Haufler & Wooton 2006). Let us make two paragraphs:

Proin Text text text text text text text text text text text text text text text text. Text text text text text text. And a subparagraph:

Velit Text text text text text text text text text text text text text text text text.

Chapter 6

Conclusion

The conclusion should briefly summarize the problem statement and the general content of the work and the emphasize on the main contribution of the work.

When writing the conclusion keep in mind that some readers may not have gone through the whole thesis, but have jumped directly to the conclusion after having read the abstract in order the decide on the personal relevance of the thesis. Therefore, the conclusion should be self contained, which means that a reader should be able to understand the essence of the conclusion without having to read the whole thesis.

The conclusion typically ends with an outlook that describes possible extensions of the presented approaches and of planned future work.

Bibliography

Bjorvatn, K. & C. Eckel (2006): "Policy Competition for Foreign Direct Investment Between Asymmetric Countries." *European Economic Review* **50(7)**: pp. 1891–1907.

Blomstrom, M. & A. Kokko (2003): "The Economics of Foreign Direct Investment Incentives." *NBER Working Papers 9489*, National Bureau of Economic Research, Inc.

Haaparanta, P. (1996): "Competition for Foreign Direct Investment." *Journal of Public Economics* **63(1)**: pp. 141–53.

Haufler, A. & I. Wooton (2006): "The Effects of Regional Tax and Subsidy Coordination on Foreign Direct Investment." *European Economic Review* **50(2)**: pp. 285–305.

He, H., Y. Bai, E. A. Garcia, & S. Li (2008): "Adasyn: Adaptive synthetic sampling approach for imbalanced learning." In "2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)," pp. 1322–1328.

Igareta, A. (2021): "Stratified sampling: You may have been splitting your dataset all wrong."

Kornbrot, D. (2014): "Point biserial correlation." *Wiley StatsRef: Statistics Reference Online* .

Ribeiro, M. T., S. Singh, & C. Guestrin (2016): """ why should i trust you?" explaining the predictions of any classifier." In "Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining," pp. 1135–1144.

Rigatti, S. J. (2017): "Random forest." *Journal of Insurance Medicine* **47(1)**: pp. 31–39.

Wells, L. T., N. Allen, J. Morisset, & N. Pirnia (2001): *Using Tax Incentives to Compete for Foreign Investment: Are They Worth the Cost?* Washington, DC: FIAS.

Wendler, T. & S. Gröttrup (2021): *Data Mining with SPSS Modeler: Theory, Exercises and Solutions.* Cham: Springer.

Appendix A

Title of Appendix A

Text text text text text text text text text text text text text text text. Text text text text text text text text text text. Text text text text text text text text text text text text text text text text text text.

Appendix B

Project's website

You can create a special website for your project which contains empirical data and MatLab/R/Stata source codes, see meta-analysis.cz/sigma, for example. Stating in your thesis that the data and source codes are available upon request is enough but please, have them prepared for such requests. The faculty does not allow enclosed DVD.

- File 1: Master's thesis
- File 2: Empirical data
- File 3: Source codes