



Credit Card Approval Prediction

ABW508D MAKMAL ANALITIK (ANALYTICS LAB)

Name Of Student: Nahla Ahmed Ibrahim Ahmed

Matric No: PEM0211/23

Email: Nahla.Ahmed@Student.usm.my

Supervisor: Professor Dr. Teh Sin Yin

Semester 1
Academic Session 2023/2024
School of Management
University of Sains Malaysia

Abstract

In response to the limitations of traditional credit scoring systems, a modern approach utilizing machine learning techniques has been employed to revolutionize credit evaluation procedures. By analyzing historical customer data with algorithms such as XGBoost, Light Gradient Boosting Machine (LGBM), Decision Trees (DT), Random Forests (RF), Support Vector Machines (SVM), and Logistic Regression (LR), significant patterns influencing credit card approval decisions have been identified. Advanced techniques including Synthetic Minority Over-sampling Technique (SMOTE) for data imbalances, Weight of Evidence (WoE) for strategic feature selection, and vintage analysis for long-term credit performance assessment were utilized. Following extensive testing, XGBoost emerged as the most efficient algorithm, achieving an impressive accuracy score of 92.8% and an F1 score of 92.9%. This research breakthrough equips financial institutions with a powerful tool to minimize risks and enhance customer satisfaction in credit evaluations, aligning with the broader trend of leveraging machine learning to address credit risk assessment challenges.

Keywords: including XGBoost, LGBM, Decision Trees, Random Forests, Support Vector Machines, Logistic Regression, SMOTE, Weight of Evidence (WoE), and vintage analysis.

Table of Contents

Abstract.....	2
Table of Contents	3
Table of Tables	5
Table of Equation	5
Table of Figure.....	5
Acknowledgement	6
Chapter 1: Prediction of Credit Card Approval	7
1.1. Introduction	7
1.2. Background of the Study	7
1.3. Credit Card Risk	8
1.4. Problem Statement.....	9
1.5. Research Objectives	10
1.6. Research Questions	10
1.7. Significance of the Study	10
1.8. Organization of the Remaining Chapters	11
Chapter 2: Literature Review	12
2.1. Introduction	12
2.2. Credit Approval Prediction	12
2.3. Supervised Learning for Binary Classifications.....	12
2.3.1. Logistics Regression	13
2.3.2. Decision Tree Classifier	14
2.3.3. Random Forest Classifier	15
2.3.4. XGBoost	15
2.3.5. LightGBM (Light Gradient Boosting Machine).....	16
2.4. Summary of Chapter 2	19
Chapter 3 Methodology	20
3.1. Introduction	20
3.2. Specific Tools	20
3.3. Flow Chart.....	20
3.4. Data Retrieval.....	22
3.5. Vintage analysis.....	23
3.6. Data Pre-processing	24

.3.6.1	Merging.....	24
.3.6.2	Deleting Unnecessary Columns.....	24
.3.6.3	Handling Missing Values	25
3.6.4.	Handling outliers.....	25
3.7.	Exploratory Data Analysis	25
3.7.1.	Descriptive Statistics	26
3.7.2.	Heatmap Correlation Matrix	26
3.8.	Encoding Categorical Variables.....	26
3.9.	Weight of Evidence	26
3.10.	Feature Scaling.....	27
3.11.	Synthetic Minority Over-sampling Technique (SMOTE)	27
3.12.	Split Data	28
3.13.	Feature Selection	29
3.13.1.	Recursive Feature Elimination (RFE).....	29
3.14.	Data Modelling.....	30
3.15.	Performance Metrics	30
CHAPTER 4	RESULT AND DISCUSSION.....	34
4.1.	Introduction.....	34
4.2.	Vintage Analysis	34
4.3.	Result of Exploratory Data Analysis	35
4.4.	Weight of Evidence (WOE):.....	38
4.5.	Machine Learning Performance.....	39
4.6.	Comparison of the Confusion Matrices	40
4.7.	Summary of Chapter 4	47
CHAPTER 5	CONCLUSION	48
5.1.	Summary and Recommendations.....	48
5.2.	Limitation	49
REFERENCES	51
APPENDIX: Python Programming in Algorithm Building	57

Table of Tables

Table 1: Summary of journal articles using different types of machine learning algorithms	17
Table 2: First Table of Data.....	22
Table 3: Second Table of Data	23
Table 4: Relationship between IV value and predictive power:.....	27
Table 5: Size of training and test dataset	28
Table 6: Features and the target variable	29
Table 7: Confusion Matrix	30
Table 8 : Descriptive Statistics.....	36
Table 9: The confusion matrix of logistic regression.....	40
Table 10: The confusion matrix of Decision Tree Classifier.....	41
Table 11: The confusion matrix of Decision Tree Classifier.....	42
Table 12: The confusion matrix of SVM	43
Table 13: The confusion matrix of LGBM Classifier.....	44
Table 14: The result of accuracy score, precision, recall and F1 score for the model	46

Table of Equation

Equation 1: Weight of Evidence.....	27
Equation 2: Information Value Information Value	27
Equation 3: Equation of Accuracy.....	31
Equation 4: Equation of Recall.....	32
Equation 5: Equation Precision	32
Equation 6: Equation of F1-score.....	32

Table of Figure

Figure 1: Flow chart for prediction of Customer Credit Card	21
Figure 2: Credit Card Dashboard.....	33
Figure 3: cumulative percentage of bad customers.....	34
Figure 4: Detect Missing Value.....	35
Figure 5: Detect outliers	37
Figure 6: Correlation Matrix.....	38
Figure 7: IV Table Visualization with percentages.....	39
Figure 8: The confusion matrix of logistic regression	41
Figure 9: The confusion matrix of Decision Tree Classifier	42
Figure 10: The confusion matrix of Decision Tree Classifier	43
Figure 11: The confusion matrix of SVM.....	44
Figure 12: The confusion matrix of LGBM Classifier	45

Acknowledgement

I would like to express my deepest gratitude to my supervisor, Dr. Teh Sin Yin, for her unwavering support, invaluable guidance, and continuous encouragement throughout the completion of my study. Her expertise, insightful feedback, and dedication have been instrumental in shaping the quality and direction of my research.

I am profoundly thankful to my parents for their unwavering support and encouragement. Their love, belief in me, and sacrifices have been a constant source of motivation, enabling me to pursue my academic endeavors.

I also extend my sincere appreciation to my friends for their understanding, encouragement, and unwavering support during this journey. Their presence and words of encouragement have been a source of motivation during challenging times.

These individuals have played a significant role in my academic journey, and I am truly grateful for their support, guidance, and belief in my abilities.

Chapter 1: Prediction of Credit Card Approval

1.1.Introduction

Credit cards have become an increasingly popular method of electronic payment worldwide because they offer several advantages to both buyers and vendors, such as the elimination of cash and checks, the provision of reliable transaction records, and the establishment of credit history (Aliapoulios et al., 2021). In China, credit cards have rapid development in recent years, driven by rising incomes and growing domestic demand, and have had several positive impacts on the Chinese economy, they offer several advantages, such as convenience, and rewards programs (Wu et al., 2021). The number of credit cards in India has doubled from 24.4 million in 2015-16 to 48.9 million in 2018-19. Transaction volume has tripled from Rs.2.4 trillion to Rs.6.07 trillion, and the number of transactions has doubled from 0.8 billion to 1.7 billion

Credit scoring aims to classify applicants as either good or bad credit risks. Even a 1% improvement in the accuracy of credit scoring for bad credit applicants can significantly reduce financial institutions' losses, improved credit scoring accuracy can also help financial institutions to expand their lending activities. By being able to identify good credit applicants more accurately, financial institutions can lend to more borrowers with confidence. This can help to boost economic growth by increasing consumer spending and investment (Kozodoi et al., 2022).

The combination of machine learning and optimization algorithms has emerged as a modern approach for both qualitative and quantitative analysis of large datasets. Predictive analysis utilizes these algorithms to process information and predict future trends. As a result, Predictive analytics can help lenders identify credit card customers at high risk of default, enabling them to make informed decisions earlier and minimize financial losses.

In this chapter, a background of the study, problem statement, research objectives, and research questions will be explained based on the background. This chapter will also explain the significance of this study.

1.2.Background of the Study

A credit card is a plastic card issued by a financial institution that grants the cardholder a line of credit such as a bank, that allows individuals to make purchases at merchants, (Esenogho et al.,

2022). This line of credit represents the maximum amount of money that the cardholder can borrow from a financial institution. The cardholder is then required to repay the borrowed amount, either in full or in monthly installments, within a specified timeframe. avoid accumulating debt and incurring high-interest charges. By using credit cards wisely and making timely payments, individuals can build a positive credit history, which can be beneficial for future borrowing opportunities, such as loans or mortgages.

Machine learning algorithms are powerful tools for detecting patterns in large, unstructured, and complex datasets. These algorithms can be trained to identify relationships between data points that are not immediately apparent to humans, even when the data is noisy or incomplete. This makes machine learning ideal for a wide range of applications, including fraud detection, customer segmentation, and medical diagnosis (Pudjihartono et al., 2022). One of the key advantages of machine learning for pattern detection is its ability to learn from data without human intervention. This means that machine learning algorithms can be used to identify patterns in data that are too large or complex for humans to analyze manually. Additionally, machine learning algorithms can be continuously updated with new data, which allows them to learn and adapt over time. Machine learning can be used to detect credit card risk by analyzing large amounts of historical data on credit card applications and approvals. Machine learning algorithms can identify patterns in this data that can be used to predict whether an applicant is likely to be approved for a credit card (Tanikella, 2020). Supervised learning algorithms are a type of machine which is trained on a dataset of labeled data where the labels indicate whether an applicant was approved or denied for a credit card. The algorithm learns to identify the patterns in the A range of machine Supervised learning algorithms, including Random Forest, Decision Tree, Logistic Regression, Support Vector Machines (SVM), XGBoost (extreme Gradient Boosting), and LightGBM (Light Gradient Boosting Machine) can be employed to predict credit card risk.

1.3. Credit Card Risk

Credit risk evaluation decisions are crucial for financial institutions, as the consequences of incorrect decisions can be severe. Predicting credit failure accurately is essential for making sound financial decisions. Financial crises are often caused by poor decision-making in financial institutions.

Accurate credit risk prediction is essential for financial institutions to make sound lending decisions and avoid financial crises. Incorrect predictions can lead to significant financial losses, as well as loss of public confidence in the financial system. Therefore, it is in the best interests of both financial institutions and the economy as a whole to develop and implement robust and reliable credit risk prediction models (Weng & Huang, 2021).

1.4.Problem Statement

Credit card approval is a particularly important decision for banks, as credit cards are a form of unsecured debt. This means that the bank does not have any collateral to seize in the event of a default. As a result, banks have a strong incentive to avoid issuing credit cards to borrowers who are likely not to repay the debt they owe on their credit cards. Bad choices lead to decreased profits, increased costs, and even financial instability for the bank institution.

The rapid development of China's Internet finance industry has led to a significant increase in financial risks, especially credit risk. Traditional credit card scoring models are widely used to evaluate credit risk but they have several shortcomings, traditional credit card scoring models are often complex and difficult to interpret, which can make it difficult for Internet finance platforms to understand the factors that are driving their credit risk decisions, and this can lead to an increase in financial risks (Fan et al., 2020). Nearly half of all consumers (44%) have unpaid credit card debt, with an average balance of \$6,597. This is a major financial burden for many people, and it can have a significant impact on their credit scores (Stavins, 2020). The rising number of new credit applications and the significant accumulation of unpaid credit card bills during the recent pandemic have exacerbated the challenge of managing personal finances effectively (Khan et al., 2020). Credit card companies face a significant challenge in identifying and assessing the risk of customers who are likely to default on their installment payments. This is because there is a lack of reliable and accurate predictors of installment default risk. As a result, credit card companies often rely on traditional credit scoring models, which are not specifically designed to predict installment default risk (Dobbie & Song, 2020).

Thus, banks and other financial institutions can reduce the risk of credit cards. This can help to protect the financial institution's bottom line and protect consumers from financial hardship, so Credit card companies need accurate prediction systems to assess the risk of approving or denying applications. A critical task is to predict which applicants are likely to default on their payments.

By using predictive analysis with machine learning (ML), credit card companies can identify potential defaulters with greater accuracy.

1.5.Research Objectives

The objectives of this study are shown as below:

RO1. To determine the most important features that influence credit card approval decisions.

RO2. To compare the performance of different machine learning algorithms for predicting credit card approval.

RO3.To identify the champions machine learning model that can accurately predict whether a credit card application will be approved or rejected.

1.6.Research Questions

RQ1.What are the most important features that influence credit card approval decisions?

RQ2.What are the performance metrics that can be used to assess the performance of the algorithms?

RQ3.Which machine learning model is optimal for accurately predicting whether a credit card application will be approved or rejected?

1.7.Significance of the Study

Machine learning within the rating classification prediction model, the credit history of current borrowers, which can be harnessed for future loan request comparisons. This approach detects the characteristics of good and bad loan histories. By leveraging credit history data and a demographic profile, the algorithm employed data mining techniques to pinpoint and help financial risks associated with financial fraud. The findings reveal the algorithm's exceptional efficiency in identifying credible bank customers. The algorithm outperformed other comparable methods.

This study provides new insights into the use of credit card data to predict customer behavior. The study's findings suggest that using the meaning of a customer's credit score to make classification predictions about their creditworthiness (Bad or Good) that does not exist in the data but using vintage analysis, binary predictions is a more effective approach than using individual credit scores. This finding could have important implications for the development of new credit scoring models.

1.8.Organization of the Remaining Chapters

The order of this report is as follows:

- Chapter 1: This chapter covers the introduction to the problem, and highlights the ML algorithm used, its purpose, objectives, and the scope of this study.
- Chapter 2: This chapter explains the methodological framework of the study from data retrieval until model evaluation.
- Chapter 3: This chapter reports on the results of the experiment performed on the selected algorithms.
- Chapter 4: This chapter is about the limitations and the discussion of the study.
- Chapter 5: This chapter contains recommendations regarding future work and concludes the report

Chapter 2: Literature Review

2.1. Introduction

This chapter delves into the existing literature pertinent to the present study. The first section meticulously examines the machine learning algorithms employed in prior studies and their associated outcomes. Additionally, a comprehensive review of various machine learning algorithms is undertaken in the second section.

2.2. Credit Approval Prediction

Credit approval prediction involves using historical data and machine learning algorithms to anticipate whether a loan application will be approved or denied. This process is crucial for various financial products, including credit cards, loans, and mortgages. Lenders evaluate a borrower's creditworthiness by assessing factors such as credit history, financial history, credit score, income, and assets to determine the likelihood of the borrower repaying the credit as agreed. Machine learning algorithms play a significant role in predicting loan approval by analyzing historical data and generating informed decisions. This process demonstrates the importance of leveraging technology to make accurate predictions in the financial industry (Z. Chen et al., 2021).

2.3. Supervised Learning for Binary Classifications

Credit card approval prediction can be considered a binary classification problem, as the goal is to classify applicants into one of two categories (Good or bad). There is no single model that consistently outperforms all others. Therefore, various techniques are evaluated to determine the most suitable algorithm for the specific dataset used in this study. Six supervised learning algorithms are chosen due to their simplicity and popularity in binary classification problems are Decision Tree Classifier, Support Vector Machine (SVM), Logistic Regression, Light Gradient Boosting Machine (LGBM), Extreme Gradient Boosting (XGBoost)and, Random Forest

The performance of these six algorithms will be evaluated on a benchmark dataset of credit card applications to determine the most suitable model for credit card approval prediction.

Support vector machines (SVMs) are supervised machine learning algorithms used for classification and regression tasks. They are particularly well-suited for classification tasks, where they can effectively distinguish between different classes of data points. SVMs are based on the

idea of finding a hyperplane that maximizes the margin between two classes of data. The hyperplane is a decision boundary that separates the data points into two classes. The margin is the distance between the hyperplane and the nearest data points from each class (Land Jr et al., 2020)

SVMs address this challenge by selecting two parallel hyperplanes that maximally separate the two classes. The distance between these two hyperplanes, known as the margin, is maximized. These two hyperplanes are referred to as the margin-maximizing hyperplanes. The data points lying on the margin-maximizing hyperplanes are called support vectors. SVMs utilize these support vectors to determine a decision surface between the margin-maximizing hyperplanes. This decision surface is then used for classification (Paoletti et al., 2020). In the presence of outliers and noise, there may be an overlap between data points from both classes. To overcome this challenge, soft-margin SVMs relax the condition on the margin-maximizing hyperplanes. This allows some data points to fall within the margins. A non-negative hyperparameter C is used by soft-margin SVMs to control the amount of relaxation. A value of $C=0$ corresponds to a hard-margin SVM, where no data points breach the margin-maximizing hyperplanes. As the value of C increases, the SVM becomes more relaxed, making it less sensitive to noise in the training data. However, this also increases bias and reduces the variance of the model (Battineni et al., 2020).

2.3.1. Logistics Regression

Logistic regression is a statistical method used to predict the probability of a binary outcome (e.g., yes or no, alive or dead) based on one or more predictor variables. LR is a statistical method used to model the relationship between a dependent variable and one or more independent variables, it is a type of generalized linear model that uses the logistic function to transform the linear combination of predictor variables into a probability between 0 and 1 (Dumitrescu et al., 2022).

Logistic regression is a valuable tool for researchers who are interested in understanding and predicting quality of life outcomes. It is a relatively simple method to understand and use, and it can be applied to a wide range of research questions. There are several extensions to logistic regression, such as multinomial logistic regression and ordinal logistic regression, that can be used to model more complex relationships between the predictor variables and the outcome (A. Das, 2021).

2.3.2. Decision Tree Classifier

Decision tree classifiers, emphasizing their simplicity, interpretability, and computational efficiency. He detailed the ID3 algorithm, the cornerstone of Decision tree classifiers learning, and its extensions, including C4.5, which handles missing values and continuous attributes effectively. Addressing overfitting, explored ensemble methods like bagging and boosting to enhance Decision tree classifiers accuracy. Highlighting Decision tree classifier's wide applicability, including medical diagnosis, spam filtering, and credit risk assessment (Priyanka & Kumar, 2020). Decision trees are a type of machine learning algorithm that is commonly used for classification and regression tasks. DTs can be classified into three main categories: binary decision trees, multi-way decision trees, and regression trees. Binary decision trees split the data into two branches at each internal node, leading to binary classification. Multi-way decision trees split the data into multiple branches at each internal node, allowing for multi-class classification. Regression trees predict continuous numerical values rather than discrete classes (Charbuty & Abdulazeez, 2021).

Decision trees as a powerful tool for data analysis and modeling. Decision trees are easy to interpret, can model complex relationships between variables, and are relatively robust to noise and missing data. However, they can be prone to overfitting, computationally expensive to train, and may not be suitable for high-dimensional data. Overall, decision trees are a versatile tool with advantages and disadvantages that should be considered when choosing a modeling technique (C. Wang et al., 2020). Some common decision tree algorithms include C4.5, ID3, and CART. Both ID3 and C4.5 utilize information entropy to select features as decision nodes. C4.5, however, introduced enhancements to ID3 in terms of tree pruning and handling missing data points, effectively preventing overfitting issues. CART, on the other hand, employs the Gini diversity index. Both Gini index and entropy are commonly used measures to approximate the quality of a feature and its resulting partitions (Elmachtoub et al., 2020). Decision tree classifiers are a powerful tool for big data classification, they are easy to understand and interpret, making them well-suited for situations where understanding the classification process is crucial (Hartmann, 2021).

2.3.3. Random Forest Classifier

Random forests are an ensemble learning method that combines multiple decision trees to make predictions, where each tree is constructed using a different random subset of the training data. Each tree in the forest makes a prediction, and the final prediction of the forest is the average of the predictions of all the trees. They are popular in machine learning because they can generate highly accurate predictions, even when dealing with complex datasets. This makes them a powerhouse for diverse real-world applications. Additionally, they are relatively robust to outliers and noise in the data, can be interpreted to some extent, and can be efficiently trained on large datasets (S. Das et al., 2023).

The randomness in the construction of the trees helps to prevent the forest from overfitting to the training data. Overfitting is a common problem in machine learning, where the model learns the training data too well and is unable to generalize to new data. Random forests are less prone to overfitting because the trees in the forest are not trained on the entire dataset, and they are also trained on a random subset of the features (Schonlau & Zou, 2020). Random forests are a versatile machine learning method that can be used for a variety of tasks, including classification, regression, and unsupervised learning. They are a popular choice for machine learning applications because of their high accuracy, robustness, and ease of implementation (Habib et al., 2020).

2.3.4. XGBoost

XGBoost (Extreme Gradient Boosting) is a decision-tree-based ensemble machine learning algorithm that uses a gradient boosting framework. It is an optimized distribution machine learning library designed for efficient and scalable training of gradient boosting models. It is known for its high performance and accuracy, and it has been widely used in a variety of machine learning competitions. It is particularly well-suited for classification and regression tasks, and it can handle missing values and outliers well. However, XGBoost can be sensitive to noise and requires careful parameter tuning (Osman et al., 2021). Imbalance-XGBoost is an effective method for addressing binary label imbalance in XGBoost classification. It is a versatile method that can be applied to a wide range of classification tasks. Weighted and focal losses are effective techniques for improving the performance of XGBoost on imbalanced datasets. This is because it is an ensemble method that relies on a series of decision trees to make predictions. Decision trees are inherently biased

towards the majority class, as they tend to split the data in a way that maximizes the separation between the two classes (Kavzoglu & Teke, 2022). boosting approach where the loss function of each iteration is optimized along the gradient direction to build a weak classifier function. The outputs of multiple weak classifiers are then combined with specific weights to form a strong classifier, resulting in the final prediction output. Extreme gradient boosting (XGBoost) enhances the gradient boosting algorithm by expanding the loss function using Taylor's second-order series expansion, facilitating faster model convergence. Additionally, a regularization term is incorporated into the loss function to prevent overfitting (T. Wang et al., 2023).

2.3.5. LightGBM (Light Gradient Boosting Machine)

LGBM (Light Gradient Boosting Machine) is a gradient boosting framework that utilizes decision trees as its base learners. It is designed to be efficient, scalable, and distributed, making it suitable for large-scale machine learning tasks, and has been shown to outperform other gradient boosting algorithms in terms of accuracy and training speed. It is a free and open-source framework that is known for its efficiency and speed. LGBM is used for a variety of machine learning tasks, including classification, regression, and ranking (Ahamed, 2021).

LGBM is a type of ensemble learning algorithm that combines multiple decision trees to make predictions. Decision trees are simple machine learning models that make predictions by splitting data into smaller and smaller subsets based on certain features. LGBM uses a technique called gradient boosting to improve the accuracy of decision trees. Gradient boosting works by iteratively adding new decision trees to the ensemble, each of which focuses on correcting the errors made by the previous trees. LGBM is known for its efficiency and speed because it uses a number of optimizations to the traditional gradient boosting algorithm. These optimizations including Histogram-based learning LGBM uses histograms to represent the data, which makes it more efficient to compute the gradients and updates for the decision trees. Early stopping: LGBM uses early stopping to prevent overfitting, which is when a model learns too much from the training data and is unable to generalize to new data. Parallel training: LGBM can be trained on multiple GPUs or CPUs, which makes it even faster (Y. Wang et al., 2023)

Table 1: Summary of journal articles using different types of machine learning algorithms

No	Article Title	Authors (Year)	Algorithm					
			Random Forest	Decision Tree	Logistic Regression	Support Vector Machine	LGBM	XGBoost
1	Credit card fraud detection using machine learning algorithms	(Varun Kumar et al., 2020)		√	√	√		
2	A Comparative Assessment of Credit Risk Model Based on Machine learning	(Y. Wang et al., 2020)	√	√	√			
3	Analysis and comparison of credit card fraud detection using machine learning	(Rout, 2021)	√		√			√
4	Predicting Credit Risk for Unsecured Lending: A Machine Learning Approach	(Naik, 2021)	√	√	√		√	√
5	A machine learning based credit card fraud detection using the GA algorithm for feature selection	(Ileberi et al., 2022)		√	√	√		

6	Enhanced Credit Card Fraud Detection Model Using Machine Learning	(Alfaiz & Fati, 2022)		✓	✓		✓	✓
7	Unbalanced Credit Card Fraud Detection Data: A Machine Learning-Oriented Comparative Study of Balancing Techniques.	(Gupta et al., 2023)	✓		✓	✓		✓
8	Towards the Improvement of Credit Card Approval Process Using Classification Algorithm.	(Del Pilar & Bongo, 2023)	✓	✓	✓	✓		
9	Assessing the feasibility of machine learning-based modelling and prediction of credit fraud outcomes using hyperparameter tuning	(Tang, 2023)	✓		✓	✓	✓	✓
10	Improve Fidelity and Utility of Synthetic Credit Card Transaction Time Series from Data-centric Perspective	(Hsieh et al., 2024)	✓	✓			✓	✓
Total			7	7	9	5	4	6

2.4. Summary of Chapter 2

The literature review provides a comprehensive analysis of the application of machine learning algorithms for credit approval prediction, emphasizing the importance of leveraging technology to accurately assess a borrower's creditworthiness. Six supervised learning algorithms, including Decision Tree Classifier, Support Vector Machine (SVM), Logistic Regression, Light Gradient Boosting Machine (LGBM), Extreme Gradient Boosting (XGBoost), and Random Forest, are evaluated for their suitability in binary classification problems, specifically credit card approval prediction.

Chapter 3 Methodology

3.1. Introduction

This study outlines the steps involved in determining the champion model for predicting bank customer credit card application acceptance. The process encompasses six main stages: data preprocessing, exploratory data analysis, feature engineering, model building, model evaluation, and model selection. Each stage plays a crucial role in ensuring the accuracy and reliability of the predictive model. The complete implementation of these steps is provided in Appendix A.

3.2. Specific Tools

The study utilizes Python programming language within the Jupiter Notebook for data analysis and visualization, including the important step of performing permutation Recursive Feature Elimination (RFE). Additionally, Power BI Desktop is employed as a business intelligence software for further data analysis and visualization. The laptop used for the study features a 64-bit operating system, an Intel(R) Core (TM) i5-6200U CPU @ 2.30GHz processor, and 8GB of RAM, providing a suitable environment for conducting the study's analytical tasks.

3.3. Flow Chart

"Flowchart and Algorithm Basics: The Art of Programming," emphasizes the importance of flowcharts as a fundamental tool for understanding and developing computer programs. Throughout the book, flowcharts are presented as a visual way to represent the logic and sequence of steps involved in solving a problem through programming (Chaudhuri, 2020). Building an accurate customer credit card prediction model requires following a specific sequence of steps, as visually depicted in the flowchart in Figure 3.1, to ensure clarity and precision throughout the process.

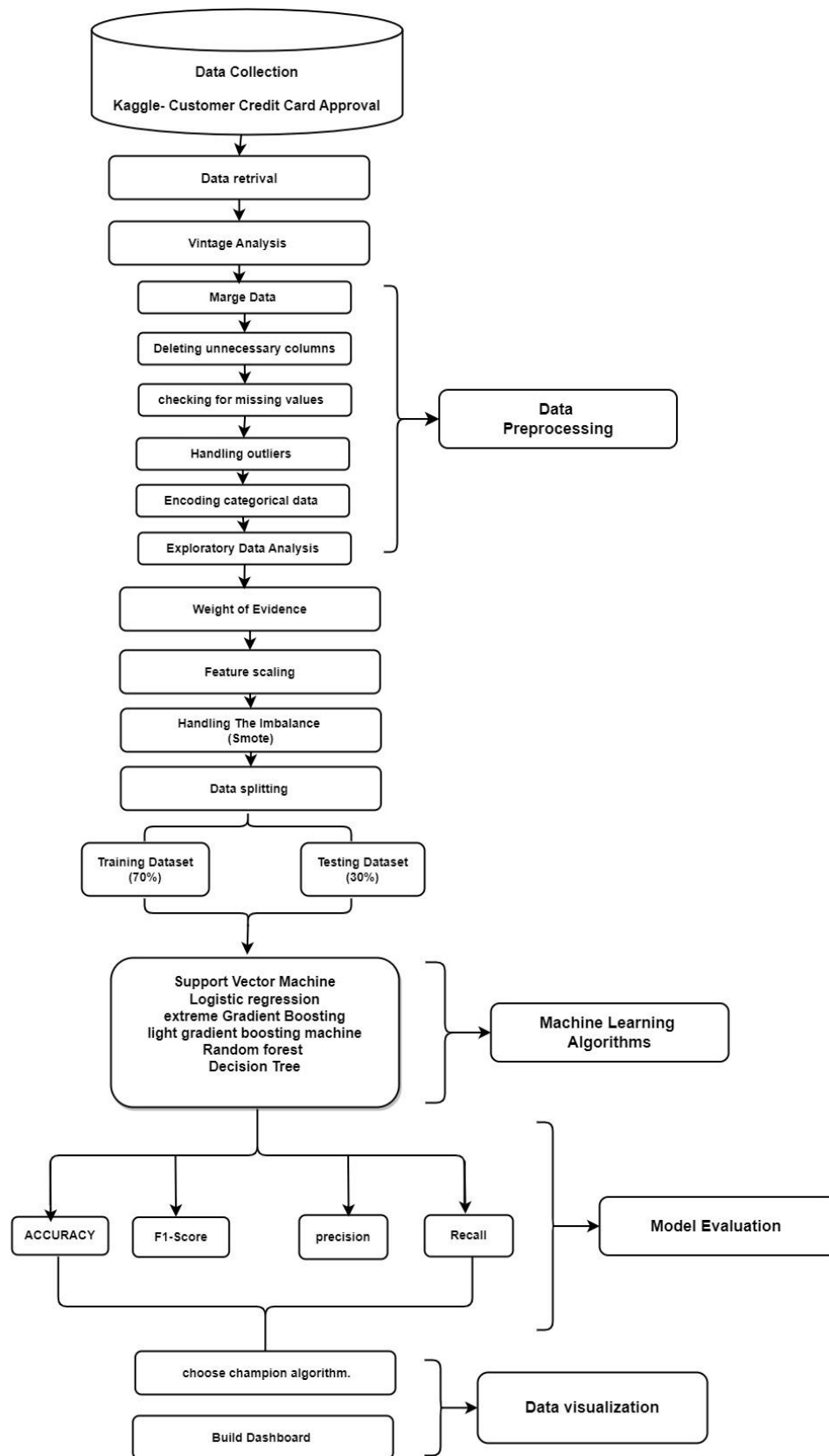


Figure 1: Flow chart for prediction of Customer Credit Card

3.4. Data Retrieval

The credit card approval data used in this study is publicly available on the Kaggle website which is available on Kaggle through the following link: <https://www.kaggle.com/datasets/rikdifos/credit-card-approval-prediction>. It comprises historical data from a bank related to customer credit card applications. The data is provided in a comma-separated values (CSV) file format and was imported into Python using the Pandas library's `read_csv()` function, which converts CSV files into Data Frames with labeled axes. The dataset consists of two tables: the first table contains 18 features and 438,558 observations, while the second table contains 3 features and 1,048,576 observations. Detailed descriptions of each feature are provided in the accompanying documentation.

Table 2: First Table of Data

Feature Name	Description
ID	A unique ID that identifies each customer
CODE_GENDER	The customer's gender (Male/Female)
FLAG_OWN_CAR	Whether the customer has a car or not (Yes/No)
FLAG_OWN_REAL	Whether the customer has a property or not (Yes/No)
CNT_CHILDREN	Number of children the customer has
AMT_INCOME_TOTAL	Annual income for each customer
NAME_INCOME_TYPE	The category of each customer's income
NAME_EDUCATION_TYPE	Education level of each customer
NAME_FAMILY_STATUS	Marital status of each customer
NAME_HOUSING_TYPE	The type of house the customer lives in
DAYS_BRITH	The customer's age in days, counted backwards from the current day (0), -1 means yesterday
DAYS_EMPLOYED	The customer's work experience in days, counted backwards from the current day (0). If positive, it means the person is currently unemployed
FLAG_MOBIL	Whether the customer has a mobile phone or not (Yes/No)

FLAG_WORK_PHONE	Whether the customer has a work phone or not
FLAG_EMAIL	Whether the customer has an email or not (Yes/No)
OCCUPATION_TYPE	The name of the customer's occupation
CNT_FAM_MEMBERS	The customer's family size

Table 3: *Second Table of Data*

Feature Name	Description
ID	A unique ID that identifies each customer
MONTHS_BALANCE	The month of the extracted data is the starting point, backwards, 0 is the current month, -1 is the previous month, and so on.
STATUS	<ul style="list-style-type: none"> - 0: 1-29 days past due - 1: 30-59 days past due - 2: 60-89 days overdue - 3: 90-119 days overdue - 4: 120-149 days overdue - 5: Overdue or bad debts, write-offs for more than 150 days - C: paid off that month - X: No loan for the month

3.5. Vintage analysis

Vintage analysis, also known as cohort analysis, is a method used in credit risk management to measure the performance of a portfolio over different periods of time after the loan or credit card was granted. It involves analyzing the cumulative charge-off rate, proportion of customers past due, utilization ratio, average balance, and other performance metrics over time. Vintage analysis allows for trend analysis and provides insights into the credit quality of a loan portfolio by analyzing net charge-offs in a given loan pool where the loans share the same origination period. This method is widely used in the analysis of retail credit card and mortgage portfolios and is essential for understanding the risk within the risk and gaining insights into allowance for credit losses. Vintage analysis provides valuable information for managing credit risk and making informed decisions in the financial and banking sectors (H. Wang et al., 2024).

3.6. Data Pre-processing

Data preprocessing is a vital process that plays a key role in enhancing the accuracy and performance of analytical models by addressing unwanted variations in data generated from instruments. It involves cleansing, transforming, and structuring the data to improve the accuracy of models, reduce computational resources, and prepare the data for effective analysis and machine learning. This essential preliminary step aims to ensure that the data is in a format that can be understood and utilized by machines, ultimately leading to faster knowledge discovery and improved performance of machine learning algorithms (Mishra et al., 2020). To prepare the data to run for the machine learning algorithm, this study is using some data preprocessing techniques. These techniques are discussed in the following section.

To indicate whether a customer's payment is considered overdue, a new column can be created based on the 'STATUS' column. Customers with a 'STATUS' of '2', '3', '4', or '5' are considered to have overdue payments, aligning with the classification of customers who delay payments for more than 60 days as bad debt (Campbell et al., 2022). By setting the Target column to 'Yes' for these 'STATUS' values, a clear distinction can be made between customers whose payments are overdue and those who are not, providing valuable insights for further analysis and make good prediction for decision-making.

3.6.1. Merging

Merging is a fundamental operation in data analysis that combines data from two or more tables. The popular Python library, Pandas, provides the `merge()` function for data manipulation, offering various merge options to suit different needs. The most common merge types are inner join and outer join (Chida et al., 2023). After using the merge operation, the resulting dataset contains 36,457 records and 21 features, consolidating the information from the original tables into a single cohesive dataset for further analysis and visualization

3.6.2. Deleting Unnecessary Columns

The removal of the ID column from the data frame using `DataFrame.drop()` was based on the understanding that it has no impact on the target variable or the derived columns `DAYS_BRITH` and `DAYS_EMPLOYED`, which were represented by Days to create by years through calculations to represent in new columns. Additionally, the columns `begin_month`, utilized in Vintage analysis to assess the quality of different risk periods, were also dropped. This decision indicates that the `customerID` does not contribute to the models' ability to predict customer credit card approval, aligning with the focus on relevant predictors for the credit approval prediction process.

3.6.3. Handling Missing Values

The functions provided by Pandas to detect missing values in the data are `'isna()'` and `'isnull()'`. Using the `'DataFrame.isna().sum()'` function, it was determined that there were 11,323 missing values in the occupation column as shown in Appendix A. A common threshold for considering dropping a column is when more than 30-50% of its values are missing (Bansal et al., 2021). With missing values constituting 31% of the column, the decision was made to drop it from the dataset. Subsequently, the dataset was reduced to 25,134 records after the removal of the 11,323 records with missing values.

3.6.4. Handling outliers

The Interquartile Range (IQR) is a measure of statistical dispersion that captures the spread of the middle 50% of the data, calculated as the difference between the third quartile (Q3) and the first quartile (Q1). Outliers are identified as data instances falling below Q1 minus 1.5 times the IQR or above Q3 plus 1.5 times the IQR, providing an objective method for outlier detection. This approach is valuable for summarizing variability, particularly when there are extreme values at the edges of the dataset (Nnamoko & Korkontzelos, 2020).

When analyzing a dataset, it's important to consider measures of central tendency and dispersion that are less sensitive to outliers. The median and interquartile range (IQR) are particularly useful in this regard. The median represents the middle value in a dataset when arranged in ascending order and is less affected by extreme values. By focusing on these statistics, analysts can gain a more robust understanding of the central tendency and spread of the data, especially in the presence of outliers or skewed distributions (Perez & Tah, 2020).

3.7. Exploratory Data Analysis

Exploratory data analysis (EDA) was employed to comprehensively understand the data and extract meaningful insights. Descriptive statistics provided a summary of the data's characteristics, while univariate and bivariate visualizations unveiled patterns and relationships among features and the target variable. This comprehensive approach facilitated the identification of key features and patterns that influence customer churn, paving the way for informed decision-making (Milo & Somech, 2020).

3.7.1. Descriptive Statistics

To gain an initial understanding of the data, the statistical properties of each feature were examined using the Pandas DataFrame `describe()` function. This function provided insights into the data distribution, including measures such as count, mean, minimum, maximum, and standard deviation. This analysis helped identify potential outliers, assess the normality of the data, and understand the overall range of values for each feature (Petrelli, 2021).

3.7.2. Heatmap Correlation Matrix

Heatmap correlation is a visual representation of the correlation between multiple variables in the form of a color-coded matrix. The color of each cell indicates the strength and direction of the correlation, with darker colors representing stronger correlations. Heatmap correlations are valuable for identifying relationships between variables and detecting multicollinearity issues in datasets. They are particularly useful for exploratory data analysis and can provide insights into the interdependencies among different variables (Patil & Franken, 2021).

3.8. Encoding Categorical Variables

To ensure compatibility with machine learning algorithms that require numerical input, encoding categorical data is necessary. One Hot Encoding from Scikit-learn's pre-processing library using the `OneHotEncoder()`, One Hot Encoding is a widely used method that assigns a unique integer to each categorical label based on alphabetical order and creates new columns to represent the categories. This technique helps address potential bias in the data that can affect detection accuracy (Al-Shehari & Alsowail, 2021). This approach effectively transforms categorical data into a numerical format, enabling machine learning algorithms to process and analyze the data accurately for model training and predictive tasks.

3.9. Weight of Evidence

WOE (Weight of Evidence) and IV (Information Value) are statistical measures used in data analysis and model building, particularly in the context of logistic regression and predictive modeling. Weight of Evidence (WOE) is a technique used to check the linear relationship of a feature with its dependent feature to be used in the model. It is a variable transformation method for both continuous and categorical features and is considered better than one-hot encoding as it does not increase the complexity of the model. WOE helps to understand if a particular class of an independent variable has a higher distribution of good or bad. It is often used in logistic regression

to transform categorical variables into a continuous form that can be used in modeling. Information Value (IV) is a measure of the predictive power of a feature and also helps point out the suspicious feature. It can be used to understand the strength of the relationship between the independent variable and the dependent variable in logistic regression. IV is calculated based on the WOE and is a good measure of the predictive power of a feature (Abraham et al., 2020).

Equation 1:Weight of Evidence

$$WOE = \ln\left(\frac{good}{bad}\right)$$

Equation 2: Information Value Information Value

$$IV = \sum_{i=1}^N (Good\ Proportion - Bad\ Proportion) * WOE_i$$

Table 4: Relationship between IV value and predictive power:

IV	Description
<0.02	Almost no predictive power
0.02 to 0.1	weak predictive power
0.1 to 0.3	Moderate predictive power
0.3~0.5	Strong predictive power
>0.5	Predictive power is too strong, need to check variables

3.10. Feature Scaling

Standard scaling transforms features by subtracting the mean and dividing by the standard deviation using the StandardScaler (), essentially centering and rescaling all values to fall within a similar range (often around zero, with a unit spread). This helps algorithms treat features equally, prevents outliers from dominating the training process, and improves model convergence, ultimately leading to better performance and generalizability (J. M. Chen, 2021).

3.11. Synthetic Minority Over-sampling Technique (SMOTE)

Synthetic Minority Over-sampling Technique (SMOTE) is a preprocessing method used to address class imbalance in a dataset. It involves generating synthetic samples for the minority class to overcome the overfitting problem posed by random oversampling. By focusing on the feature

space and using interpolation between existing positive instances, SMOTE creates new instances to balance the class distribution. This technique is particularly valuable for imbalanced classification problems, where the minority class is underrepresented compared to the majority class. SMOTE aims to improve the performance of machine learning models by providing a more balanced and representative training dataset, ultimately enhancing the model's ability to accurately classify minority class instances (Pradipta et al., 2021).

In a study utilizing the Credit Card dataset from Kaggle, it was observed to be extremely imbalanced, with a significantly smaller number of "bad" customers compared to "good" ones. To address this issue and enhance the effectiveness of machine learning algorithms, synthetic samples were generated using the Synthetic Minority Over-sampling Technique (SMOTE). This method aims to balance the dataset by creating synthetic samples for the minority class, thereby improving the performance of machine learning algorithms. Specifically, SMOTE was applied after splitting the dataset into training and testing sets, exclusively on the training set to prevent information leakage. The implementation of SMOTE was carried out using the imblearn library in Python, which offers various resampling methods. By leveraging SMOTE, the accuracy of predicting credit card fraud can be enhanced, ultimately leading to improved results in credit card fraud detection algorithms.

3.12. Split Data

To evaluate the performance of the machine learning models, the dataset was divided into training and testing sets using the `train_test_split()` function from the Scikit-learn model selection library. The training set, which comprised 70% of the data, was used to train the models, while the testing set, which comprised the remaining 30% of the data, was used to evaluate their performance (Alam et al., 2020).

Table 5: Size of training and test dataset

Properties of dataset	Training	Testing	Total
Number of observations	34596	14828	49424
Percentage of partition	70%	30%	100%

3.13. Feature Selection

Feature selection is the process of isolating the most consistent, non-redundant, and relevant features to use in model construction. It involves systematically reducing the size of datasets to improve the performance of predictive models and reduce the computational cost of modeling. Feature selection is a crucial component of feature engineering, aiming to select the most important features for input into machine learning algorithms. This process is essential for reducing the number of input variables by eliminating redundant or irrelevant features and narrowing down the set of features to those most relevant to the machine learning model. The goal of feature selection is to enhance the efficiency of supervised models like classification and regression, ultimately leading to optimized models in machine learning (Khaire & Dhanalakshmi, 2022).

Table 6: Features and the target variable

Variables	Columns
Features	Gender, Car, Reality, IncomeType, EducationType, Martial_status, HousingType, FLAG_MOBIL, WorkPhone, Phone, Email, OccupationType, ChildNoegp, gp_Income, gp_Age, 'gp_worktm',famsizegp .
Target Variable	Target, binary 0 and 1

3.13.1. Recursive Feature Elimination (RFE)

In a study focusing on the prediction of customers' credit card behavior, the Recursive Feature Elimination (RFE) method will be employed to determine the significance of features for champions Model XGBoost. RFE is a feature selection technique that aims to reduce the complexity of a model by iteratively removing less important features, thereby enhancing the training speed, interpretability, and performance of the model. This process involves ranking features based on their importance and recursively eliminating a small number of features per loop until the optimal number of features needed for peak performance is achieved. RFE helps in removing unnecessary noise generated from less important features, addressing dependencies and collinearity between input features, and preventing overfitting in machine learning models. By leveraging RFE, the accuracy and efficiency of predictive models can be significantly improved.

3.14. Data Modelling

At this stage, several classification algorithms, including Random Forest, Decision Tree, Support Vector Machine, Logistic Regression, LGBM, and XGBoost, were built. Each model has an object created to store the algorithm and specified parameters. The `fit ()` function is then used to train the model on the training set (X-train and y_train), where X represents the features and y represents the target. Subsequently, the trained model is used to make predictions on the test set (X_test) using the `predict ()` function.

Used the Grid search to the best hyperparameters for determined champions Model XGBoost, Grid search is a popular hyperparameter optimization technique used to find the best set of hyperparameters for a machine learning model. It works by creating a grid of all possible hyperparameter combinations and then systematically searching through the grid to identify the combination that yields the best performance. By exhaustively evaluating each combination, grid search helps to identify the optimal hyperparameters for a given model, making it a valuable tool for improving model performance and generalization (Belete & Huchaiah, 2022).

the XGBoost model from the `sklearn.ensemble` was configured with hyperparameters including a learning rate = 0.1, maximum depth = 20, `n_estimators`= 500, and a subsample = 0.5. XGBoost, standing for Extreme Gradient Boosting, is known for its efficient training and high performance, making it a popular choice for classification tasks.

3.15. Performance Metrics

The performance of the classification models was assessed using three key metrics: F1-score, accuracy, recall, and precision. These metrics were calculated based on the confusion matrix, which summarizes the number of correct and incorrect predictions made by each model.

Table 7: Confusion Matrix

		Predicted Label	
		Good-No (0)	Bad-Yes (1)
Actual Label	Good-No (0)	True Negative (TN)	False Positive (FP)
	Bad-No (1)	False Negative (FN)	True Positive (TP)

3.15.1. Accuracy

Accuracy represents the proportion of predictions that were correct. It is calculated as the total number of correct predictions divided by the total number of predictions. The formula for calculating accuracy is defined in Equation (3).

Equation 3: Equation of Accuracy

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

In classification tasks, various performance metrics are used to evaluate the effectiveness of a model. These metrics include True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN). Let's delve into the significance of these metrics and how they contribute to assessing the performance of a classification model.

- True Positive, True Negative, False Positive, and False Negative
- True Positive (TP): This represents the number of observations that belong to the accurately predicted positive class.
- True Negative (TN): It reports on observations that belong to the accurately predicted negative class.
- False Positive (FP): Also known as Type I error, it reveals observations that belong to the negative class but were projected to be in the positive class.
- False Negative (FN): Also known as Type II error, it reveals observations that belong to the positive class but were projected to be in the negative class.

Accuracy, a common metric for evaluating machine learning models, can be misleading when dealing with imbalanced datasets. Due to the inflated accuracy scores, alternative metrics like precision, recall, and F1-score should be considered to assess the model's performance on the minority class (Vuttipittayamongkol et al., 2021).

3.15.2. Recall

Recall, also known as sensitivity, measures the proportion of actual positive cases that are correctly identified. It is calculated as the number of true positives (TP) divided by the total number of actual positive cases (TP + FN). The formula for calculating Precision is defined in Equation (4).

Equation 4: Equation of Recall

$$\text{Recall} = \frac{\text{TP}}{(\text{TP} + \text{FN})}$$

3.15.3. Precision

Precision measures the proportion of positive predictions that are actually correct. It is calculated as the number of true positives (TP) divided by the total number of predicted positive cases (TP + FP). The formula for calculating Precision is defined in Equation (5).

Equation 5: Equation Precision

$$\text{Precision} = \frac{\text{TP}}{(\text{TP} + \text{FP})}$$

3.15.4. F1-score

The F1-score strikes a balance between precision and recall, two crucial metrics in classification tasks. Precision measures the proportion of positive predictions that are actually correct, while recall measures the proportion of actual positive cases that are correctly identified. The F1-score is the harmonic mean of precision and recall, providing a balanced assessment of both metrics. The formula for calculating Precision is defined in Equation (6).

Equation 6: Equation of F1-score

$$\text{F1 - Score} = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

Recall measures the proportion of positive instances that are correctly identified as positive by a classification model. It is calculated by dividing the number of true positives by the sum of true positives and false negatives. A high recall value indicates that the model is effectively detecting all positive instances, while a low value suggests that the model is missing many positive instances.

3.16. Data Visualization

In a study aimed at better understanding the credit card dataset, a credit card approval prediction dashboard will be built using Power BI Desktop. This dashboard will visualize the probability of a customer's credit card approval and provide insights regarding the five most significant features

of credit card approval. The significant features of credit card approval that will be visualized in the dashboard include worktmgp, agegp, Martial_status, Reality, and Gender.

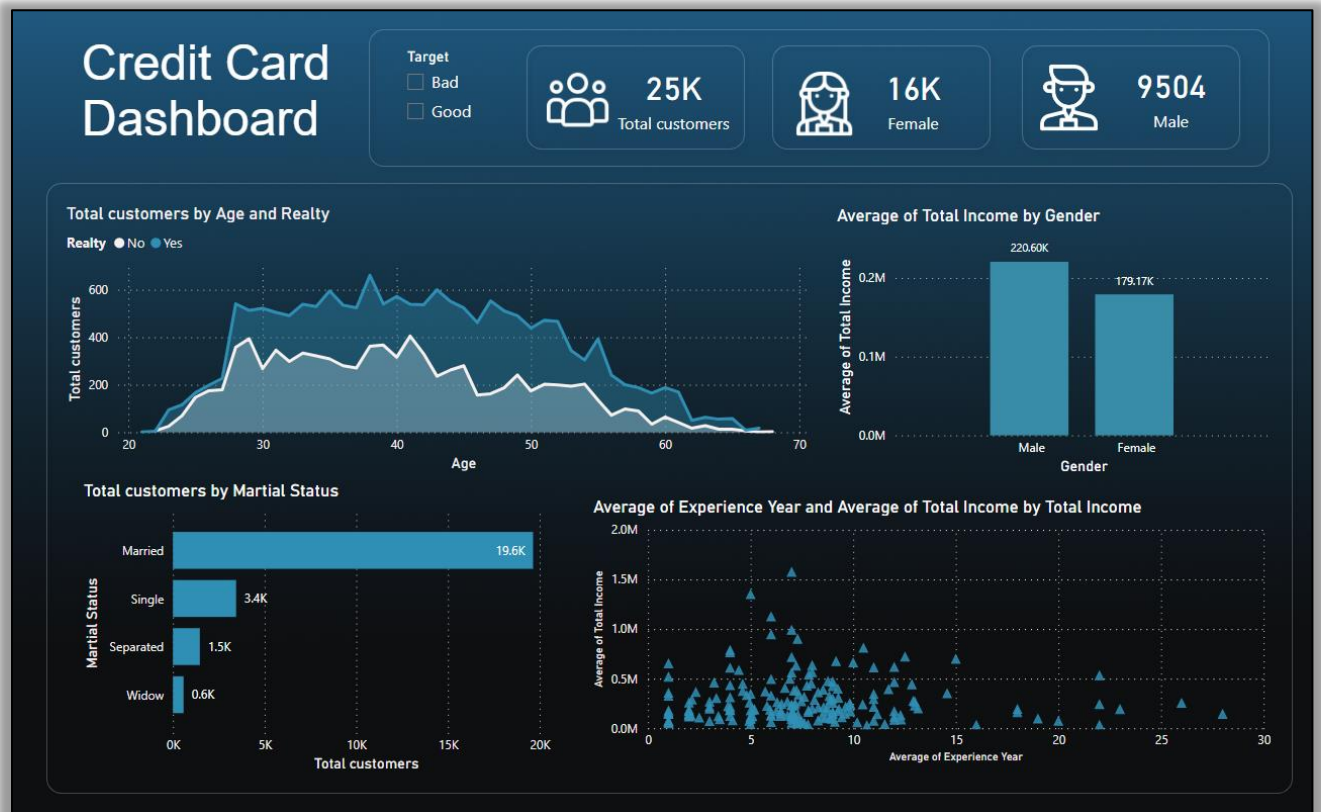


Figure 2: Credit Card Dashboard

3.17. Summary of Chapter 3

In the study, the methodology discussed encompasses data pre-processing, data modeling, and data visualization. The data pre-processing phase involves data exploration, encoding of categorical data, Vintage analysis, feature scaling, WEO, and data splitting. Additionally, the machine learning algorithm section includes feature selection using Recursive Feature Elimination (RFE) and the creation and evaluation of supervised machine learning algorithms such as Random Forest, Decision Tree, Support Vector Machine, Logistic Regression, Xgboost, and LGBM. Finally, the study plans to visualize using a dashboard to present the insights gained from the data pre-processing and machine learning model evaluation.

CHAPTER 4 RESULT AND DISCUSSION

4.1. Introduction

Chapter 4 analyzed the performance of machine learning algorithms in predicting credit card approvals. Exploiting insights from Exploratory Data Analysis and feature selection, the study tackled data imbalance and evaluated algorithm performance. The "champion" algorithm with the highest accuracy was identified, and its visualization through a dashboard was explained.

4.2. Vintage Analysis

Analyzing the trends using Vintage Analysis. For example, we can examine the average balance for customers opened in different quarters and observe their trend in the subsequent months after the account opening date. In the charts shown below, we are presenting vintage analysis of average balance and utilization ratio. This is a standard vintage analysis table. The rows represent months of opening accounts, columns represent months after opening accounts, and values are accumulating past-due rate. As open-month closes to 0.

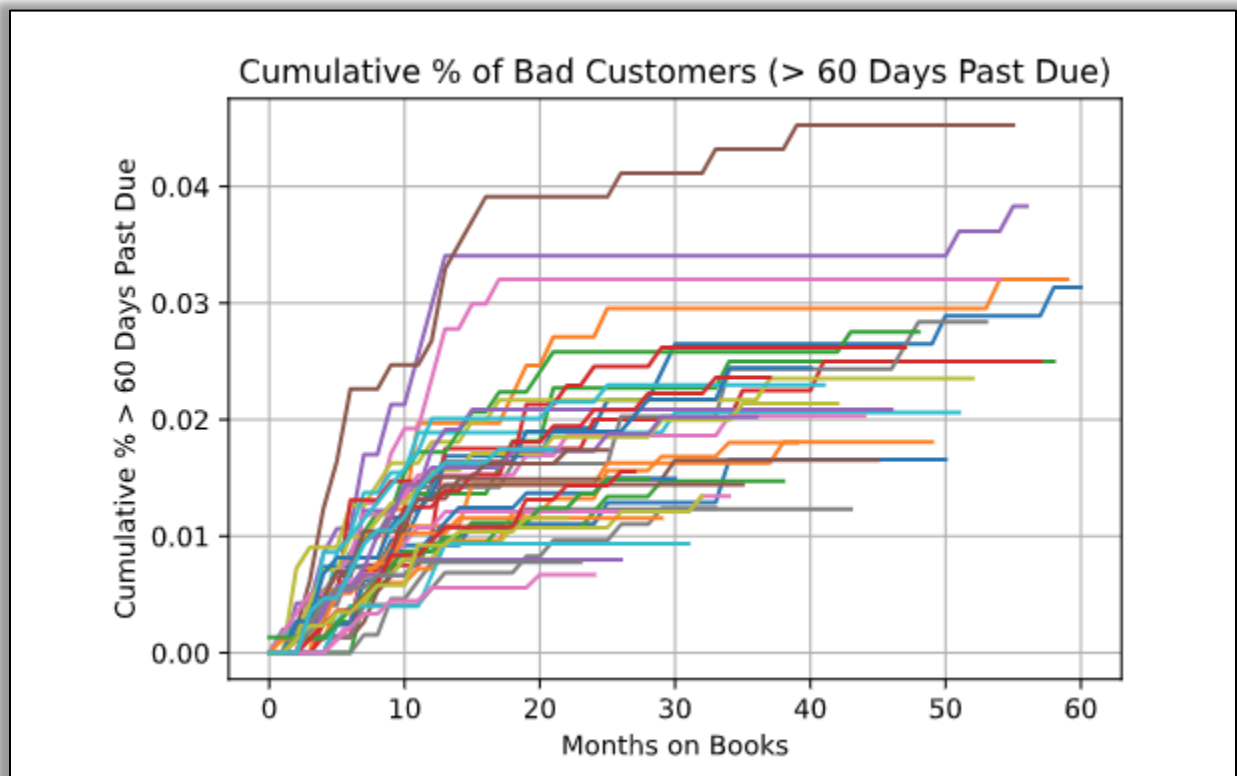
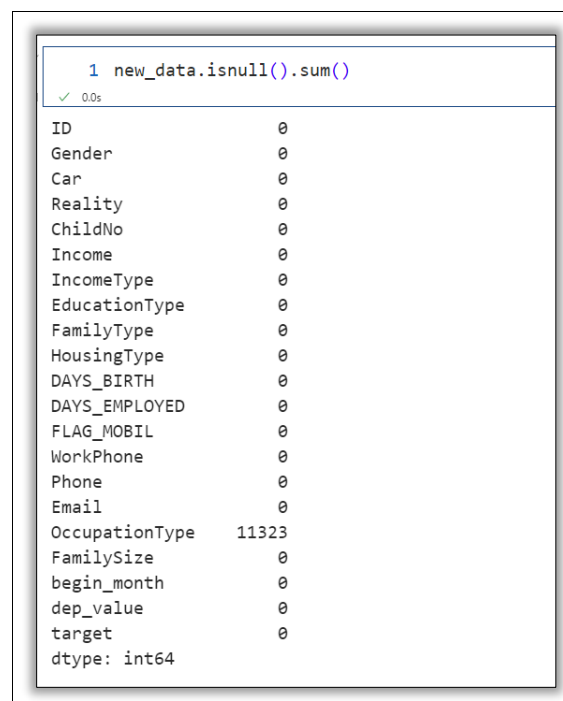


Figure 3: cumulative percentage of bad customers

The graph itself shows the cumulative percentage of bad customers (defined as those who are more than 60 days past due) over time. It shows that the percentage of bad customers has been increasing steadily over the past few months. This could be due to a number of factors, such as changes in the economy, the company's credit policies, or the quality of its customer base.

4.3. Result of Exploratory Data Analysis

Before merging, the first table contains 438,557 records, while the second table contains 1,048,575 records. After merging the two tables using an ID for a left join, we are left with 36,457 records and 21 features. Upon checking for missing values in the data, we discovered that there are approximately 11,323 records with missing values in only the OCCUPATION column, accounting for 31.06% of the column. As a result, we have made the decision to drop these rows. Additionally, we examined duplicated records excluding the ID column, and found 2,937 duplicates. Consequently, we have decided to remove these duplicates as well. Now have a dataset with 33,520 unique records and 21 features.



```
1 new_data.isnull().sum()
```

ID	0
Gender	0
Car	0
Reality	0
ChildNo	0
Income	0
IncomeType	0
EducationType	0
FamilyType	0
HousingType	0
DAYS_BIRTH	0
DAYS_EMPLOYED	0
FLAG_MOBIL	0
WorkPhone	0
Phone	0
Email	0
OccupationType	11323
FamilySize	0
begin_month	0
dep_value	0
target	0
dtype: int64	

Figure 4: Detect Missing Value

This study involves a comprehensive analysis of a dataset that includes both numeric and categorical variables. Descriptive statistics have been generated to provide insights into the characteristics of each numeric feature, such as count, mean, standard deviation, minimum and maximum values, and percentile values. The descriptive statistics, as presented in Table (8), reveal important information about the dataset.

Table 8 : *Descriptive Statistics*

	count	mean	std	min	25%	50%	75%	max
ChildNo	36457.0	4.303152e-01	0.742367	0.0	0.0	0.0	1.0	19.0
Income	36457.0	1.866857e+05	101789.226482	27000.0	121500.0	157500.0	225000.0	1575000.0
DAYS_BIRTH	36457.0	-1.597517e+04	4200.549944	-25152.0	-19438.0	-15563.0	-12462.0	-7489.0
DAYS_EMPLOYED	36457.0	5.926294e+04	137651.334859	-15713.0	-3153.0	-1552.0	-408.0	365243.0
Begin_month	36457.0	-2.616419e+01	16.501854	-60.0	-39.0	-24.0	-12.0	0.0

For the numeric feature there are detect that their outliers in the data.

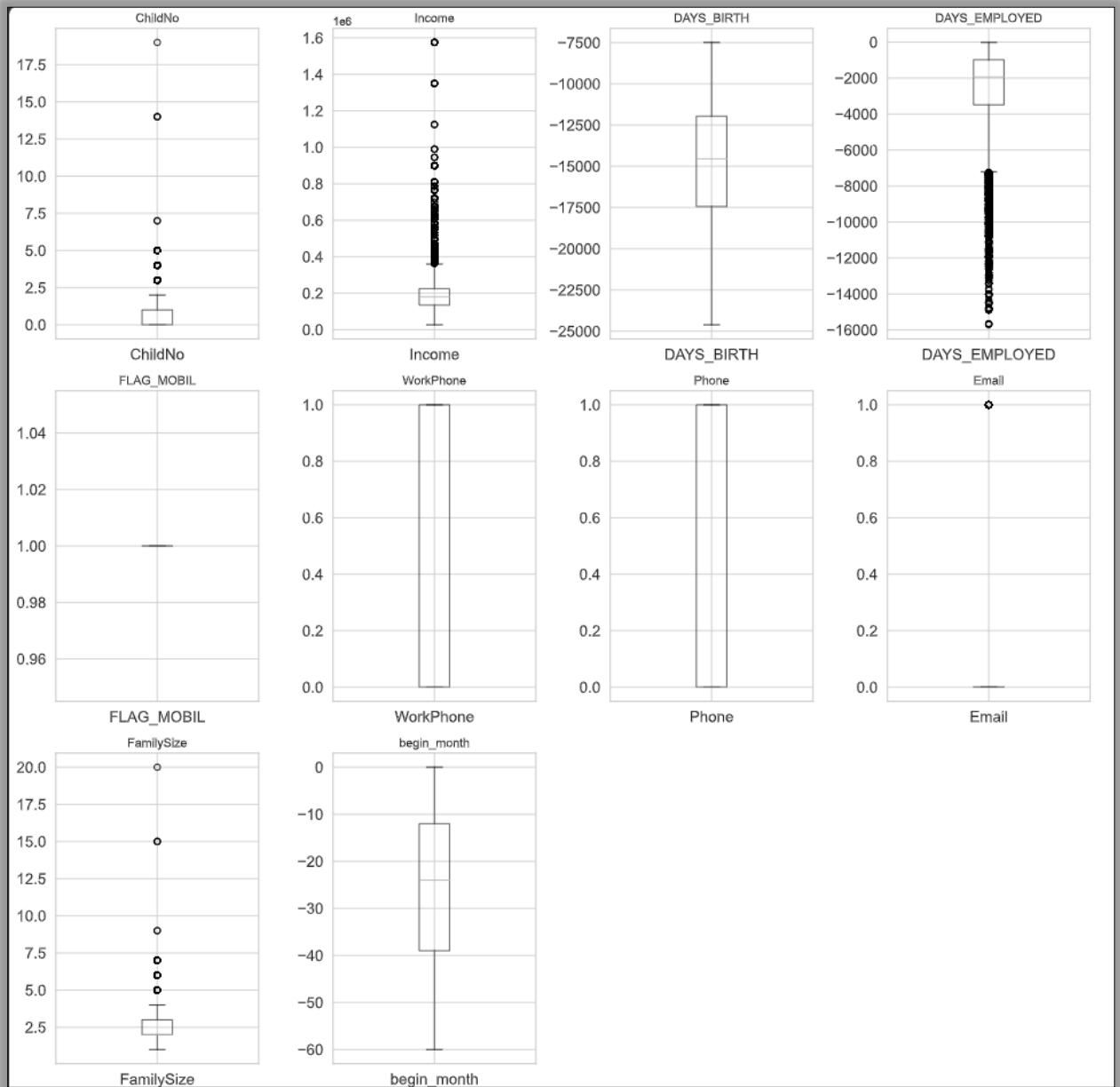


Figure 5: Detect outliers

Outliers are data points that deviate significantly from the rest of the observations in a dataset. In data, the columns ChildNO, income, DAYS_Employed, Email, and Familysize contain outliers. These outliers can have a significant impact on analysis and may lead to misleading results.

Statistical tests can be employed to detect outliers based on certain criteria, such as the 1.5xIQR rule. Once identified, outliers be handled by replacing them with appropriate Median values.

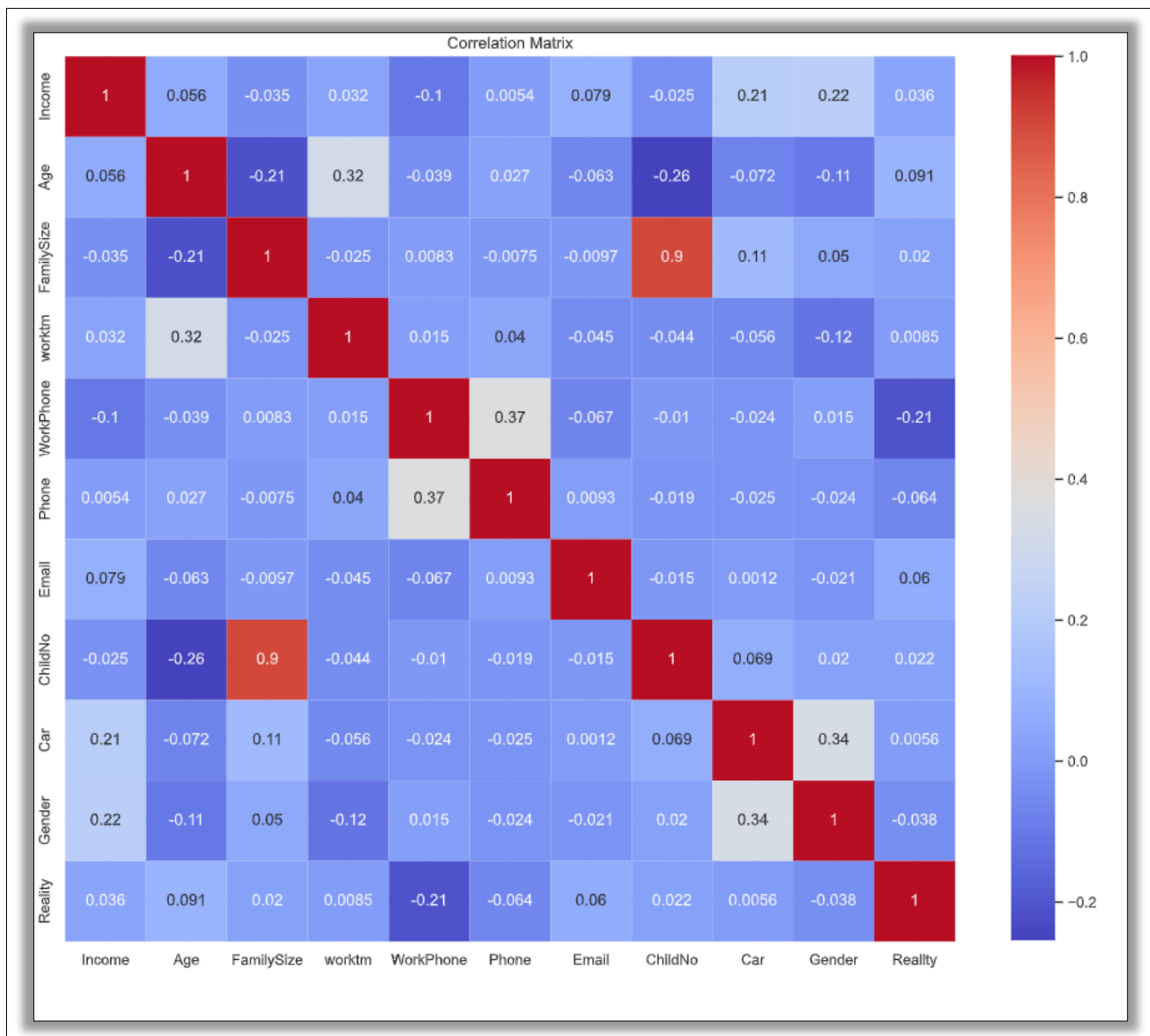


Figure 6: Correlation Matrix

Based on the correlation matrix, it appears that " ChildNo " has a strong positive relationship with " Family Size " with correlation values of 0.9, respectively, indicating a strong association between these features.

4.4. Weight of Evidence (WOE):

the Weight of Evidence (WOE) technique assesses and transforms categorical features to boost their predictive power, aiding in identifying informative features, enhancing model performance, and improving interpretability. Its statistical calculations and Information Value measure reveal

insights into feature-target relationships, making it valuable for applications like credit risk modeling and medical diagnosis.

Analyzing correlations with the target variable, Worktmgp reigns supreme with 0.098 influence, followed closely by Agegp (0.066), Marital_status (0.032), and reality (0.027). While features like Gender (0.025), and Income () contribute less prominently, their impact shouldn't be overlooked, as even smaller correlations like and OccupationType (0.65%) can subtly contribute to overall model accuracy.

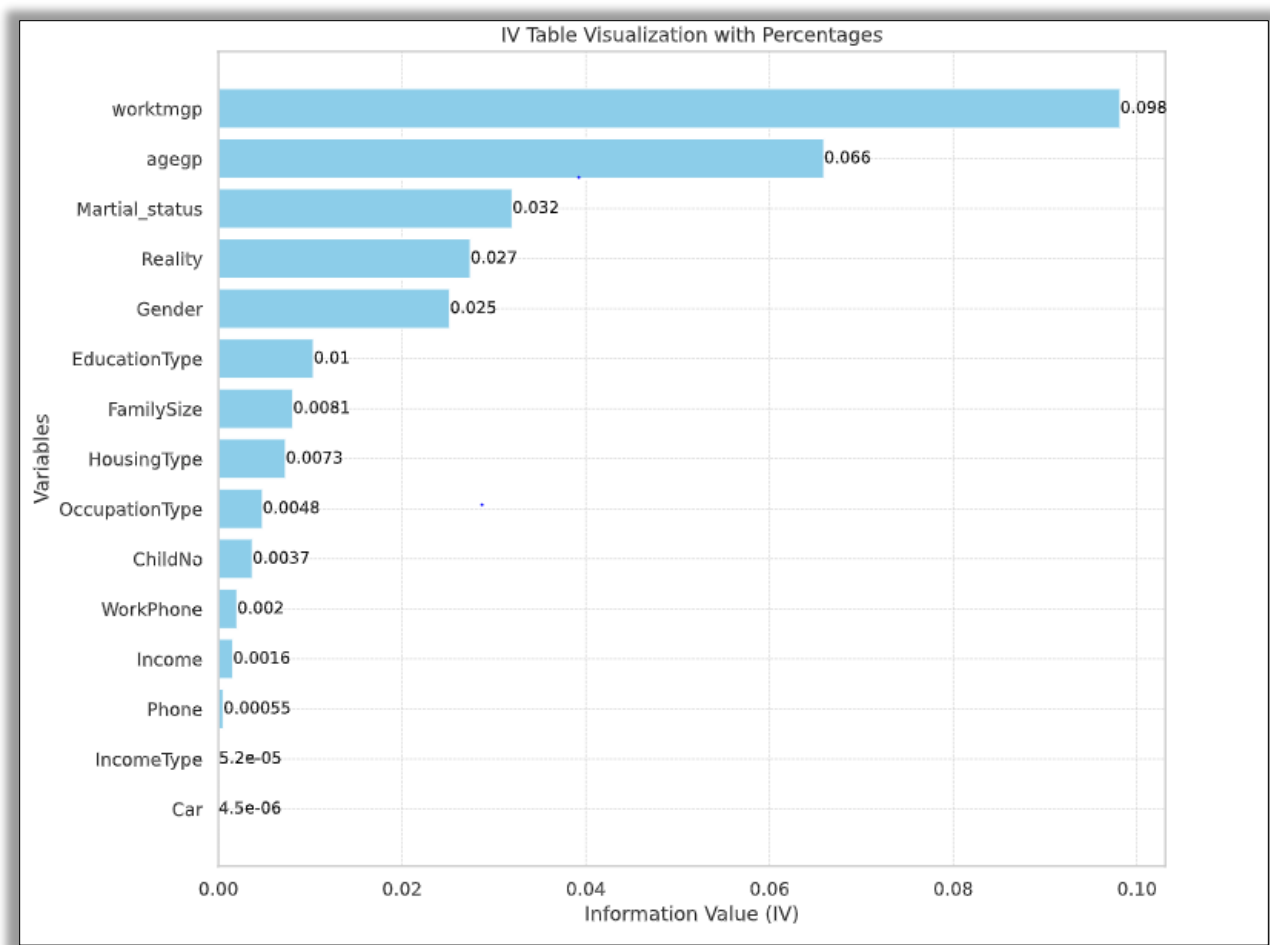


Figure 7: IV Table Visualization with percentages

4.5. Machine Learning Performance

The performance of machine learning algorithms in predicting customer credit card approval can be evaluated using various performance indicators. These indicators include the confusion

matrix, accuracy score, precision, recall, F1 score, and ROC curve. By analyzing these measures, we were able to identify the champion algorithm, which accurately predicts whether customers are good candidates for credit card approval with a high level of confidence. The results of the confusion matrix and the performance metrics for the selected prediction model are detailed in the following section.

4.6. Comparison of the Confusion Matrices

The confusion matrix is a table used to assess the performance of a classification model. It summarizes the counts of true positives, true negatives, false positives, and false negatives. These values are crucial for evaluating the accuracy, precision, recall, and F1-score of the model. It's important to note that the confusion matrix provides a more comprehensive understanding of the model's performance compared to simple accuracy measurements. It allows us to analyze the number of correctly and incorrectly classified instances for each class, providing valuable insights into the model's performance.

The confusion matrix for the logistic regression model is presented in Table 4.3. According to the results, the logistic regression algorithm accurately classified 4420 customers as good candidates for credit card approval and 5003 customers as bad candidates for credit card approval. However, the algorithm misclassified 2994 customers by classifying them as bad candidates when they were actually good candidates. Additionally, the algorithm misclassified 2411 customers by classifying them as good candidates when they were actually bad candidates.

Table 9: The confusion matrix of logistic regression

		Predicted Label		
True Label	observation	Good (0)	Bad (1)	Total
	Good (0)	4420	2994	7414
	Bad (1)	2411	5003	7414
	Total	6831	7997	14828

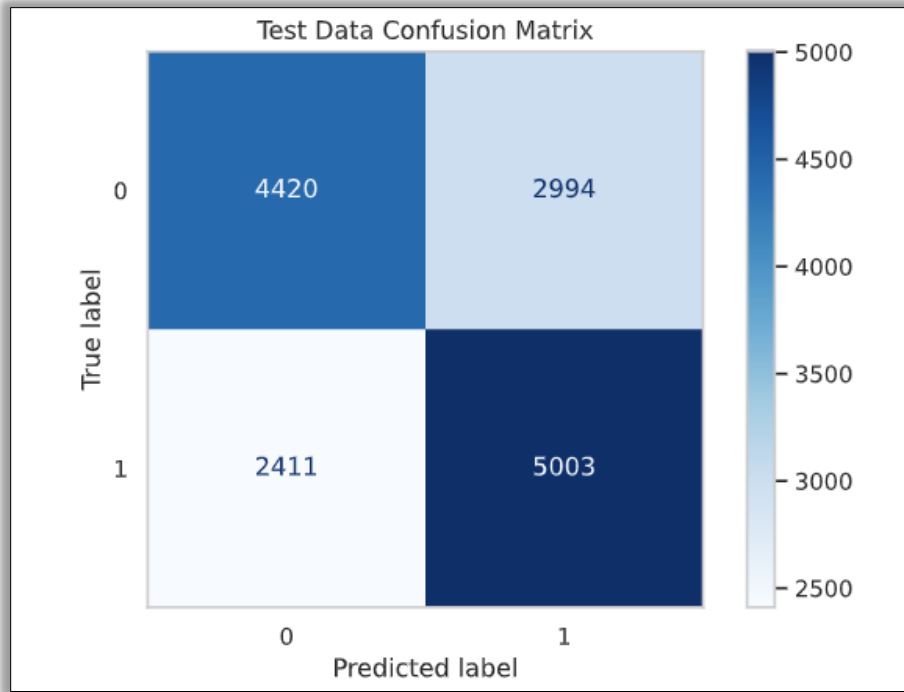


Figure 8: *The confusion matrix of logistic regression*

The confusion matrix for the Decision Tree Classifier model is presented in Table 4.3. According to the results, the Decision Tree Classifier algorithm accurately classified 6650 customers as good candidates for credit card approval and 6944 customers as bad candidates for credit card approval. However, the algorithm misclassified 764 customers by classifying them as bad candidates when they were actually good candidates. Additionally, the algorithm misclassified 470 customers by classifying them as good candidates when they were actually bad candidates.

Table 10: *The confusion matrix of Decision Tree Classifier*

		Predicted Label		
		Good (0)	Bad (1)	Total
True Label	observation			
	Good (0)	6650	764	7414
	Bad (1)	470	6944	7414
Total		7120	7708	14828

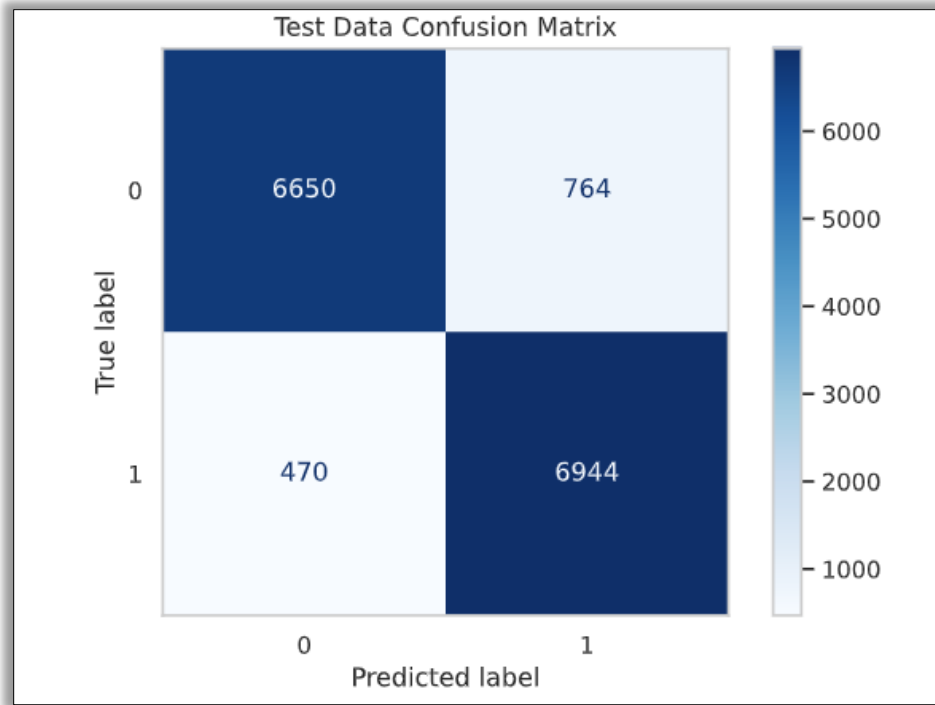


Figure 9: The confusion matrix of Decision Tree Classifier

The confusion matrix for the Random Forest Classifier model is presented in Table 4.3. According to the results, the Random Forest Classifier algorithm accurately classified 6686 customers as good candidates for credit card approval and 6984 customers as bad candidates for credit card approval. However, the algorithm misclassified 728 customers by classifying them as bad candidates when they were actually good candidates. Additionally, the algorithm misclassified 430 customers by classifying them as good candidates when they were actually bad candidates.

Table 11: The confusion matrix of Decision Tree Classifier

		Predicted Label		
		Good (0)	Bad (1)	Total
True Label	observation			
	Good (0)	6686	728	7414
	Bad (1)	430	6984	7414
Total		7116	7712	14828

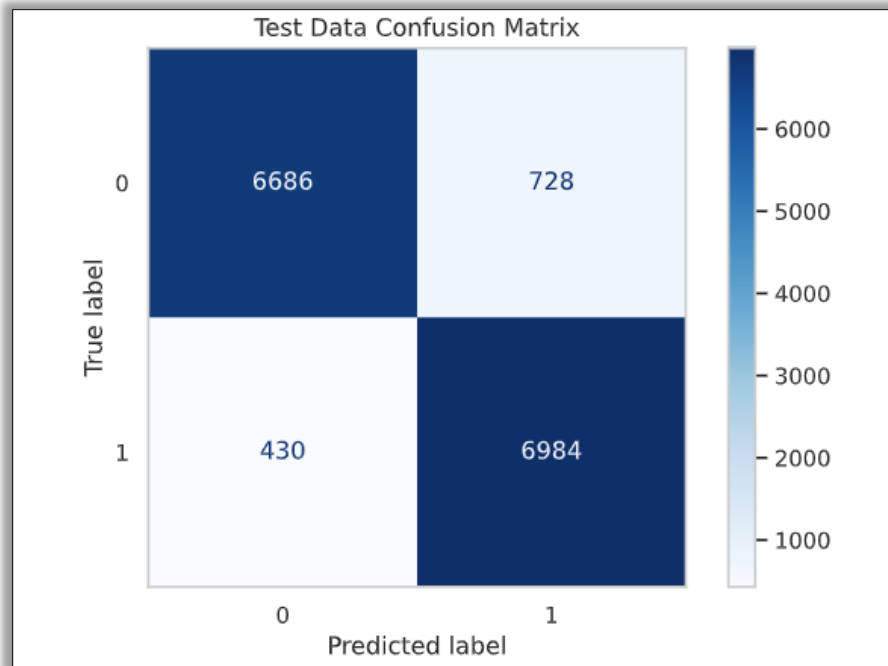


Figure 10: The confusion matrix of Decision Tree Classifier

The confusion matrix for the SVM model is presented in Table 4.3. According to the results, the SVM algorithm accurately classified 6016 customers as good candidates for credit card approval and 1398 customers as bad candidates for credit card approval. However, the algorithm misclassified 1398 customers by classifying them as bad candidates when they were actually good candidates. Additionally, the algorithm misclassified 6748 customers by classifying them as good candidates when they were actually bad candidates.

Table 12: The confusion matrix of SVM

		Predicted Label		
		Good (0)	Bad (1)	Total
True Label	observation			
	Good (0)	6016	1398	7414
	Bad (1)	666	6748	7414
Total		6682	8146	14828

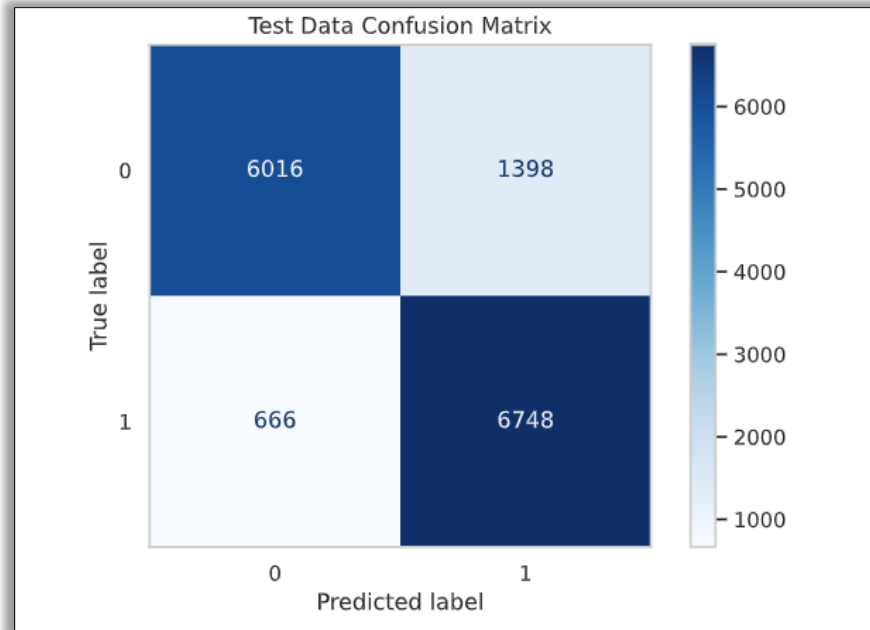


Figure 11: The confusion matrix of SVM

The confusion matrix for the LGBM Classifier model is presented in Table 4.3. According to the results, the LGBM Classifier algorithm accurately classified 6661 customers as good candidates for credit card approval and 6985 customers as bad candidates for credit card approval. However, the algorithm misclassified 753 customers by classifying them as bad candidates when they were actually good candidates. Additionally, the algorithm misclassified 429 customers by classifying them as good candidates when they were actually bad candidates.

Table 13: The confusion matrix of LGBM Classifier

		Predicted Label		
observation		Good (0)	Bad (1)	Total
True Label	Good (0)	6661	753	7414
	Bad (1)	429	6985	7414
	Total	7090	7738	14828

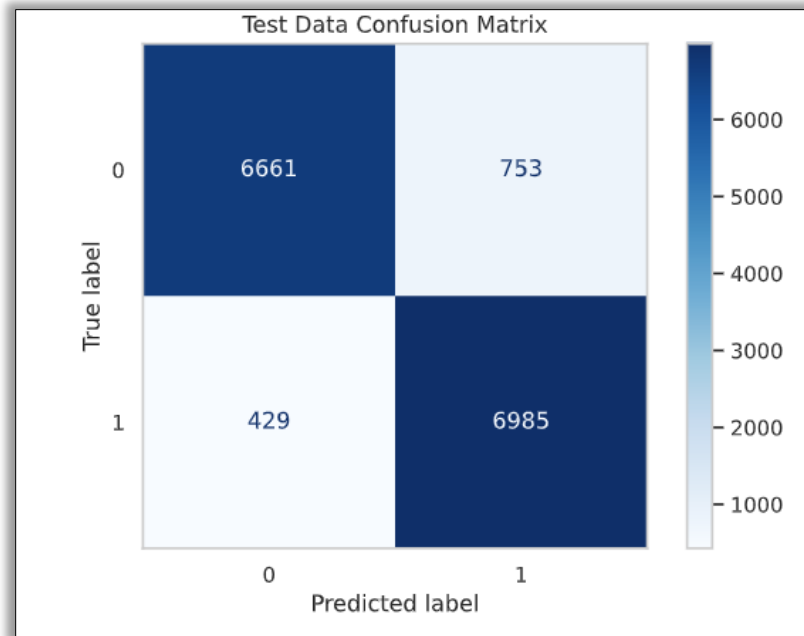


Figure 12: The confusion matrix of LGBM Classifier

The confusion matrix for the XGB Classifier model is presented in Table 4.3. According to the results, the XGB Classifier algorithm accurately classified 6695 customers as good candidates for credit card approval and 7061 customers as bad candidates for credit card approval. However, the algorithm misclassified 719 customers by classifying them as bad candidates when they were actually good candidates. Additionally, the algorithm misclassified 353 customers by classifying them as good candidates when they were actually bad candidates.

		Predicted Label		
		Good (0)	Bad (1)	Total
True Label	observation			
	Good (0)	6695	719	7414
	Bad (1)	353	7061	7414
Total		7048	7780	14828

4.1. Comparison of Performance Metrics

Based on the accuracy score in Table 14, the XGB Classifier outperforms the Random Forest, Decision Tree, Support Vector Machine, Logistic Regression, and LGBM Classifier models. The

XGB Classifier demonstrates the ability to predict the greatest number of True Positives and True Negatives compared to the other models. While accuracy is an important metric, a comprehensive evaluation of model performance should also consider precision, recall, and F1 score to gain a more complete understanding of the models' capabilities. Therefore, while the XGB Classifier shows superior accuracy in this context, a holistic assessment should take into account a range of performance metrics to ensure a thorough evaluation of the machine learning models.

Table 14: The result of accuracy score, precision, recall and F1 score for the model

Model	Accuracy Score	F1 Score	Recall	Precision
Random Forest	0.922%	0.923%	0.922%	0.923%
Decision Tree	0.917%	0.918%	0.917%	0.917%
Support Vector Machine	0.861%	0.867%	0.861%	0.864%
Logistic Regression	0.635%	0.649%	0.635%	0.636%
XGBoost	0.928%	0.929%	0.928%	0.929%
LGBM	0.920%	0.922%	0.920%	0.921%

Based on the data from Table 14, the accuracy scores for the various models are as follows: Random Forest (92.2%), Decision Tree (91.7%), Support Vector Machine (86.1%), Logistic Regression (63.5%), XGBoost (92.8%), and LGBM (92.0%). Additionally, the F1 scores, recall, and precision for each model are also provided in the table.

The XGBoost model stands out with the highest accuracy score of 92.8%, closely followed by Random Forest and LGBM with scores of 92.2% and 92.0% respectively. Logistic Regression lags behind with an accuracy score of 63.5%. These scores provide valuable insights into the overall performance of the models in making correct predictions.

It's important to note that accuracy scores alone may not provide a complete picture of a model's performance, and considering additional metrics such as F1 score, recall, and precision is crucial for a comprehensive evaluation. These metrics collectively offer a more nuanced understanding of a model's predictive capabilities and its ability to minimize misclassifications.

4.7. Summary of Chapter 4

Chapter 4 of the study discusses the findings of the machine learning algorithms employed, including Random Forest, Decision Tree, Support Vector Machine, Logistic Regression, XGBoost, and lightgbm. The performance of each machine learning algorithm is assessed using performance metrics, allowing for the identification of the most effective algorithm. Additionally, Exploratory Data Analysis is discussed in detail within this chapter. The chapter concludes with an explanation of the developed dashboard.

CHAPTER 5 CONCLUSION

5.1. Summary and Recommendations

This study utilized various machine learning algorithms, such as Random Forest, Decision Tree, Support Vector Machine, Logistic Regression, Xgboost, and LGBM, to predict customer credit card usage. The primary goal was to determine the most accurate algorithm and provide insights for effective credit risk management in the banking industry. To prepare the data for modeling, categorical variables were transformed into numerical format, and vintage analysis was conducted to assess risk quality. The dataset was then split into training and testing sets, with 70% allocated to training and 30% to testing. The study successfully achieved its research objectives and addressed the research questions, highlighting the effectiveness of the machine learning algorithms in predicting credit card usage.

Based on the findings and results obtained from the machine learning algorithms for predicting customer credit card approval, the following recommendations are suggested for financial institutions and credit card companies:

Firstly, financial institutions should focus on enhancing predictive accuracy by emphasizing the most influential features identified, such as `workmgn`, `Agegp`, `Martial_status`, `Gender` and `Reality`. By prioritizing these features during the credit evaluation process, institutions can improve their predictive models, leading to more accurate credit card approval decisions.

Secondly, leveraging the XGBoost algorithm, which demonstrated superior performance metrics, should be a priority for institutions aiming to optimize their credit card approval systems. By integrating the XGBoost algorithm into their existing systems, institutions can benefit from enhanced prediction accuracy, thereby reducing the risk associated with defaulting customers.

Furthermore, addressing outliers and missing values in the dataset is crucial to ensuring the reliability and effectiveness of the predictive model. Financial institutions should implement robust data preprocessing techniques, including outlier detection and data imputation, to enhance the quality of the dataset and improve the model's performance.

Additionally, financial institutions should establish a systematic approach to regularly update and refine the predictive model based on new data and evolving economic conditions. By continuously monitoring the model's performance and incorporating new data, institutions can ensure that their

credit card approval systems remain accurate, relevant, and aligned with current credit risk factors and customer behavior patterns.

Lastly, to minimize the cost or effects resulting from false positives and false negatives, financial institutions should invest in comprehensive training programs and resources for their credit evaluation teams. Providing employees with the necessary tools, knowledge, and expertise to interpret and utilize the predictive model effectively will enhance decision-making consistency and accuracy, ultimately leading to improved customer satisfaction and retention.

By implementing these recommendations, financial institutions and credit card companies can enhance their credit card approval processes, mitigate risks associated with defaulting customers, and foster a more efficient and reliable credit evaluation system.

5.2. Limitation

Despite the promising capabilities of machine learning algorithms in predicting credit card approval outcomes, several inherent limitations must be acknowledged to ensure a comprehensive understanding of the study's scope and applicability. The dataset used for the credit card approval project is relatively small, which can limit the model's ability to generalize and make accurate predictions. A small dataset may lead to overfitting and reduced model performance, especially when dealing with complex patterns and relationships within the data. Additionally, the dataset contains outliers and missing values, which can introduce noise and bias into the model's predictions. Outliers can significantly impact statistical analyses and model performance, while missing values require careful handling during data preprocessing to ensure the model can make accurate predictions. The credit card approval dataset may have an imbalanced distribution of approved and rejected applications, which can affect the model's ability to accurately predict both classes. Imbalanced data can lead to biased model predictions and may require specific techniques such as resampling or using different evaluation metrics to address the imbalance. Furthermore, the dataset contains a mixture of numerical and non-numerical features with values from different ranges, which may require extensive feature engineering to extract meaningful information for the model. Feature engineering complexities can impact the model's ability to capture relevant patterns and relationships within the data.

In addition to these limitations, there are some other important considerations. Firstly, a critical limitation of this study pertains to the scope and granularity of the features incorporated into the

predictive model for credit card approval. The model's reliance on a limited set of variables may not encompass the multifaceted factors that financial institutions consider during the credit evaluation process. As a result, the model's predictive accuracy and reliability may be compromised, given that it may not fully capture intricate consumer financial behaviors, economic indicators, or evolving market dynamics that significantly influence credit card approval decisions. Secondly, the predominant use of machine learning algorithms as the primary analytical approach in this study presents another limitation. While machine learning algorithms are adept at discerning patterns and making predictions based on historical data, their static nature may introduce biases and fail to adapt to rapidly changing economic conditions, consumer behaviors, and regulatory landscapes. Consequently, the model's predictive capabilities may be constrained, as it may not adequately account for unforeseen variables or emerging trends that could impact credit card approval decisions in real-time scenarios. Furthermore, the study's emphasis on a specific subset of machine learning algorithms, such as Random Forest, Decision Tree, Support Vector Machine, and Logistic Regression, represents an additional limitation. By limiting the analysis to these particular algorithms, the study may overlook alternative modeling techniques or ensemble methods that could offer superior predictive performance, robustness, and interpretability. This constraint may limit the study's generalizability and applicability across different financial institutions, consumer demographics, and credit evaluation frameworks, thereby potentially compromising its broader relevance and impact.

REFERENCES

- Abraham, A., Cherukuri, A. K., Melin, P., & Gandhi, N. (Eds.). (2020). *Intelligent Systems Design and Applications: 18th International Conference on Intelligent Systems Design and Applications (ISDA 2018) held in Vellore, India, December 6-8, 2018, Volume 1* (Vol. 940). Springer International Publishing. <https://doi.org/10.1007/978-3-030-16657-1>
- Ahamed, B. S. (2021). Prediction of type-2 diabetes using the LGBM classifier methods and techniques. *Turkish Journal of Computer and Mathematics Education (TURCOMAT)*, 12(12), 223–231.
- Alam, T. M., Shaukat, K., Hameed, I. A., Luo, S., Sarwar, M. U., Shabbir, S., Li, J., & Khushi, M. (2020). An investigation of credit card default prediction in the imbalanced datasets. *IEEE Access*, 8, 201173–201198.
- Alfaiz, N. S., & Fati, S. M. (2022). Enhanced Credit Card Fraud Detection Model Using Machine Learning. *Electronics*, 11(4), 662. <https://doi.org/10.3390/electronics11040662>
- Aliapoulios, M., Ballard, C., Bhalerao, R., Lauinger, T., & McCoy, D. (n.d.). *Swiped: Analyzing Ground-truth Data of a Marketplace for Stolen Debit and Credit Cards*.
- Al-Shehari, T., & Alsowail, R. A. (2021). An insider data leakage detection using one-hot encoding, synthetic minority oversampling and machine learning techniques. *Entropy*, 23(10), 1258.
- Bansal, P., Deshpande, P., & Sarawagi, S. (2021). Missing value imputation on multidimensional time series. *arXiv Preprint arXiv:2103.01600*.
- Battineni, G., Chintalapudi, N., & Amenta, F. (2020). *Comparative Machine Learning Approach in Dementia Patient Classification using Principal Component Analysis*. 780–784.
- Belete, D. M., & Huchaiah, M. D. (2022). Grid search in hyperparameter optimization of machine learning models for prediction of HIV/AIDS test results. *International Journal of Computers and Applications*, 44(9), 875–886.

- Campbell, D., Grant, A., & Thorp, S. (2022). Reducing credit card delinquency using repayment reminders. *Journal of Banking & Finance*, 142, 106549.
- Charbuty, B., & Abdulazeez, A. (2021). Classification based on decision tree algorithm for machine learning. *Journal of Applied Science and Technology Trends*, 2(01), 20–28.
- Chaudhuri, A. (2020). *Flowchart and algorithm basics: The art of programming*. Mercury Learning and Information.
- Chen, J. M. (2021). An introduction to machine learning for panel data. *International Advances in Economic Research*, 27(1), 1–16.
- Chen, Z., Lin, G., & Jin, X. (2021). *Credit Approval Prediction Based on Boruta-GBM Model*. 1–5.
- Chida, K., Hamada, K., Ichikawa, A., Kii, M., & Tomida, J. (2023). *Communication-Efficient Inner Product Private Join and Compute with Cardinality*. 678–688.
- Das, A. (2021). Logistic regression. In *Encyclopedia of Quality of Life and Well-Being Research* (pp. 1–2). Springer.
- Das, S., Imtiaz, M. S., Neom, N. H., Siddique, N., & Wang, H. (2023). A hybrid approach for Bangla sign language recognition using deep transfer learning model with random forest classifier. *Expert Systems with Applications*, 213, 118914.
- Del Pilar, E. C. D., & Bongo, M. F. (2023). *Towards the Improvement of Credit Card Approval Process Using Classification Algorithm*. 461–465.
- Dobbie, W., & Song, J. (2020). Targeted Debt Relief and the Origins of Financial Distress: Experimental Evidence from Distressed Credit Card Borrowers. *American Economic Review*, 110(4), 984–1018.
<https://doi.org/10.1257/aer.20171541>
- Dumitrescu, E., Hué, S., Hurlin, C., & Tokpavi, S. (2022). Machine learning for credit scoring: Improving logistic regression with non-linear decision-tree effects. *European Journal of Operational Research*, 297(3), 1178–1192.

- Elmachtoub, A. N., Liang, J. C. N., & McNellis, R. (2020). *Decision trees for decision-making under the predict-then-optimize framework*. 2858–2867.
- Esenogho, E., Mienye, I. D., Swart, T. G., Aruleba, K., & Obaido, G. (2022). A neural network ensemble with feature engineering for improved credit card fraud detection. *IEEE Access*, *10*, 16400–16407.
- Fan, S., Shen, Y., & Peng, S. (2020). Improved ML-Based Technique for Credit Card Scoring in Internet Financial Risk Control. *Complexity*, *2020*, 1–14. <https://doi.org/10.1155/2020/8706285>
- Gupta, P., Varshney, A., Khan, M. R., Ahmed, R., Shuaib, M., & Alam, S. (2023). Unbalanced Credit Card Fraud Detection Data: A Machine Learning-Oriented Comparative Study of Balancing Techniques. *Procedia Computer Science*, *218*, 2575–2584.
- Habib, N., Hasan, M. M., Reza, M. M., & Rahman, M. M. (2020). Ensemble of CheXNet and VGG-19 feature extractor with random forest classifier for pediatric pneumonia detection. *SN Computer Science*, *1*, 1–9.
- Hartmann, J. (2021). Classification using decision tree ensembles. In *The Machine Age of Customer Insight* (pp. 103–117). Emerald Publishing Limited.
- Hsieh, D.-Y., Wang, C.-H., & Cheng, G. (2024). *Improve Fidelity and Utility of Synthetic Credit Card Transaction Time Series from Data-centric Perspective* (arXiv:2401.00965). arXiv. <http://arxiv.org/abs/2401.00965>
- Ileberi, E., Sun, Y., & Wang, Z. (2022). A machine learning based credit card fraud detection using the GA algorithm for feature selection. *Journal of Big Data*, *9*(1), 24. <https://doi.org/10.1186/s40537-022-00573-8>
- Kavzoglu, T., & Teke, A. (2022). Predictive Performances of ensemble machine learning algorithms in landslide susceptibility mapping using random forest, extreme gradient boosting (XGBoost) and

- natural gradient boosting (NGBoost). *Arabian Journal for Science and Engineering*, 47(6), 7367–7385.
- Khaire, U. M., & Dhanalakshmi, R. (2022). Stability of feature selection algorithm: A review. *Journal of King Saud University-Computer and Information Sciences*, 34(4), 1060–1073.
- Khan, S. A., Zainuddin, M., Mahi, M., & Arif, I. (2020). *Behavioral Intention to Use Online Learning During COVID-19: An Analysis of the Technology Acceptance Model*.
- Kozodoi, N., Jacob, J., & Lessmann, S. (2022). Fairness in credit scoring: Assessment, implementation and profit implications. *European Journal of Operational Research*, 297(3), 1083–1094.
- Land Jr, W. H., Schaffer, J. D., Land, W. H., & Schaffer, J. D. (2020). The support vector machine. *The Art and Science of Machine Intelligence: With An Innovative Application for Alzheimer's Detection from Speech*, 45–76.
- Mishra, P., Biancolillo, A., Roger, J. M., Marini, F., & Rutledge, D. N. (2020). New data preprocessing trends based on ensemble of multiple preprocessing techniques. *TrAC Trends in Analytical Chemistry*, 132, 116045.
- Naik, K. (2021). Predicting Credit Risk for Unsecured Lending: A Machine Learning Approach. *arXiv Preprint arXiv:2110.02206*.
- Nnamoko, N., & Korkontzelos, I. (2020). Efficient treatment of outliers and class imbalance for diabetes prediction. *Artificial Intelligence in Medicine*, 104, 101815.
<https://doi.org/10.1016/j.artmed.2020.101815>
- Osman, A. I. A., Ahmed, A. N., Chow, M. F., Huang, Y. F., & El-Shafie, A. (2021). Extreme gradient boosting (Xgboost) model to predict the groundwater levels in Selangor Malaysia. *Ain Shams Engineering Journal*, 12(2), 1545–1556.
- Paoletti, M. E., Haut, J. M., Tao, X., Miguel, J. P., & Plaza, A. (2020). A new GPU implementation of support vector machines for fast hyperspectral image classification. *Remote Sensing*, 12(8), 1257.

- Patil, V. H., & Franken, F. H. (2021). Visualization of statistically significant correlation coefficients from a correlation matrix: A call for a change in practice. *Journal of Marketing Analytics*, 9(4), 286–297.
- Perez, H., & Tah, J. H. M. (2020). Improving the Accuracy of Convolutional Neural Networks by Identifying and Removing Outlier Images in Datasets Using t-SNE. *Mathematics*, 8(5), 662.
<https://doi.org/10.3390/math8050662>
- Pradipta, G. A., Wardoyo, R., Musdholifah, A., Sanjaya, I. N. H., & Ismail, M. (2021). *SMOTE for handling imbalanced data problem: A review*. 1–8.
- Priyanka, & Kumar, D. (2020). Decision tree classifier: A detailed survey. *International Journal of Information and Decision Sciences*, 12(3), 246–269.
- Pudjihartono, N., Fadason, T., Kempa-Liehr, A. W., & O'Sullivan, J. M. (2022). A review of feature selection methods for machine learning-based disease risk prediction. *Frontiers in Bioinformatics*, 2, 927312.
- Rout, M. (2021). Analysis and comparison of credit card fraud detection using machine learning. In *Artificial Intelligence and Machine Learning in Business Management* (pp. 81–93). CRC Press.
- Schonlau, M., & Zou, R. Y. (2020). The random forest algorithm for statistical learning. *The Stata Journal*, 20(1), 3–29.
- Stavins, J. (2020). Credit card debt and consumer payment choice: What can we learn from credit bureau data? *Journal of Financial Services Research*, 58(1), 59–90.
- Tang, Z. (2023). Assessing the feasibility of machine learning-based modelling and prediction of credit fraud outcomes using hyperparameter tuning. *Advances in Computer, Signals and Systems*, 7(2), 84–92.
- Tanikella, U. (2020). *Credit Card Approval Verification Model*.

- Varun Kumar, K., Vijaya Kumar, V., Vijay Shankar, A., & Pratibha, K. (2020). Credit card fraud detection using machine learning algorithms. *International Journal of Engineering Research & Technology (IJERT)*, 9(07), 5–8.
- Vuttipittayamongkol, P., Elyan, E., & Petrovski, A. (2021). On the class overlap problem in imbalanced data classification. *Knowledge-Based Systems*, 212, 106631.
<https://doi.org/10.1016/j.knosys.2020.106631>
- Wang, C., Deng, C., & Wang, S. (2020). Imbalance-XGBoost: Leveraging weighted and focal losses for binary label-imbalanced classification with XGBoost. *Pattern Recognition Letters*, 136, 190–197.
- Wang, H., Bellotti, A. G., Qu, R., & Bai, R. (2024). *Discrete-time Survival Models with Neural Networks for Age-Period-Cohort Analysis of Credit Risk* [Preprint]. Computer Science and Mathematics.
<https://doi.org/10.20944/preprints202401.0040.v1>
- Wang, T., Bian, Y., Zhang, Y., & Hou, X. (2023). Classification of earthquakes, explosions and mining-induced earthquakes based on XGBoost algorithm. *Computers & Geosciences*, 170, 105242.
- Wang, Y., Liu, Y., Zhao, J., & Zhang, Q. (2023). Low-Complexity Fast CU Classification Decision Method Based on LGBM Classifier. *Electronics*, 12(11), 2488.
- Wang, Y., Zhang, Y., Lu, Y., & Yu, X. (2020). A Comparative Assessment of Credit Risk Model Based on Machine Learning—A case study of bank loan data. *Procedia Computer Science*, 174, 141–149.
<https://doi.org/10.1016/j.procs.2020.06.069>
- Weng, C.-H., & Huang, C.-K. (2021). A Hybrid Machine Learning Model for Credit Approval. *Applied Artificial Intelligence*, 35(15), 1439–1465.
- Wu, C.-F., Huang, S.-C., Chiou, C.-C., & Wang, Y.-M. (2021). A predictive intelligence system of credit scoring based on deep multiple kernel learning. *Applied Soft Computing*, 111, 107668.

APPENDIX: Python Programming in Algorithm Building

The Python programming language was used to create the prediction model in this study. The complete input code and its result are presented below

Appendix A: Vintage Analysis

Vintage Analysis

```
In [ ]: 1 credit= record.copy()
```

```
In [ ]: 1
2 grouped = credit.groupby('ID')
3 #convert credit data to wide format which every ID is a row
4 pivot_tb = credit.pivot(index = 'ID', columns = 'MONTHS_BALANCE', values = 'STATUS')
5 pivot_tb['open_month'] = grouped['MONTHS_BALANCE'].min() # smallest value of MONTHS_BALANCE, is the month when loan was granted
6 pivot_tb['end_month'] = grouped['MONTHS_BALANCE'].max() # biggest value of MONTHS_BALANCE, might be observe over or cancelin
7 pivot_tb['ID'] = pivot_tb.index
8 pivot_tb = pivot_tb[['ID', 'open_month', 'end_month']]
9 pivot_tb['window'] = pivot_tb['end_month'] - pivot_tb['open_month'] # calculate observe window
10 pivot_tb.reset_index(drop = True, inplace = True)
11 credit = pd.merge(credit, pivot_tb, on = 'ID', how = 'left') # join calculated information
12 credit0 = credit.copy()
13 credit = credit[credit['window'] > 20] # delete users whose observe window less than 20
14 credit['status'] = np.where((credit['STATUS'] == '2') | (credit['STATUS'] == '3') | (credit['STATUS'] == '4') | (credit['STA
15 credit['status'] = credit['status'].astype(np.ints) # 1: overdue 0: not
16 credit['month_on_book'] = credit['MONTHS_BALANCE'] - credit['open_month'] # calculate month on book: how many months after o
17 credit.sort_values(by = ['ID', 'month_on_book'], inplace = True)
18
19 #denominator
20 denominator = pivot_tb.groupby(['open_month']).agg({'ID': ['count']}) # count how many users in every month the account was
21 denominator.reset_index(inplace = True)
22 denominator.columns = ['open_month', 'sta_sum']
23
24 #vintage table
25 vintage = credit.groupby(['open_month', 'month_on_book']).agg({'ID': ['count']})
26 vintage.reset_index(inplace = True)
27 vintage.columns = ['open_month', 'month_on_book', 'sta_sum']
28 vintage['due_count'] = np.nan
29 vintage = vintage[['open_month', 'month_on_book', 'due_count']] # delete aggerate column
30 vintage = pd.merge(vintage, denominator, on = ['open_month'], how = 'left') # join sta_sum colun to vintage table
31 vintage
```

```
Out[408]:
```

	open_month	month_on_book	due_count	sta_sum
0	-80	0	NaN	415
1	-80	1	NaN	415
2	-80	2	NaN	415
3	-80	3	NaN	415
4	-80	4	NaN	415
...
1655	-21	17	NaN	858
1656	-21	18	NaN	858
1657	-21	19	NaN	858
1658	-21	20	NaN	858
1659	-21	21	NaN	858

1660 rows x 4 columns

```

In [ ]: 1 for j in range(-60,1): # outer Loop: month in which account was opened
        2     ls = []
        3     for i in range(0,61): # inner Loop time after the credit card was granted
        4         due = list(credit[(credit['status'] == 1) & (credit['month_on_book'] == i) & (credit['open_month'] == j)]['ID']) # g
        5         ls.extend(due) # As time goes, add bad customers
        6         vintage.loc[(vintage['month_on_book'] == i) & (vintage['open_month'] == j), 'due_count'] = len(set(ls)) # calculate
        7
        8 vintage['sta_rate'] = vintage['due_count'] / vintage['sta_sum'] # calculate cumulative % of bad customers
        9 vintage

```

```

Out[409]:
  open_month  month_on_book  due_count  sta_sum  sta_rate
0          -60             0         0.0      415  0.000000
1          -60             1         0.0      415  0.000000
2          -60             2         0.0      415  0.000000
3          -60             3         0.0      415  0.000000
4          -60             4         0.0      415  0.000000
...
1655        -21             17        15.0      858  0.017483
1656        -21             18        15.0      858  0.017483
1657        -21             19        15.0      858  0.017483
1658        -21             20        15.0      858  0.017483
1659        -21             21        15.0      858  0.017483

```

1660 rows x 5 columns

```

In [ ]: 1 # Using pivot to convert Long data to wide data:
        2 # Vintage wide table
        3 vintage_wide = vintage.pivot(index = 'open_month',
        4                               columns = 'month_on_book',
        5                               values = 'sta_rate')
        6 vintage_wide

```

```

Out[410]:
  month_on_book  0      1      2      3      4      5      6      7      8      9  ...  51  52  53  54
open_month
-60  0.00000  0.00000  0.00000  0.00000  0.00000  0.002410  0.002410  0.007229  0.007229  0.008639  ...  0.028918  0.028918  0.028918  0.028918
-59  0.00000  0.00000  0.00000  0.004926  0.004926  0.004926  0.007389  0.007389  0.007389  0.012315  ...  0.029557  0.029557  0.029557  0.032020
-58  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.004545  0.008818  0.008818  ...  0.025000  0.025000  0.025000  0.025000
-57  0.00000  0.00000  0.00000  0.00000  0.002500  0.002500  0.002500  0.007500  0.007500  0.007500  ...  0.025000  0.025000  0.025000  0.025000
-56  0.00000  0.00000  0.004255  0.004255  0.008511  0.010838  0.010838  0.017021  0.017021  0.021277  ...  0.036170  0.036170  0.036170  0.036170
-55  0.00000  0.00000  0.002058  0.006173  0.012346  0.016481  0.022634  0.022634  0.022634  0.024891  ...  0.045267  0.045267  0.045267  0.045267
-54  0.00000  0.00000  0.00000  0.002137  0.004274  0.004274  0.008410  0.012821  0.012821  0.017094  ...  0.032051  0.032051  0.032051  0.032051
-53  0.00000  0.002028  0.002028  0.002028  0.004057  0.004057  0.008114  0.010142  0.010142  0.012170  ...  0.028398  0.028398  0.028398  NaN
-52  0.00000  0.00000  0.007233  0.009042  0.009042  0.009042  0.012658  0.012658  0.014467  0.018275  ...  0.023508  0.023508  NaN  NaN
-51  0.00000  0.001718  0.001718  0.003436  0.003436  0.008591  0.010309  0.013746  0.013746  0.015484  ...  0.020619  NaN  NaN  NaN

```

40 rows x 61 columns

```

In [ ]: 1 # plot vintage Line chart
        2 plt.rcParams['figure.facecolor'] = 'white'
        3 vintage0 = vintage_wide.replace(0,np.nan)
        4 lst = [i for i in range(0,61)]
        5 vintage_wide[lst].T.plot(legend = False, grid = True, title = 'Cumulative % of Bad Customers (> 60 Days Past Due)')
        6 #plt.axvline(30)
        7 #plt.axvline(25)
        8 #plt.axvline(20)
        9 plt.xlabel('Months on Books')
       10 plt.ylabel('Cumulative % > 60 Days Past Due')
       11 plt.show()

```

<Figure size 640x480 with 1 Axes>

Appendix B: Feature Engineering & Exploratory Data Analysis

Feature Engineering

Response Variable

```
In [ ]: 1 # find all users' account open month.
2 begin_month=pd.DataFrame(record.groupby(["ID"])[["MONTHS_BALANCE"]].agg(min))
3 begin_month=begin_month.rename(columns={'MONTHS_BALANCE':'begin_month'})
4 new_data=pd.merge(data,begin_month,how="left",on="ID") #merge to record data
```

Generally, users in risk should be in 3%, thus I choose users who overdue for more than 60 days as target risk users. Those samples are marked as '1', else are '0'.

```
In [ ]: 1 record['dep_value'] = None
2 record['dep_value'][record['STATUS'] == '2'] = 'Yes'
3 record['dep_value'][record['STATUS'] == '3'] = 'Yes'
4 record['dep_value'][record['STATUS'] == '4'] = 'Yes'
5 record['dep_value'][record['STATUS'] == '5'] = 'Yes'
```

```
In [ ]: 1 cpunt=record.groupby('ID').count()
2 cpunt['dep_value'][cpunt['dep_value'] > 0]='Yes'
3 cpunt['dep_value'][cpunt['dep_value'] == 0]='No'
4 cpunt = cpunt[['dep_value']]
5 new_data=pd.merge(new_data,cpunt,how='inner',on='ID')
6 new_data['target']=new_data['dep_value']
7 new_data.loc[new_data['target']=='Yes','target']=1
8 new_data.loc[new_data['target']=='No','target']=0
```

```
In [ ]: 1 new_data.to_csv('Creditcard.csv')
```

```
In [ ]: 1 sns.countplot(x=cpunt["dep_value"]),
2 cpunt['dep_value'].value_counts(normalize=True)
```

```
Out[302]: No      0.985495
Yes       0.014505
Name: dep_value, dtype: float64

<Figure size 640x480 with 1 Axes>
```

Features

- rename

```
In [ ]: 1 new_data.rename(columns={'CODE_GENDER':'Gender', 'FLAG_OWN_CAR':'Car', 'FLAG_OWN_REALTY':'Reality',
2                               'CNT_CHILDREN':'ChildNo', 'AMT_INCOME_TOTAL':'Income',
3                               'NAME_EDUCATION_TYPE':'EducationType', 'NAME_FAMILY_STATUS':'Marital_status',
4                               'NAME_HOUSING_TYPE':'HousingType', 'FLAG_EMAIL':'Email',
5                               'NAME_INCOME_TYPE':'IncomeType', 'FLAG_WORK_PHONE':'WorkPhone',
6                               'FLAG_PHONE':'Phone', 'CNT_FAM_MEMBERS':'FamilySize',
7                               'OCCUPATION_TYPE':'OccupationType'
8                               }, inplace=True)
9
```

Features

- rename

```
In [ ]: 1 new_data.rename(columns={'CODE_GENDER':'Gender', 'FLAG_OWN_CAR':'Car', 'FLAG_OWN_REALTY':'Reality',
2                                'CNT_CHILDREN':'ChildNo', 'AMT_INCOME_TOTAL':'Income',
3                                'NAME_EDUCATION_TYPE':'EducationType', 'NAME_FAMILY_STATUS':'Marital_status',
4                                'NAME_HOUSING_TYPE':'HousingType', 'FLAG_EMAIL':'Email',
5                                'NAME_INCOME_TYPE':'IncomeType', 'FLAG_WORK_PHONE':'WorkPhone',
6                                'FLAG_PHONE':'Phone', 'CNT_FAM_MEMBERS':'FamilySize',
7                                'OCCUPATION_TYPE':'OccupationType'
8                                }, inplace=True)
9
```

```
In [ ]: 1 new_data[['ID','ChildNo','Income','DAYS_BIRTH','DAYS_EMPLOYED','begin_month']].describe().T
```

```
Out[304]:
```

	count	mean	std	min	25%	50%	75%	max
ID	36457.0	5.078227e+08	41875.240788	5008804.0	5042028.0	5074614.0	5115306.0	5150487.0
ChildNo	36457.0	4.303152e-01	0.742367	0.0	0.0	0.0	1.0	19.0
Income	36457.0	1.868657e+05	101789.226482	27000.0	121500.0	157500.0	225000.0	1575000.0
DAYS_BIRTH	36457.0	-1.597517e+04	4200.549944	-25152.0	-19438.0	-15563.0	-12462.0	-7489.0
DAYS_EMPLOYED	36457.0	5.926294e+04	137651.334859	-15713.0	-3153.0	-1552.0	-408.0	365243.0
begin_month	36457.0	-2.616419e+01	16.501854	-80.0	-39.0	-24.0	-12.0	0.0

```
In [ ]: 1 new_data.isnull().sum()
```

```
Out[305]:
```

ID	0
Gender	0
Car	0
Reality	0
ChildNo	0
Income	0
IncomeType	0
EducationType	0
Marital_status	0
HousingType	0
DAYS_BIRTH	0
DAYS_EMPLOYED	0
FLAG_MOBIL	0
WorkPhone	0
Phone	0
Email	0
OccupationType	11323
FamilySize	0
begin_month	0
dep_value	0
target	0

dtype: int64

```
In [ ]: 1 total_missing = new_data.isnull().sum().sum()
2 total_values = len(new_data)
3 percentage_missing = round((total_missing / total_values) * 100, 2)
4
5 # Print the percentage
6 print("Percentage of missing values:", percentage_missing, "%")
```

Percentage of missing values: 31.06 %

```
In [ ]: 1 new_data.dropna()
        2 new_data = new_data.mask(new_data == 'NULL').dropna()
```

```
In [ ]: 1 new_data.shape
```

Out[308]: (25134, 21)

```
In [ ]: 1 # Select numerical columns (excluding object type)
        2 not_object = new_data.select_dtypes(exclude=object).columns
        3 # Calculate the number of rows and columns for subplots
        4 n_rows = len(not_object) // 4 + (len(not_object) % 4 > 0)
        5 n_cols = 4
        6
        7 # Create subplots
        8 fig, axes = plt.subplots(nrows=n_rows, ncols=n_cols, figsize=(15, 5 * n_rows))
        9
        10 # Flatten the axes if needed
        11 axes = axes.flatten()
        12
        13 # Loop through numerical columns and plot boxplots
        14 for idx, column in enumerate(not_object):
        15     new_data[column].boxplot(fontsize=15, ax=axes[idx])
        16     axes[idx].set_title(column) # Set title for each subplot
        17
        18 # Remove any unused subplots
        19 for idx in range(len(not_object), n_rows * n_cols):
        20     fig.delaxes(axes[idx])
        21
        22 # Adjust layout to prevent overlap
        23 plt.tight_layout()
```

<Figure size 1500x1500 with 11 Axes>

```
In [ ]: 1 ivtable=pd.DataFrame(new_data.columns,columns=['variable'])
        2 ivtable['IV']=None
        3 namelist = ['FLAG_MOBIL','begin_month','dep_value','target','ID']
        4
        5 for i in namelist:
        6     ivtable.drop(ivtable[ivtable['variable'] == i].index, inplace=True)
```

Binary Features

Binary Features

target

```
In [ ]: 1 # Example data
        2 counts = new_data["target"].value_counts()
        3 percentages = counts / len(data) * 100
        4
        5 # Set Seaborn style
        6 sns.set(style="whitegrid")
        7
        8 # Identify the index of the slice you want to explode
        9 explode_index = 0
        10
        11 # Create the pie chart with explosion
        12 plt.figure(figsize=(10, 8))
        13 colors = sns.color_palette("pastel")
        14 explode = [0.1 if i == explode_index else 0 for i in range(len(counts))]
        15 plt.pie(counts, labels=counts.index, autopct="%1.1f%%", startangle=90, colors=colors, explode=explode)
        16
        17 plt.title("Customers Risk Good(0) and Bad(1)", fontsize=16)
        18
        19 # Show the plot
        20 plt.show()
```

<Figure size 1000x800 with 1 Axes>

Gender

```
1 [ ]: 1 # Example data
2       counts = new_data["Gender"].value_counts()
3       percentages = counts / len(data) * 100
4
5       # Set Seaborn style
6       sns.set(style="whitegrid")
7
8       # Identify the index of the slice you want to explode
9       explode_index = 0
10
11      # Create the pie chart with explosion
12      plt.figure(figsize=(10, 8))
13      colors = sns.color_palette("pastel")
14      explode = [0.1 if i == explode_index else 0 for i in range(len(counts))]
15      plt.pie(counts, labels=counts.index, autopct="%1.1f%%", startangle=90, colors=colors, explode=explode)
16
17      plt.title("Customers Gender Female(F) and Male(M)", fontsize=16)
18
19      # Show the plot
20      plt.show()
```

<Figure size 1000x800 with 1 Axes>

```
1 [ ]: 1 new_data['Gender'] = new_data['Gender'].replace(['F','M'],[0,1])
2       print(new_data['Gender'].value_counts())
3       iv, data = calc_iv(new_data, 'Gender', 'target')
4       ivtable.loc[ivtable['variable']=='Gender', 'IV']=iv
5       data.head()
```

```
0    15630
1     9504
Name: Gender, dtype: int64
This variable's IV is: 0.02520350452745081
0    15630
1     9504
Name: Gender, dtype: int64
```

```
['313]:
```

	Variable	Value	All	Good	Bad	Share	Bad Rate	Distribution Good	Distribution Bad	WoE	IV
0	Gender	0	15630	15400	230	0.821887	0.014715	0.623179	0.545024	0.134005	0.010473
1	Gender	1	9504	9312	192	0.378133	0.020202	0.376821	0.454978	-0.188475	0.014730

—
—

Having a car or not

```
In [ ]: 1 # Example data
2       counts = new_data["Car"].value_counts()
3       percentages = counts / len(data) * 100
4
5       # Set Seaborn style
6       sns.set(style="whitegrid")
7
8       # Identify the index of the slice you want to explode
9       explode_index = 0
10
11      # Create the pie chart with explosion
12      plt.figure(figsize=(8, 8))
13      colors = sns.color_palette("pastel")
14      explode = [0.1 if i == explode_index else 0 for i in range(len(counts))]
15      plt.pie(counts, labels=counts.index, autopct="%1.1f%%", startangle=90, colors=colors, explode=explode)
16
17      plt.title("Customers have a Car without it", fontsize=16)
18
19      # Show the plot
20      plt.show()
```

<Figure size 800x800 with 1 Axes>

```
In [ ]: 1 new_data['Car'] = new_data['Car'].replace(['N','Y'],[0,1])
2 print(new_data['Car'].value_counts())
3 iv, data=calc_iv(new_data,'Car','target')
4 ivtable.loc[ivtable['variable']=='Car','IV']=iv
5 data.head()
```

```
0    14618
1    10516
Name: Car, dtype: int64
This variable's IV is: 4.54248124999671e-06
0    14618
1    10516
Name: Car, dtype: int64
```

```
Out[315]:
```

	Variable	Value	All	Good	Bad	Share	Bad Rate	Distribution Good	Distribution Bad	WoE	IV
0	Car	0	14618	14373	245	0.581803	0.018780	0.58182	0.580589	0.00181	0.000002
1	Car	1	10516	10339	177	0.418397	0.018831	0.41838	0.419431	-0.00251	0.000003

Having house reality or not

```
In [ ]: 1 # Example data
2 counts = new_data["Reality"].value_counts()
3 percentages = counts / len(data) * 100
4
5 # Set Seaborn style
6 sns.set(style="whitegrid")
7
8 # Identify the index of the slice you want to explode
9 explode_index = 0
10
11 # Create the pie chart with explosion
12 plt.figure(figsize=(10, 8))
13 colors = sns.color_palette("pastel")
14 explode = [0.1 if i == explode_index else 0 for i in range(len(counts))]
15 plt.pie(counts, labels=counts.index, autopct="%1.1f%%", startangle=90, colors=colors, explode=explode)
16
17 plt.title("Customers with and without Property", fontsize=16)
18
19 # Show the plot
20 plt.show()
```

<Figure size 1000x800 with 1 Axes>

```
In [ ]: 1 new_data['Reality'] = new_data['Reality'].replace(['N','Y'],[0,1])
2 print(new_data['Reality'].value_counts())
3 iv, data=calc_iv(new_data,'Reality','target')
4 ivtable.loc[ivtable['variable']=='Reality','IV']=iv
5 data.head()
```

```
1    16461
0     8673
Name: Reality, dtype: int64
This variable's IV is: 0.02744070350168343
1    16461
0     8673
Name: Reality, dtype: int64
```

```
Out[317]:
```

	Variable	Value	All	Good	Bad	Share	Bad Rate	Distribution Good	Distribution Bad	WoE	IV
0	Reality	0	8873	8494	179	0.34507	0.020839	0.34372	0.424171	-0.210309	0.016920
1	Reality	1	16461	16218	243	0.85493	0.014762	0.65628	0.575829	0.130777	0.010521

Having a phone or not

```
In [ ]: 1 # Example data
2 counts = new_data["Phone"].value_counts()
3 percentages = counts / len(data) * 100
4
5 # Set Seaborn style
6 sns.set(style="whitegrid")
7
8 # Identify the index of the slice you want to explode
9 explode_index = 0
10
11 # Create the pie chart with explosion
12 plt.figure(figsize=(10, 8))
13 colors = sns.color_palette("pastel")
14 explode = [0.1 if i == explode_index else 0 for i in range(len(counts))]
15 plt.pie(counts, labels=counts.index, autopct="%1.1f%%", startangle=90, colors=colors, explode=explode)
16
17 plt.title("Customers with and without phone", fontsize=16)
18
19 # Show the plot
20 plt.show()
```

<Figure size 1000x800 with 1 Axes>

```
In [ ]: 1 new_data['Phone']=new_data['Phone'].astype(str)
2 print(new_data['Phone'].value_counts(normalize=True,sort=False))
3 new_data.drop(new_data[new_data['Phone'] == 'nan' ].index, inplace=True)
4 iv, data=calc_iv(new_data,'Phone','target')
5 ivtable.loc[ivtable['variable']=='Phone','IV']=iv
6 data.head()
```

```
0    0.707209
1    0.292791
Name: Phone, dtype: float64
This variable's IV is: 0.0005480495762639297
0    17775
1     7359
Name: Phone, dtype: int64
```

```
rt[319]:
```

	Variable	Value	All	Good	Bad	Share	Bad Rate	Distribution Good	Distribution Bad	WoE	IV
0	Phone	0	17775	17481	294	0.707209	0.016540	0.707389	0.696682	0.015251	0.000163
1	Phone	1	7359	7231	128	0.292791	0.017394	0.292811	0.303318	-0.035937	0.000385

Having an email or not

```
In [ ]: 1 # Example data
2 counts = new_data["Email"].value_counts()
3 percentages = counts / len(data) * 100
4
5 # Set Seaborn style
6 sns.set(style="whitegrid")
7
8 # Identify the index of the slice you want to explode
9 explode_index = 0
10
11 # Create the pie chart with explosion
12 plt.figure(figsize=(10, 8))
13 colors = sns.color_palette("pastel")
14 explode = [0.1 if i == explode_index else 0 for i in range(len(counts))]
15 plt.pie(counts, labels=counts.index, autopct="%1.1f%%", startangle=90, colors=colors, explode=explode)
16
17 plt.title("Customers with and without email", fontsize=16)
18
19 # Show the plot
20 plt.show()
```

<Figure size 1000x800 with 1 Axes>

```
In [ ]: 1 print(new_data['Email'].value_counts(normalize=True, sort=False))
2 new_data['Email'] = new_data['Email'].astype(str)
3 iv, data = calc_iv(new_data, 'Email', 'target')
4 ivtable.loc[ivtable['variable']=='email', 'IV']=iv
5 data.head()
```

```
0    0.89934
1    0.10066
Name: Email, dtype: float64
This variable's IV is: 1.7343581493999816e-05
0    22604
1     2530
Name: Email, dtype: int64
```

```
Out[321]:
```

	Variable	Value	All	Good	Bad	Share	Bad Rate	Distribution Good	Distribution Bad	WoE	IV
0	Email	0	22604	22225	379	0.89934	0.016767	0.899361	0.898104	0.001398	0.000002
1	Email	1	2530	2487	43	0.10066	0.018998	0.100639	0.101896	-0.012407	0.000016

Having a Work Phone or not

```
In [ ]: 1 # Example data
2 counts = new_data["WorkPhone"].value_counts()
3 percentages = counts / len(data) * 100
4
5 # Set Seaborn style
6 sns.set(style="whitegrid")
7
8 # Identify the index of the slice you want to explode
9 explode_index = 0
10
11 # Create the pie chart with explosion
12 plt.figure(figsize=(10, 8))
13 colors = sns.color_palette("pastel")
14 explode = [0.1 if i == explode_index else 0 for i in range(len(counts))]
15 plt.pie(counts, labels=counts.index, autopct="%1.1f%%", startangle=90, colors=colors, explode=explode)
16
17 plt.title("Customers with and without work phone", fontsize=16)
18
19 # Show the plot
20 plt.show()
```

<Figure size 1000x800 with 1 Axes>

```
In [ ]: 1 new_data['WorkPhone']=new_data['WorkPhone'].astype(str)
2 iv, data = calc_iv(new_data, 'WorkPhone', 'target')
3 new_data.drop(new_data[new_data['WorkPhone'] == 'nan' ].index, inplace=True)
4 ivtable.loc[ivtable['variable']=='WorkPhone', 'IV']=iv
5 data.head()
```

This variable's IV is: 0.002042429795148461

0 18252

1 6882

Name: WorkPhone, dtype: int64

```
Out[323]:
```

	Variable	Value	All	Good	Bad	Share	Bad Rate	Distribution Good	Distribution Bad	WoE	IV
0	WorkPhone	0	18252	17954	298	0.726188	0.016327	0.72653	0.706161	0.028436	0.000579
1	WorkPhone	1	6882	6758	124	0.273812	0.018018	0.27347	0.293839	-0.071838	0.001483

Continuous Variables

Children Numbers

```
In [ ]: 1 new_data.columns
```

```
Out[324]: Index(['ID', 'Gender', 'Car', 'Reality', 'ChildNo', 'Income', 'IncomeType',  
              'EducationType', 'Marital_status', 'HousingType', 'DAYS_BIRTH',  
              'DAYS_EMPLOYED', 'FLAG_MOBIL', 'WorkPhone', 'Phone', 'Email',  
              'OccupationType', 'FamilySize', 'begin_month', 'dep_value', 'target'],  
              dtype='object')
```

```
In [ ]: 1 # Filter the data where 'Marital_status' is 'Single / not married' and 'NO_Childern' is >= 0  
2 condition_to_Single = (new_data['Marital_status'] == 'Single / not married') & (new_data['ChildNo'] > 0)  
3 filtered_data = new_data[condition_to_Single]  
4  
5 # Create a box plot  
6 plt.figure(figsize=(10, 6))  
7 sns.boxplot(x='FamilySize', y='ChildNo', data=filtered_data, palette='viridis')  
8 plt.title('Box Plot: Family Size vs. Number of Children for Singles')  
9 plt.xlabel('Family Size')  
10 plt.ylabel('Number of Children')  
11 plt.grid(True) # Add grid lines for better readability  
12 plt.show()
```

<Figure size 1000x600 with 1 Axes>

```
In [ ]: 1 # Replace 'NO_Childern' values with 0 for the filtered rows  
2 new_data.loc[condition_to_Single, 'Marital_status'] = 'Married'
```

```
In [ ]: 1 # Create a box plot using catplot  
2 plt.figure(figsize=(10, 6))  
3 sns.catplot(x='ChildNo', kind='box', data=new_data, height=6, aspect=2, palette="pastel")  
4 plt.title("Box Plot for NO Children", fontsize=16)  
5  
6 # Show the plot  
7 plt.show()
```

<Figure size 1000x600 with 0 Axes>

<Figure size 1200x600 with 1 Axes>

```
In [ ]: 1 # Assuming new_data is your DataFrame  
2 # Calculate the median value of the 'ChildNo' column  
3 median_value = np.median(new_data['ChildNo'])  
4  
5 # Calculate the interquartile range (IQR) for the 'ChildNo' column  
6 Q1 = new_data['ChildNo'].quantile(0.25)  
7 Q3 = new_data['ChildNo'].quantile(0.75)  
8 IQR = Q3 - Q1  
9  
10 # Define the threshold and bounds for outliers  
11 threshold = 1.5  
12 lower_bound = Q1 - threshold * IQR  
13 upper_bound = Q3 + threshold * IQR  
14  
15 # Replace outliers with the median value for the 'ChildNo' column  
16 new_data['ChildNo'] = np.where(  
17     (new_data['ChildNo'] < lower_bound) | (new_data['ChildNo'] > upper_bound),  
18     median_value,  
19     new_data['ChildNo'])  
20 )
```

```
In [ ]: 1 new_data['ChildNoegp'] = new_data['ChildNo']  
2 new_data['ChildNoegp'] = new_data['ChildNoegp'].astype(object)  
3 new_data.loc[new_data['ChildNoegp'] >= 2, 'ChildNoegp'] = '2More'  
4 iv, data = calc_iv(new_data, 'ChildNoegp', 'target')  
5 ivtable.loc[ivtable['variable'] == 'ChildNo', 'IV'] = iv  
6 data.head()
```

This variable's IV is: 0.0037129193760631765

```
0.0    16301  
1.0     6118  
2More   2715  
Name: ChildNoegp, dtype: int64
```

```
ut[329]:
```

	Variable	Value	All	Good	Bad	Share	Bad Rate	Distribution Good	Distribution Bad	WoE	IV
0	ChildNoegp	0.0	16301	16016	285	0.848564	0.017484	0.848106	0.875355	-0.041185	0.001122
1	ChildNoegp	1.0	6118	6021	97	0.243415	0.015855	0.243847	0.229858	0.058259	0.000803
2	ChildNoegp	2More	2715	2675	40	0.108021	0.014733	0.108247	0.094787	0.132786	0.001787

Annual Income

bins the data based on sample quantiles

```
In [ ]: 1 new_data['Income']=new_data['Income'].astype(object)
2 new_data['Income'] = new_data['Income']/10000
3 print(new_data['Income'].value_counts(bins=10,sort=False))
4 plt.figure(figsize=(10, 6))
5 plt.hist(new_data['Income'], bins=50, color='skyblue', edgecolor='black')
6 plt.title('Total_income Distribution')
7 plt.xlabel('Total_income')
8 plt.ylabel('Frequency')
9 plt.grid(axis='y', linestyle='--', alpha=0.7)
10 plt.show()
```

```
(2.544, 18.18]    14663
(18.18, 33.66]   8464
(33.66, 49.14]   1637
(49.14, 64.62]   175
(64.62, 80.1]    124
(80.1, 95.58]    50
(95.58, 111.06]  4
(111.06, 126.54] 3
(126.54, 142.02] 6
(142.02, 157.5]  8
Name: Income, dtype: int64
```

<Figure size 1000x600 with 1 Axes>

```
In [ ]: 1 plt.figure(figsize=(10, 6))
2 sns.catplot(x='Income', kind='box', data=new_data, height=6, aspect=2, palette="pastel")
3 plt.title('Box Plot for total income', fontsize=16)
4
5 # Show the plot
6 plt.show()
```

<Figure size 1000x600 with 0 Axes>

<Figure size 1200x600 with 1 Axes>

```
In [ ]: 1 # Assuming new_data is your DataFrame
2 # Calculate the median value of the 'Income' column
3 median_value = np.median(new_data['Income'])
4
5 # Calculate the interquartile range (IQR) for the 'Income' column
6 Q1 = new_data['Income'].quantile(0.25)
7 Q3 = new_data['Income'].quantile(0.75)
8 IQR = Q3 - Q1
9
10 # Define the threshold and bounds for outliers
11 threshold = 1.5
12 lower_bound = Q1 - threshold * IQR
13 upper_bound = Q3 + threshold * IQR
14
15 # Replace outliers with the median value for the 'Income' column
16 new_data['Income'] = np.where(
17     (new_data['Income'] < lower_bound) | (new_data['Income'] > upper_bound),
18     median_value,
19     new_data['Income']
20 )
```

```
In [ ]: 1 plt.figure(figsize=(10, 6))
2 plt.hist(new_data['Income'], bins=50, color='skyblue', edgecolor='black')
3 plt.title('Total_income Distribution')
4 plt.xlabel('Total_income')
5 plt.ylabel('Frequency')
6 plt.grid(axis='y', linestyle='--', alpha=0.7)
7 plt.show()
```

<Figure size 1000x600 with 1 Axes>

```
In [ ]: 1 new_data = get_category(new_data,'Income',3, ["low","medium", "high"])
2 iv, data = calc_iv(new_data,'gp_Income','target')
3 ivtable.loc[ivtable['variable']=='Income','IV'] = iv
4 data.head()
```

This variable's IV is: 0.0015948343029934082
medium 12207
low 8996
high 3931
Name: gp_Income, dtype: int64

```
Out[334]:
```

	Variable	Value	All	Good	Bad	Share	Bad Rate	Distribution Good	Distribution Bad	WoE	IV
0	gp_Income	high	3931	3850	72	0.156402	0.018316	0.156159	0.170616	-0.088542	0.001280
1	gp_Income	low	8996	8849	147	0.357922	0.016341	0.358085	0.348341	0.027588	0.000269
2	gp_Income	medium	12207	12004	203	0.485677	0.016830	0.485756	0.481043	0.009750	0.000046

Age

```
In [ ]: 1 new_data['Age']=(new_data['DAYS_BIRTH'])//365
2 plt.figure(figsize=(10, 6))
3 plt.hist(new_data['Age'], bins=20, color='skyblue', edgecolor='black')
4 plt.title('Age Distribution')
5 plt.xlabel('Age')
6 plt.ylabel('Frequency')
7 plt.grid(axis='y', linestyle='--', alpha=0.7)
8 plt.show()
```

<Figure size 1000x600 with 1 Axes>

```
In [ ]: 1 sns.set(style="whitegrid")
2
3 # Create a box plot using catplot
4 plt.figure(figsize=(10, 6))
5 sns.catplot(x='Age', kind='box', data=new_data, height=6, aspect=2, palette="pastel")
6 plt.title("Box Plot for Age", fontsize=16)
7
8 # Show the plot
9 plt.show()
```

<Figure size 1000x600 with 0 Axes>

<Figure size 1200x600 with 1 Axes>

```
In [ ]: 1 new_data = get_category(new_data,'Age',5, ["lowest","low","medium","high","highest"])
2 iv, data = calc_iv(new_data,'gp_Age','target')
3 ivtable.loc[ivtable['variable']=='DAYS_BIRTH','IV'] = iv
4 data.head()
```

This variable's IV is: 0.06593513858884348
medium 7916
low 7806
high 4414
lowest 4005
highest 993
Name: gp_Age, dtype: int64

```
In [ ]: 1 new_data = get_category(new_data,'Age',5, ["lowest","low","medium","high","highest"])
2 iv, data = calc_iv(new_data,'gp_Age','target')
3 ivtable.loc[ivtable['variable']=='DAYS_BIRTH','IV'] = iv
4 data.head()
```

This variable's IV is: 0.06593513858884348
medium 7916
low 7806
high 4414
lowest 4005
highest 993
Name: gp_Age, dtype: int64

```
Out[337]:
```

	Variable	Value	All	Good	Bad	Share	Bad Rate	Distribution Good	Distribution Bad	WoE	IV
0	gp_Age	high	4414	4323	91	0.175619	0.020616	0.174935	0.215640	-0.209194	0.008515
1	gp_Age	highest	993	989	4	0.039508	0.004028	0.040021	0.009479	1.440381	0.043992
2	gp_Age	low	7806	7686	120	0.310675	0.015373	0.311023	0.284380	0.089625	0.002390
3	gp_Age	lowest	4005	3921	84	0.159346	0.020974	0.158868	0.199052	-0.226754	0.009157
4	gp_Age	medium	7916	7793	123	0.314952	0.015538	0.315353	0.291489	0.078758	0.001881

Working Years

```
In [ ]: 1 # Calculate work experience in years
2 new_data['worktm'] = -(new_data['DAYS_EMPLOYED']) // 365
3 new_data.loc[new_data['worktm'] < 0, 'worktm'] = np.nan
4 new_data['worktm'].fillna(new_data['worktm'].mean(), inplace=True)
5
6 # Plotting the histogram
7 plt.figure(figsize=(10, 6))
8 plt.hist(new_data['worktm'], bins=20, color='skyblue', edgecolor='black')
9 plt.title('Experience Year Distribution')
10 plt.xlabel('Experience Year')
11 plt.ylabel('Frequency')
12 plt.grid(axis='y', linestyle='--', alpha=0.7)
13 plt.show()
14
```

<Figure size 1000x600 with 1 Axes>

```
In [ ]: 1 sns.set(style="whitegrid")
2
3 # Create a box plot using catplot
4 plt.figure(figsize=(10, 6))
5 sns.catplot(x='worktm', kind='box', data=new_data, height=6, aspect=2, palette="pastel")
6 plt.title("Box Plot for Age", fontsize=16)
7
8 # Show the plot
9 plt.show()
```

<Figure size 1000x600 with 0 Axes>

<Figure size 1200x600 with 1 Axes>

```
In [ ]: 1 # Assuming new_data is your DataFrame
2 # Calculate the median value of the 'worktm' column
3 median_value = np.median(new_data['worktm'])
4
5 # Calculate the interquartile range (IQR) for the 'worktm' column
6 Q1 = new_data['worktm'].quantile(0.25)
7 Q3 = new_data['worktm'].quantile(0.75)
8 IQR = Q3 - Q1
9
10 # Define the threshold and bounds for outliers
11 threshold = 1.5
12 lower_bound = Q1 - threshold * IQR
13 upper_bound = Q3 + threshold * IQR
14
15 # Replace outliers with the median value for the 'worktm' column
16 new_data['worktm'] = np.where(
17     (new_data['worktm'] < lower_bound) | (new_data['worktm'] > upper_bound),
18     median_value,
19     new_data['worktm']
20 )
```

```
In [ ]: 1 new_data = get_category(new_data, 'worktm', 5, ["lowest", "low", "medium", "high", "highest"])
2 iv, data = calc_iv(new_data, 'gp_worktm', 'target')
3 ivtable.loc[ivtable['variable']=='DAYS_EMPLOYED', 'IV']=iv
4 data.head()
```

This variable's IV is: 0.09818376914551227

```
lowest    9533
low       8629
medium    4094
high      1983
highest    895
Name: gp_worktm, dtype: int64
```

Out[342]:

	Variable	Value	All	Good	Bad	Share	Bad Rate	Distribution Good	Distribution Bad	WoE	IV
0	gp_worktm	high	1983	1984	19	0.078897	0.009581	0.079476	0.045024	0.588261	0.019578
1	gp_worktm	highest	895	883	12	0.035609	0.013408	0.035732	0.028436	0.228380	0.001886
2	gp_worktm	low	8629	8524	105	0.343320	0.012168	0.344934	0.248815	0.328642	0.031398
3	gp_worktm	lowest	9533	9314	219	0.379287	0.022973	0.378902	0.518657	-0.319837	0.045435
4	gp_worktm	medium	4094	4027	67	0.162887	0.016365	0.162957	0.158768	0.028045	0.000109

Family Size

```
In [ ]: 1 # Create a box plot using catplot
2 plt.figure(figsize=(10, 6))
3 sns.catplot(x='FamilySize', kind='box', data=new_data, height=6, aspect=2, palette="pastel")
4 plt.title("Box Plot for Family size", fontsize=16)
5
6 # Show the plot
7 plt.show()
```

<Figure size 1000x600 with 0 Axes>

<Figure size 1200x600 with 1 Axes>

```
In [ ]: 1 # Assuming new_data is your DataFrame
2 # Calculate the median value of the 'FamilySize' column
3 median_value = np.median(new_data['FamilySize'])
4
5 # Calculate the interquartile range (IQR) for the 'FamilySize' column
6 Q1 = new_data['FamilySize'].quantile(0.25)
7 Q3 = new_data['FamilySize'].quantile(0.75)
8 IQR = Q3 - Q1
9
10 # Define the threshold and bounds for outliers
11 threshold = 1.5
12 lower_bound = Q1 - threshold * IQR
13 upper_bound = Q3 + threshold * IQR
14
15 # Replace outliers with the median value for the 'FamilySize' column
16 new_data['FamilySize'] = np.where(
17     (new_data['FamilySize'] < lower_bound) | (new_data['FamilySize'] > upper_bound),
18     median_value,
19     new_data['FamilySize']
20 )
```

```
In [ ]: 1 new_data['FamilySize'].value_counts(sort=False)
```

```
Out[345]: 2.0    13079
1.0     4263
3.0     5216
4.0     2576
Name: FamilySize, dtype: int64
```

```
In [ ]: 1 new_data['famsizegp']=new_data['FamilySize']
2 new_data['famsizegp']=new_data['famsizegp'].astype(object)
3 new_data.loc[new_data['famsizegp']>=3,'famsizegp']='3more'
4 iv, data=calc_iv(new_data,'famsizegp','target')
5 ivtable.loc[ivtable['variable']=='FamilySize','IV']=iv
6 data.head()
```

This variable's IV is: 0.008125750039075303

```
2.0    13079
3more   7792
1.0     4263
Name: famsizegp, dtype: int64
```

```
In [ ]: 1 new_data['famsizegp']=new_data['FamilySize']
2 new_data['famsizegp']=new_data['famsizegp'].astype(object)
3 new_data.loc[new_data['famsizegp']>=3,'famsizegp']='3more'
4 iv, data=calc_iv(new_data,'famsizegp','target')
5 ivtable.loc[ivtable['variable']=='FamilySize','IV']=iv
6 data.head()
```

This variable's IV is: 0.008125750039075303

```
2.0    13079
3more   7792
1.0     4263
Name: famsizegp, dtype: int64
```

```
Out[346]:
```

	Variable	Value	All	Good	Bad	Share	Bad Rate	Distribution Good	Distribution Bad	WoE	IV
0	famsizegp	1.0	4263	4179	84	0.189811	0.019704	0.189108	0.199052	-0.183028	0.004882
1	famsizegp	2.0	13079	12850	220	0.520371	0.016821	0.520354	0.521327	-0.001887	0.000002
2	famsizegp	3more	7792	7874	118	0.310018	0.015144	0.310537	0.279821	0.104870	0.003242

Categorical Features

Income Type

```
In [ ]: 1 mean_income_by_IncomeType = new_data.groupby('IncomeType')['Income'].mean()
2 sorted_income = mean_income_by_IncomeType.sort_values(ascending=False)
3
4 plt.figure(figsize=(10, 6))
5
6 # Use sorted series for both x and y axes
7 sns.barplot(
8     x=sorted_income.index,
9     y=sorted_income.values,
10    color='skyblue'
11 )
12
13 plt.xlabel('Income Type')
14 plt.ylabel('Mean Total Income')
15 plt.title('Mean Total Income by Income Type')
16 plt.xticks(rotation=90)
17 plt.show()
18
```

<Figure size 1000x600 with 1 Axes>

```
In [ ]: 1 print(new_data['IncomeType'].value_counts(sort=False))
2 print(new_data['IncomeType'].value_counts(normalize=True,sort=False))
3 new_data.loc[new_data['IncomeType']=='Pensioner','IncomeType']='State servant'
4 new_data.loc[new_data['IncomeType']=='Student','IncomeType']='State servant'
5 iv, data=calc_iv(new_data,'IncomeType','target')
6 ivtable.loc[ivtable['variable']=='IncomeType','IV']=iv
7 data.head()
```

```
Working          15622
Commercial associate    7052
State servant         2437
Student              10
Pensioner            13
Name: IncomeType, dtype: int64
Working          0.621549
Commercial associate    0.280576
State servant         0.096960
Student            0.000398
Pensioner          0.000517
Name: IncomeType, dtype: float64
This variable's IV is: 5.159303327851404e-05
Working          15622
Commercial associate    7052
State servant         2460
Name: IncomeType, dtype: int64
```

```
Out[349]:
```

	Variable	Value	All	Good	Bad	Share	Bad Rate	Distribution Good	Distribution Bad	WoE	IV
0	IncomeType	Commercial associate	7052	6933	119	0.280576	0.016875	0.280552	0.281991	-0.005115	0.000007
1	IncomeType	State servant	2460	2418	42	0.097875	0.017073	0.097847	0.099526	-0.017013	0.000029
2	IncomeType	Working	15622	15381	261	0.621549	0.016707	0.621601	0.618483	0.005028	0.000016

Occupation Type

```
In [ ]: 1 mean_income_by_OccupationType = new_data.groupby('OccupationType')['Income'].mean()
2 sorted_income = mean_income_by_OccupationType.sort_values(ascending=False)
3
4 plt.figure(figsize=(10, 6))
5
6 # Use sorted series for both x and y axes
7 sns.barplot(
8     x=sorted_income.index,
9     y=sorted_income.values,
10    color='skyblue'
11 )
12
13 plt.xlabel('OccupationType')
14 plt.ylabel('Mean Total Income')
15 plt.title('Mean Total Income by OccupationType')
16 plt.xticks(rotation=90)
17 plt.show()
18
```

<Figure size 1000x600 with 1 Axes>

```
In [ ]: 1 new_data.loc[(new_data['OccupationType']=='Cleaning staff') | (new_data['OccupationType']=='Cooking staff') | (new_data['OccupationType']=='Accountants') | (new_data['OccupationType']=='Core staff') | (new_data['OccupationType']=='Managers') | (new_data['OccupationType']=='High skill tech staff') | (new_data['OccupationType']=='Other staff')]
2 new_data.loc[(new_data['OccupationType']=='Accountants') | (new_data['OccupationType']=='Core staff') | (new_data['OccupationType']=='Managers') | (new_data['OccupationType']=='High skill tech staff') | (new_data['OccupationType']=='Other staff')]
3 new_data.loc[(new_data['OccupationType']=='Managers') | (new_data['OccupationType']=='High skill tech staff') | (new_data['OccupationType']=='Other staff')]
4 print(new_data['OccupationType'].value_counts())
5 iv, data=calc_iv(new_data, 'OccupationType', 'target')
6 ivtable.loc[ivtable['variable']=='OccupationType', 'IV']=iv
7 data.head()
```

```
Laborwk      10496
officewk      10183
hightecwk     4455
Name: OccupationType, dtype: int64
This variable's IV is: 0.004820472062853304
Laborwk      10496
officewk      10183
hightecwk     4455
Name: OccupationType, dtype: int64
```

```
ut[351]:
```

	Variable	Value	All	Good	Bad	Share	Bad Rate	Distribution Good	Distribution Bad	WoE	IV
0	OccupationType	Laborwk	10496	10311	185	0.417802	0.017828	0.417247	0.438389	-0.049428	0.001045
1	OccupationType	hightecwk	4455	4375	80	0.177250	0.017957	0.177039	0.189573	-0.088404	0.000857
2	OccupationType	officewk	10183	10026	157	0.405148	0.015418	0.405714	0.372038	0.088652	0.002918

House Type

```
In [ ]: 1 import matplotlib.pyplot as plt
2
3 # Assuming new_data is your DataFrame
4 HousingType_counts = new_data['HousingType'].value_counts(normalize=True, sort=False)
5
6 # Create a bar chart
7 plt.figure(figsize=(10, 6)) # Set the figure size
8 bars = HousingType_counts.plot(kind='bar', color='skyblue') # Create the bar chart
9
10 # Add Labels on the bars
11 for bar in bars.patches:
12     plt.annotate(format(bar.get_height(), '.2%'),
13                 (bar.get_x() + bar.get_width() / 2,
14                  bar.get_height()),
15                 ha='center',
16                 va='center',
17                 size=12,
18                 xytext=(0, 5),
19                 textcoords='offset points')
20
21 plt.title('Distribution of Housing Type') # Set the title
22 plt.xlabel('Housing Type') # Set the x-axis label
23 plt.ylabel('Normalized Frequency') # Set the y-axis label
24 plt.xticks(rotation=45) # Rotate x-axis labels for better visibility if needed
25 plt.grid(axis='y') # Add gridlines on the y-axis
26 plt.tight_layout() # Adjust layout to ensure labels and titles are displayed properly
27 plt.show() # Display the plot
28
```

<Figure size 1000x600 with 1 Axes>

```
In [ ]: 1 iv, data=calc_iv(new_data, 'HousingType', 'target')
2 ivtable.loc[ivtable['variable']=='HousingType', 'IV']=iv
3 data.head()
```

This variable's IV is: 0.0073275026880227365

House / apartment	22102
With parents	1430
Municipal apartment	812
Rented apartment	439
Office apartment	199
Co-op apartment	152

Name: HousingType, dtype: int64

```
Out[353]:
```

	Variable	Value	All	Good	Bad	Share	Bad Rate	Distribution Good	Distribution Bad	WoE	IV
0	HousingType	Co-op apartment	152	149	3	0.008048	0.019737	0.008029	0.007109	-0.164705	0.000178
1	HousingType	House / apartment	22102	21738	364	0.879367	0.016469	0.879854	0.882559	0.019824	0.000335
2	HousingType	Municipal apartment	812	793	19	0.032307	0.023399	0.032090	0.046024	-0.338655	0.004380
3	HousingType	Office apartment	199	194	5	0.007918	0.025126	0.007850	0.011848	-0.411619	0.001646
4	HousingType	Rented apartment	439	433	6	0.017466	0.013667	0.017522	0.014218	0.208939	0.000690

Education

Education

```
In [ ]: 1 mean_income_by_EducationType = new_data.groupby('EducationType')['Income'].mean()
2 sorted_income = mean_income_by_EducationType.sort_values(ascending=False)
3
4 plt.figure(figsize=(10, 6))
5
6 # Use sorted series for both x and y axes
7 sns.barplot(
8     x=sorted_income.index,
9     y=sorted_income.values,
10    color='skyblue'
11 )
12
13 plt.xlabel('Education Type')
14 plt.ylabel('Mean Total Income')
15 plt.title('Mean Total Income by Education Type')
16 plt.xticks(rotation=90)
17 plt.show()
```

<Figure size 1000x600 with 1 Axes>

```
In [ ]: 1 new_data.loc[new_data['EducationType']=='Academic degree','EducationType']='Higher education'
2 iv, data=calc_iv(new_data,'EducationType','target')
3 ivtable.loc[ivtable['variable']=='EducationType','IV']=iv
4 data.head()
```

This variable's IV is: 0.010361794017679489

Secondary / secondary special 16808

Higher education 7146

Incomplete higher 993

Lower secondary 187

Name: EducationType, dtype: int64

```
Out[356]:
```

	Variable	Value	All	Good	Bad	Share	Bad Rate	Distribution Good	Distribution Bad	WoE	IV
0	EducationType	Higher education	7146	7018	128	0.284316	0.017912	0.283992	0.303318	-0.065836	0.001272
1	EducationType	Incomplete higher	993	972	21	0.039508	0.021148	0.039333	0.049783	-0.235206	0.002453
2	EducationType	Lower secondary	187	181	6	0.007440	0.032088	0.007324	0.014218	-0.863301	0.004573
3	EducationType	Secondary / secondary special	16808	16541	267	0.868736	0.015885	0.869351	0.832701	0.058310	0.002084

Martial status

```
In [ ]: 1 mean_income_by_Martial_status = new_data.groupby('Martial_status')['Income'].mean()
2 sorted_income = mean_income_by_Martial_status.sort_values(ascending=False)
3
4 plt.figure(figsize=(10, 6))
5
6 # Use sorted series for both x and y axes
7 sns.barplot(
8     x=sorted_income.index,
9     y=sorted_income.values,
10    color='skyblue'
11 )
12
13 plt.xlabel('Martial_status')
14 plt.ylabel('Mean Total Income')
15 plt.title('Mean Total Income by Martial_status')
16 plt.xticks(rotation=90)
17 plt.show()
```

<Figure size 1000x600 with 1 Axes>

```
In [ ]: 1 iv, data=calc_iv(new_data,'Martial_status','target')
2 ivtable.loc[ivtable['variable']=='Martial_status','IV']=iv
3 data.head()
```

This variable's IV is: 0.032004120494654835

```
Married      18108
Single / not married  2846
Civil marriage  2133
Separated    1467
Widow        580
Name: Martial_status, dtype: int64
```

```
Out[358]:
```

	Variable	Value	All	Good	Bad	Share	Bad Rate	Distribution Good	Distribution Bad	WoE	IV
0	Martial_status	Civil marriage	2133	2101	32	0.084865	0.015002	0.085019	0.075829	0.114394	0.001051
1	Martial_status	Married	18108	17813	295	0.720458	0.016291	0.720824	0.699052	0.030670	0.000668
2	Martial_status	Separated	1467	1452	15	0.058367	0.010225	0.058757	0.035545	0.502808	0.011668
3	Martial_status	Single / not married	2846	2781	65	0.113233	0.022839	0.112536	0.154028	-0.313880	0.013023
4	Martial_status	Widow	580	565	15	0.023076	0.025882	0.022883	0.035545	-0.441283	0.005598

Appendix C: Data Encoding

Convert dummy

```
In [ ]: 1 from sklearn.preprocessing import OneHotEncoder
2 import pandas as pd
3
4 # Convert columns to string data type if needed
5 columns_to_convert = ['famsizegp', 'gp_Income', 'gp_Age', 'gp_worktm', 'IncomeType',
6                       'OccupationType', 'HousingType', 'EducationType', 'Marital_status']
7
8 for column in columns_to_convert:
9     new_data[column] = new_data[column].astype(str)
10
11 # Create the encoder
12 encoder = OneHotEncoder()
13
14 # Fit and transform the data
15 new_data_encoded = encoder.fit_transform(new_data[columns_to_convert])
16
17 # Convert the encoded data to a DataFrame
18 encoded_df = pd.DataFrame(new_data_encoded.toarray(), columns=encoder.get_feature_names_out(columns_to_convert))
19
20 # Reset the index for joining
21 data_reset = new_data.reset_index(drop=True)
22 encoded_df_reset = encoded_df.reset_index(drop=True)
23
24 # Merge with main DataFrame based on a common key, e.g., index
25 New_df = pd.concat([data_reset, encoded_df_reset], axis=1)
26
```

Appendix D: IV & WOE

IV & WOE

```
In [ ]: 1 ivtable=ivtable.sort_values(by='IV',ascending=False)
2 ivtable.loc[ivtable['variable']=='DAYS_BIRTH','variable']='agegp'
3 ivtable.loc[ivtable['variable']=='DAYS_EMPLOYED','variable']='worktmgp'
4 ivtable.loc[ivtable['variable']=='inc','variable']='incgp'
5 ivtable
6
```

```
Out[363]:
```

	variable	IV
11	worktmgp	0.098184
10	agegp	0.085935
8	Marital_status	0.032004
3	Reality	0.027441
1	Gender	0.025204
7	EducationType	0.010362
17	FamilySize	0.008126
9	HousingType	0.007328
16	OccupationType	0.00482
4	ChildNo	0.003713
13	WorkPhone	0.002042
5	Income	0.001595
14	Phone	0.000548
6	IncomeType	0.000052
2	Car	0.000005
15	Email	None

Appendix E: Correlation Matrix

Correlation Matrix

```
In [ ]: 1 # Assuming new_data is your dataframe containing all the features
2 # Filter new_data to only include the specified columns
3 selected_features = [
4     'Income', 'Age', 'FamilySize', 'worktm',
5     'WorkPhone', 'Phone', 'Email',
6     'ChildNo', 'Car', 'Gender', 'Reality'
7 ]
8 filtered_data = new_data[selected_features]
9
10 # Compute correlation matrix for selected features
11 correlation_matrix = filtered_data.corr()
12
13 # Plot the correlation matrix as a heatmap
14 plt.figure(figsize=(15, 12))
15 sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
16 plt.title('Correlation Matrix')
17 plt.show()
```

<Figure size 1500x1200 with 2 Axes>

```
In [ ]: 1 # Find highly correlated features
2 high_corr = set() # Set to store highly correlated features
3 for i in range(len(correlation_matrix.columns)):
4     for j in range(i):
5         if abs(correlation_matrix.iloc[i, j]) > 0.8: # Change the correlation threshold as needed
6             colname = correlation_matrix.columns[i]
7             high_corr.add(colname)
8
9 # Remove highly correlated features
10 df_dropped = new_data.drop(high_corr, axis=1)
11 num_deleted = len(new_data.columns) - len(df_dropped.columns)
12 deleted_features = list(set(new_data.columns) - set(df_dropped.columns))
13
14 # Print the number of deleted features and their names
15 print(f"Number of features deleted: {num_deleted}")
16 print(f"Deleted feature names: {'', '.join(deleted_features)}")
17
```

Number of features deleted: 1
Deleted feature names: ChildNo

```
In [ ]: 1 new_data.columns
```

```
Out[361]: Index(['ID', 'Gender', 'Car', 'Reality', 'ChildNo', 'Income', 'IncomeType',
'EducationType', 'Marital_status', 'HousingType', 'DAYS_BIRTH',
'DAYS_EMPLOYED', 'FLAG_MOBIL', 'WorkPhone', 'Phone', 'Email',
'OccupationType', 'FamilySize', 'begin_month', 'dep_value', 'target',
'ChildNoegp', 'gp_Income', 'Age', 'gp_Age', 'worktm', 'gp_worktm',
'famsizegp'],
dtype='object')
```

Appendix F: Feature Scaling

Synthetic Minority Over-Sampling Technique(SMOTE)

```
In [ ]: 1 Y = Y.astype(int)
```

```
In [ ]: 1 X_SMOTE, Y_SMOTE = SMOTE().fit_resample(X, Y)
2
3 # Convert the numpy array to a pandas DataFrame with appropriate column names
4 X_SMOTE = pd.DataFrame(X_SMOTE, columns=X.columns)
5
6
```

Appendix G: Data Splitting

Split Dataset

```
In [ ]: 1 Y = New_df['target']
2 X = New_df[['Gender','Reality','famsizegp_1.0', 'famsizegp_2.0', 'famsizegp_3more',
3           'gp_Age_high',
4           'gp_Age_highest', 'gp_Age_low', 'gp_Age_lowest', 'gp_Age_medium',
5           'gp_worktm_high', 'gp_worktm_highest', 'gp_worktm_low',
6           'gp_worktm_lowest', 'gp_worktm_medium',
7           'OccupationType_Laborwk',
8           'OccupationType_hightecwk', 'OccupationType_officewk',
9           'HousingType_Co-op apartment', 'HousingType_House / apartment',
10          'HousingType_Municipal apartment', 'HousingType_Office apartment',
11          'HousingType_Rented apartment', 'HousingType_With parents',
12          'EducationType_Higher education', 'EducationType_Incomplete higher',
13          'EducationType_Lower secondary',
14          'EducationType_Secondary / secondary special',
15          'Marital_status_Civil marriage', 'Marital_status_Married',
16          'Marital_status_Separated', 'Marital_status_Single / not married',
17          'Marital_status_Widow']]
```

```
In [ ]: 1 X_train, X_test, y_train, y_test = train_test_split(scaled_X_SMOTE,Y_SMOTE,
2                                     stratify=Y_SMOTE,
3                                     test_size=0.3,
4                                     random_state = 0)
```

```
In [ ]: 1 X_train.shape , X_test.shape
```

```
Out[371]: ((34596, 33), (14828, 33))
```

Appendix H: Logistic Regression

Logistic Regression

```
In [ ]: 1 log_reg_model = LogisticRegression(random_state=42)
2
3 # Fit the model to the training data
4 log_reg_model.fit(X_train, y_train)
5
6 # Make predictions using the optimized logistic regression model on the train data and test data
7 y_train_pred_logreg = cross_val_predict(log_reg_model, X_train, y_train, cv=3)
8 y_test_pred_logreg = cross_val_predict(log_reg_model, X_test, y_test, cv=3)
9
10 print_score(y_train, y_train_pred_logreg, train=True)
11 print_score(y_test, y_test_pred_logreg, train=False)
12
13 lr_fscore= f1_score(y_test, y_test_pred_logreg)
14 lr_accuracy = accuracy_score(y_test, y_test_pred_logreg)
15 lr_recall = recall_score(y_test, y_test_pred_logreg, average='weighted')
16 lr_precision = precision_score(y_test, y_test_pred_logreg, average='weighted')
```

Train Result:

```
=====
Accuracy Score: 63.37%
F1 Score: 64.59%
```

Test Result:

```
=====
Accuracy Score: 63.55%
F1 Score: 64.93%
```

Classification Report:

	0	1	accuracy	macro avg	weighted avg
precision	0.647050	0.625610	0.635487	0.636330	0.636330
recall	0.596169	0.674804	0.635487	0.635487	0.635487
f1-score	0.620569	0.649276	0.635487	0.634923	0.634923
support	7414.000000	7414.000000	0.635487	14828.000000	14828.000000

<Figure size 640x480 with 2 Axes>

Appendix I: Decision Tree

Decision Tree

```
In [ ]: 1 Dec_Tre_model = DecisionTreeClassifier(random_state=42)
2
3 # Fit the model to the training data
4 Dec_Tre_model.fit(X_train, y_train)
5
6 # Make predictions using the optimized Decision Tree Classifier model on the train data and test data
7 y_train_pred_DecTre = cross_val_predict(Dec_Tre_model, X_train, y_train, cv=3)
8 y_test_pred_DecTre = cross_val_predict(Dec_Tre_model, X_test, y_test, cv=3)
9
10 print_score(y_train, y_train_pred_DecTre, train=True)
11 print_score(y_test, y_test_pred_DecTre, train=False)
12
13 dt_fscore= f1_score(y_test, y_test_pred_DecTre)
14 dt_accuracy = accuracy_score(y_test, y_test_pred_DecTre)
15 dt_recall = recall_score(y_test, y_test_pred_DecTre, average='weighted')
16 dt_precision = precision_score(y_test, y_test_pred_DecTre, average='weighted')
```

Train Result:

=====

Accuracy Score:	92.55%
F1 Score:	92.68%

Test Result:

=====

Accuracy Score:	91.68%
F1 Score:	91.84%

Classification Report:

	0	1	accuracy	macro avg	weighted avg
precision	0.933989	0.900882	0.916779	0.917435	0.917435
recall	0.896952	0.936606	0.916779	0.916779	0.916779
f1-score	0.915096	0.918397	0.916779	0.916746	0.916746
support	7414.000000	7414.000000	0.916779	14828.000000	14828.000000

<Figure size 640x480 with 2 Axes>

Appendix J: Random Forest

Random Forest

```
In [ ]: 1 Ran_For_model = RandomForestClassifier(random_state=42)
2
3 # Fit the model to the training data
4 Ran_For_model.fit(X_train, y_train)
5
6 # Make predictions using the optimized Random Forest Classifier model on the train data and test data
7 y_train_pred_RanFor = cross_val_predict(Ran_For_model, X_train, y_train, cv=3)
8 y_test_pred_RanFor = cross_val_predict(Ran_For_model, X_test, y_test, cv=3)
9
10 print_score(y_train, y_train_pred_RanFor, train=True)
11 print_score(y_test, y_test_pred_RanFor, train=False)
12
13 rf_fscore= f1_score(y_test, y_test_pred_RanFor)
14 rf_accuracy = accuracy_score(y_test, y_test_pred_RanFor)
15 rf_recall = recall_score(y_test, y_test_pred_RanFor, average='weighted')
16 rf_precision = precision_score(y_test, y_test_pred_RanFor, average='weighted')
```

Train Result:

=====

Accuracy Score:	92.82%
F1 Score:	92.96%

Test Result:

=====

Accuracy Score:	92.19%
F1 Score:	92.34%

Classification Report:

	0	1	accuracy	macro avg	weighted avg
precision	0.939573	0.905602	0.921905	0.922587	0.922587
recall	0.901807	0.942002	0.921905	0.921905	0.921905
f1-score	0.920303	0.923443	0.921905	0.921873	0.921873
support	7414.000000	7414.000000	0.921905	14828.000000	14828.000000

<Figure size 640x480 with 2 Axes>

Appendix K: SVM

SVM

```
In [ ]: 1 SVM_model = svm.SVC(random_state=42)
2
3 # Fit the model to the training data
4 SVM_model.fit(X_train, y_train)
5
6 # Make predictions using the optimized SVM model on the train data and test data
7 y_train_pred_SVM = cross_val_predict(SVM_model, X_train, y_train, cv=3)
8 y_test_pred_SVM = cross_val_predict(SVM_model, X_test, y_test, cv=3)
9
10 print_score(y_train, y_train_pred_SVM, train=True)
11 print_score(y_test, y_test_pred_SVM, train=False)
12
13 SVM_fscore = f1_score(y_test, y_test_pred_SVM)
14 SVM_accuracy = accuracy_score(y_test, y_test_pred_SVM)
15 SVM_recall = recall_score(y_test, y_test_pred_SVM, average='weighted')
16 SVM_precision = precision_score(y_test, y_test_pred_SVM, average='weighted')
```

Train Result:

Accuracy Score: 88.10%
F1 Score: 88.51%

Test Result:

Accuracy Score: 86.08%
F1 Score: 86.74%

Classification Report:

	0	1	accuracy	macro avg	weighted avg
precision	0.900329	0.828382	0.860804	0.864356	0.864356
recall	0.811438	0.910170	0.860804	0.860804	0.860804
f1-score	0.853575	0.867352	0.860804	0.860464	0.860464
support	7414.000000	7414.000000	0.860804	14828.000000	14828.000000

<Figure size 640x480 with 2 Axes>

Appendix L: LightGBM

LightGBM

```
In [ ]: 1 lgbm_Model = LGBMClassifier(random_state=42)
2
3 # Fit the model to the training data
4 lgbm_Model.fit(X_train, y_train)
5
6 # Make predictions using the optimized LGBM Classifier on the train data and test data
7 y_train_pred_lgbm = cross_val_predict(lgbm_Model, X_train, y_train, cv=3)
8 y_test_pred_lgbm = cross_val_predict(lgbm_Model, X_test, y_test, cv=3)
9
10 print_score(y_train, y_train_pred_lgbm, train=True)
11 print_score(y_test, y_test_pred_lgbm, train=False)
12
13 LGBM_fscore = f1_score(y_test, y_test_pred_lgbm)
14 LGBM_accuracy = accuracy_score(y_test, y_test_pred_lgbm)
15 LGBM_recall = recall_score(y_test, y_test_pred_lgbm, average='weighted')
16 LGBM_precision = precision_score(y_test, y_test_pred_lgbm, average='weighted')
```

LightGBM [Warning] Found whitespaces in feature names, replace with underscores

Accuracy Score: 92.27%
F1 Score: 92.37%

Test Result:

Accuracy Score: 92.03%
F1 Score: 92.20%

Classification Report:

	0	1	accuracy	macro avg	weighted avg
precision	0.939492	0.902688	0.920286	0.921090	0.921090
recall	0.898435	0.942136	0.920286	0.920286	0.920286
f1-score	0.918505	0.921990	0.920286	0.920248	0.920248
support	7414.000000	7414.000000	0.920286	14828.000000	14828.000000

Appendix M: Xgboost

Xgboost

```
In [ ]: 1 xgb_classifier = XGBClassifier(random_state=42)
2
3 # Fit the model to the training data
4 xgb_classifier.fit(X_train, y_train)
5
6 # Make predictions using the optimized LGBM Classifier on the train data and test data
7 y_train_pred_xgb = cross_val_predict(xgb_classifier, X_train, y_train, cv=3)
8 y_test_pred_xgb = cross_val_predict(xgb_classifier, X_test, y_test, cv=3)
9
10 print_score(y_train, y_train_pred_xgb, train=True)
11 print_score(y_test, y_test_pred_xgb, train=False)
12
13 XGB_fscore = f1_score(y_test, y_test_pred_xgb)
14 XGB_accuracy = accuracy_score(y_test, y_test_pred_xgb)
15 XGB_recall = recall_score(y_test, y_test_pred_xgb, average='weighted')
16 XGB_precision = precision_score(y_test, y_test_pred_xgb, average='weighted')
```

Train Result:

=====

Accuracy Score:	92.75%
F1 Score:	92.84%

Test Result:

=====

Accuracy Score:	92.53%
F1 Score:	92.69%

Classification Report:

	0	1	accuracy	macro avg	weighted avg
precision	0.944217	0.908009	0.925344	0.926113	0.926113
recall	0.904100	0.946588	0.925344	0.925344	0.925344
f1-score	0.923724	0.926897	0.925344	0.925310	0.925310
support	7414.000000	7414.000000	0.925344	14828.000000	14828.000000

<Figure size 640x480 with 2 Axes>

Appendix N: Summary result for Models

Summary result for Models

```
In [ ]: 1 models = ['Random Forest', 'Decision Tree', 'Support Vector Machine', 'Logistic Regression', 'XGBoost', 'LGBM']
2 f1_scores = [rf_fscore, dt_fscore, svm_fscore, lr_fscore, XGB_fscore, LGBM_fscore]
3 accuracies = [rf_accuracy, dt_accuracy, svm_accuracy, lr_accuracy, XGB_accuracy, LGBM_accuracy]
4 recall = [rf_recall, dt_recall, svm_recall, lr_recall, XGB_recall, LGBM_recall]
5 precision = [rf_precision, dt_precision, svm_precision, lr_precision, XGB_precision, LGBM_precision]
6
7
8 df_second_table = create_summary(models, f1_scores, accuracies, recall, precision)
9 # Apply border style to the dataframe for display
10 styled_table = df_second_table.style.highlight_max(subset=['Accuracy', 'F1 Score', 'Recall', 'Precision'], color='green')
11 styled_table
12
```

Out[392]:

	Models	Accuracy	F1 Score	Recall	Precision
0	Random Forest	0.922%	0.923%	0.922%	0.923%
1	Decision Tree	0.917%	0.918%	0.917%	0.917%
2	Support Vector Machine	0.881%	0.887%	0.881%	0.884%
3	Logistic Regression	0.836%	0.849%	0.836%	0.836%
4	XGBoost	0.925%	0.927%	0.925%	0.926%
5	LGBM	0.920%	0.922%	0.920%	0.921%

Appendix O: After Feature selection & HyperParameters For champion

Xgboost

```
In [ ]: 1 best_score = 0
2 best_features = []
3
4 # Loop through a range of feature counts to find the best set of features
5 for num_features in range(1, len(X.columns) + 1):
6     # Initialize XGBoost classifier
7     model = XGBClassifier(use_label_encoder=False, eval_metric='logloss') # You can add more parameters based on your requirements
8
9     # Initialize RFE with the current number of features
10    rfe = RFE(model, n_features_to_select=num_features)
11
12    # Fit RFE
13    rfe.fit(X_train, y_train)
14
15    # Get selected features
16    selected_features = X.columns[rfe.support_]
17
18    # Train a model using the selected features
19    model.fit(X_train[selected_features], y_train)
20
21    # Predict on the test set
22    y_pred = model.predict(X_test[selected_features])
23
24    # Calculate accuracy score
25    score = accuracy_score(y_test, y_pred)
26
27    # Update best score and best features if the current model is better
28    if score > best_score:
29        best_score = score
30        best_features = selected_features.tolist()
31
32    print("Best Features:", best_features)
33    print("Best Accuracy Score:", best_score)
34
```

Best Features: ['Gender', 'Reality', 'famsizegp_1.0', 'famsizegp_2.0', 'famsizegp_3more', 'gp_Age_high', 'gp_Age_highest', 'gp_Age_low', 'gp_Age_lowest', 'gp_Age_medium', 'gp_worktm_high', 'gp_worktm_highest', 'gp_worktm_low', 'gp_worktm_lowest', 'gp_worktm_medium', 'OccupationType_Laborwk', 'OccupationType_hightecwk', 'OccupationType_officewk', 'HousingType_House / apartment', 'HousingType_with parents', 'EducationType_Higher education', 'EducationType_Incomplete higher', 'EducationType_Lower secondary', 'EducationType_Secondary / secondary special', 'Marital_status_Civil marriage', 'Marital_status_Married', 'Marital_status_Separated', 'Marital_status_Single / not married', 'Marital_status_Widow']

Best Accuracy Score: 0.9283113029403831

```
In [ ]: 1 X_train_best = X_train[best_features]
2 X_test_best = X_test[best_features]
```

```
In [ ]: 1 # Hyperparameters grid for XGBClassifier
2 param_grid = {
3     'learning_rate': [0.01, 0.05, 0.1],
4     'max_depth': [5, 10, 20],
5     'n_estimators': [100, 500, 1000],
6     'subsample': [0.5, 0.7, 1.0]
7 }
8
9 # Initialize the XGBClassifier with some default parameters
10 xgb_model = XGBClassifier(random_state=42)
11
12 # Initialize GridSearchCV with the XGBClassifier, hyperparameter grid, and scoring metric (e.g., accuracy)
13 grid_search = GridSearchCV(estimator=xgb_model, param_grid=param_grid, scoring='accuracy', cv=3, n_jobs=-1, verbose=2)
14
15 # Fit the grid search to the training data (using the best features)
16 grid_search.fit(X_train_best, y_train)
17
18 # Get the best hyperparameters and model
19 best_params = grid_search.best_params_
20 best_xgb_model = grid_search.best_estimator_
21
22 print("Best Hyperparameters:", best_params)
23 print("Best XGB Model:", best_xgb_model)
24
```

Fitting 3 folds for each of 81 candidates, totalling 243 fits
Best Hyperparameters: {'learning_rate': 0.1, 'max_depth': 20, 'n_estimators': 500, 'subsample': 0.5}
Best XGB Model: XGBClassifier(base_score=None, booster=None, callbacks=None,
colsample_bylevel=None, colsample_bynode=None,
colsample_bytree=None, device=None, early_stopping_rounds=None,
enable_categorical=False, eval_metric=None, feature_types=None,
gamma=None, grow_policy=None, importance_type=None,
interaction_constraints=None, learning_rate=0.1, max_bin=None,
max_cat_threshold=None, max_cat_to_onehot=None,
max_delta_step=None, max_depth=20, max_leaves=None,
min_child_weight=None, missing=nan, monotone_constraints=None,
multi_strategy=None, n_estimators=500, n_jobs=None,
num_parallel_tree=None, random_state=42, ...)

```
In [ ]: 1 # Make predictions using the optimized LGBM Classifier on the test data
2 y_test_pred_xgb = cross_val_predict(best_xgb_model, X_test_best, y_test, cv=3)
3
4 print_score(y_test, y_test_pred_xgb, train=False)
5
6 XGB_fscore = f1_score(y_test, y_test_pred_xgb)
7 XGB_accuracy = accuracy_score(y_test, y_test_pred_xgb)
8 XGB_recall = recall_score(y_test, y_test_pred_xgb, average='weighted')
9 XGB_precision = precision_score(y_test, y_test_pred_xgb, average='weighted')
```

Test Result:

=====

Accuracy Score:	92.77%
F1 Score:	92.94%

Classification Report:					
	0	1	accuracy	macro avg	weighted avg
precision	0.949915	0.907584	0.927704	0.928749	0.928749
recall	0.903021	0.952387	0.927704	0.927704	0.927704
f1-score	0.925875	0.929446	0.927704	0.927660	0.927660
support	7414.000000	7414.000000	0.927704	14828.000000	14828.000000

<Figure size 640x480 with 2 Axes>

```

In [ ]: 1 models = ['Random Forest', 'Decision Tree', 'Support Vector Machine', 'Logistic Regression', 'XGBoost', 'LGBM']
        2 f1_scores = [rf_fscore, dt_fscore, SVM_fscore, lr_fscore, XGB_fscore, LGBM_fscore ]
        3 accuracies = [rf_accuracy, dt_accuracy, SVM_accuracy, lr_accuracy, XGB_accuracy, LGBM_accuracy]
        4 recall = [rf_recall, dt_recall, SVM_recall, lr_recall, XGB_recall, LGBM_recall]
        5 precision= [rf_precision, dt_precision, SVM_precision, lr_precision, XGB_precision, LGBM_precision]
        6
        7
        8 df_second_table = create_summary(models, f1_scores, accuracies, recall, precision)
        9 # Apply border style to the dataframe for display
       10 styled_table = df_second_table.style.highlight_max(subset=['Accuracy', 'F1 Score', 'Recall', 'Precision'], color='green')
       11 styled_table
       12

```

Out[397]:

	Models	Accuracy	F1 Score	Recall	Precision
0	Random Forest	0.922%	0.923%	0.922%	0.923%
1	Decision Tree	0.917%	0.918%	0.917%	0.917%
2	Support Vector Machine	0.881%	0.887%	0.881%	0.884%
3	Logistic Regression	0.835%	0.840%	0.835%	0.838%
4	XGBoost	0.928%	0.929%	0.928%	0.929%
5	LGBM	0.920%	0.922%	0.920%	0.921%