INTERNATIONALE
HOCHSCHULE
FERNSTUDIUM

Case Study: Model Engineering

(DLMDSME01)

A Case Study on

Credit Card Routing for Online Purchase via Predictive Modelling

Author: Kaushik Puttaswamy

Matriculation Number: 321150196

Customer ID: 10549633

Tutor's Name 1: Mohammad reza Nilchiyan

Tutor's Name 2: Sahar Qaadan

Date: 13 January 2024

Table of Contents

List of Figures

List of Tables

List of Outputs

Table of Abbreviations

PSP	Payment Service Provider
CRISP-DM	CRoss-Industry Standard Process for Data Mining
KNN	K-Nearest Neighbor
SVM	Support Vector Machine
VCS	Version Control System
EDA	Exploratory Data Analysis
IT	Information Technology
ML	Machine Learning
DQA	Data Quality Assessment
3D Secure	Three Domains Secure
SMOTE	Synthetic Minority Oversampling Technique
AUC-ROC	Area Under the Receiver Operating Characteristic Curve
SVC	Support Vector Classification
CV	Cross Validation
SHAP	SHapley Additive exPlanations
AUC-PR	Area Under the Precision-Recall Curve
GUI	Graphical User Interface

Chapter 1

Introduction

1.1 Overview:

In the fast-paced world of online retail, rapid credit card transaction processing is critical for a positive customer experience and financial success. As a newly appointed data scientist at one of the world's leading retailers, our first task is to solve a major concern: a considerable increase in the failure rate of online credit card payments over the previous year. This practice has not only cost the corporation significant financial losses, but it has also left customers dissatisfied with their online buying experience. The current manual and rule-based credit card routing system, which is operated by payment service providers (PSPs), has proven insufficient. In response to this problem, business stakeholders are looking for a transformative response via predictive modelling. The goal is straightforward: build an automated credit card routing system that not only improves payment success rates but also strategically reduces the cost of transactions. This project takes a comprehensive approach, designed according to the CRoss-Industry Standard Process for Data Mining (CRISP-DM), providing a two-model strategy to achieve the dual goals of enhancing success rates and lowering transaction costs. The goal of this project is to transform credit card routing in the online retail realm using a combination of investigative methodologies, data analysis, preparation, modelling, evaluation, and deployment, aligning with the broader business objectives of financial efficiency and customer satisfaction.

1.2 Problem Definition:

The online payment department of a major retailer is dealing with a considerable increase in online credit card transaction failures, which is producing significant financial losses and consumer dissatisfaction. The current manual credit card routing method via payment service providers (PSPs) is inefficient, necessitating the need for a data-driven solution. The primary challenge is developing predictive models to automate credit card routing, increase transaction success rates, and reduce fees. This assignment entails selecting a PSP based on success possibilities and contractual agreements with four PSPs. The ultimate goal is to create an efficient predictive model system that aligns with the company's larger goals of financial performance and customer satisfaction.

1.3 Objectives:

- Automate Credit Card Routing: Developing a predictive model to automate the credit card routing process and replace the present manual and rule-based method.
- Increase Payment Success Rate: Improving the success rate of online credit card payments by using predictive modelling approaches to optimize the selection of payment service providers (PSPs).

- Minimize Transaction Fees: Reducing transaction fees strategically by introducing predictive models into the decision-making process that take both success probabilities and transaction fees into account.
- Ensure Model Interpretability: To build confidence in business stakeholders, emphasize model interpretability and transparency by offering clear insights into the elements impacting credit card routing decisions and supporting informed decision-making.

1.4 Research Questions:

1) Effectiveness of a Two-Model Approach:

How can the two-model strategy, which includes success prediction (Model 1) and PSP selection (Model 2), help to improve payment success rates and reduce transaction fees? What is the impact of using different modelling techniques, such as K-Nearest Neighbor (KNN), Logistic Regression, Support Vector Machines (SVM), and Random Forest Classification, in reaching these goals?

2) Model Selection and Performance:

Which of the modelling strategies (KNN, Logistic Regression, Support Vector Machines (SVM), or Random Forest Classification) performs best in terms of predicting success probability (Model 1) and choosing the best PSP (Model 2)? In terms of accuracy, interpretability, and overall effectiveness in credit card routing, how do the selected models compare to a baseline model?

1.5 Scope and Significance of Case Study:

Scope:

- Addresses the high rate of credit card transaction failure in online retail by using historical data and machine learning approaches to automate routing across two predictive models (KNN, Logistic Regression, and SVM, Random Forest).
- Increase success rates while lowering transaction costs.
- Collaborate with departments to review comprehensive data, with an emphasis on interpretability and transparency.

Significance:

- Automated credit card routing can be used to transform online retail payments.
- Implementing a CRISP-DM-aligned two-model strategy.
- Targeting minimizes financial losses while increasing customer satisfaction.
- Contribute to financial and customer satisfaction objectives.
- Incorporating machine learning techniques to optimize online payment procedures.
- Contingencies should be used to address risks, and good stakeholder communication should be prioritized.

- Creating a deployment strategy for long-term optimization.

1.6 Structure of Case Study:

The use case study structure begins with Chapter 2: building a Git Repository Proposal, which describes a well-organized structure for the credit card routing project. Furthermore, the CRISP-DM procedure is introduced in the Investigative Method (Chapter 3), corresponding with the project's two-model approach. The proposal outlines business comprehension. The proposal emphasizes effective stakeholder communication while outlining business understanding, project context, objectives, and a thorough strategy. Requirements, risks, and tools are addressed concisely, exhibiting alignment with business goals and effective data interpretation, preparation, and modelling procedures. The evaluation and deployment steps are briefly described to ensure a comprehensive approach to meeting project objectives. The study concludes by summarizing key results and insights.

Chapter 2

Git Repository Proposal

A Git repository is essentially a database that contains all of the information needed to maintain and manage the changes and history of a project. A Git repository, like most version control systems, preserves an exact copy of the entire project for the rest of its existence. Unlike most other VCSs, the Git repository provides not only a complete working copy of all the files in the repository, but also a copy of the repository itself to work with (*4. Basic Git Concepts - Version Control with Git, 2nd Edition [Book]*, n.d.). As a result, below is the proposed git repository for the project credit card routing for online purchases via predictive modelling.

```
credit-card-routing/
|-- data/
|       |-- raw/
|       |   |-- raw_data_ psp_jan_feb_2019.xlsx
|       |-- processed/
|       |   |-- cleaned_data_ psp_jan_feb_2019.xlsx
|       |   |-- features/
|       |   |   |-- selected_features_ psp_jan_feb_2019.xlsx
|-- notebooks/
|       |-- exploratory_data_analysis.ipynb
|       |-- data_preparation.ipynb
|       |-- modeling/
|       |   |-- model_1_training.ipynb
|       |   |-- model_2_training.ipynb
|       |-- evaluation/
|       |   |-- confusion_matrix_analysis.ipynb
|       |   |-- precision_recall_tradeoff.ipynb
|       |   |-- roc_curve_analysis.ipynb
|       |   |-- misclassified_samples_inspection.ipynb
|          |  |-- cross_validation_stability.ipynb
|          |  |-- overfitting_analysis.ipynb
|-- src/
|          |-- data/
|          |  |-- data_loading.py
|          |  |-- data_cleaning.py
|          |  |-- feature_engineering.py
|          |  |-- data_splitting.py
|          |-- modeling/
|          |  |-- model_1.py
|          |  |-- model_2.py
|          |-- evaluation/
|          |  |-- confusion_matrix_analysis.py
|          |  |-- precision_recall_tradeoff.py
|          |  |-- roc_curve_analysis.py
|          |  |-- misclassified_samples_inspection.py
|          |  |-- cross_validation_stability.py
|          |  |-- overfitting_analysis.py
|-- requirements.txt
|-- README.md
|-- .gitignore
```

The Git repository for credit card routing in online purchases through predictive modelling is divided into three sections: "data/" for raw and processed data, "notebooks/" for Jupyter notebooks covering EDA, data preparation, modelling, and evaluation, and "src/" for organized source code. A "requirements.txt" file ensures that dependencies are consistent, while "README.md" gives a project description, installation guide, and important information. By identifying excluded files, the ".gitignore" file simplifies collaboration. This framework makes project creation, collaboration, and comprehension easier.

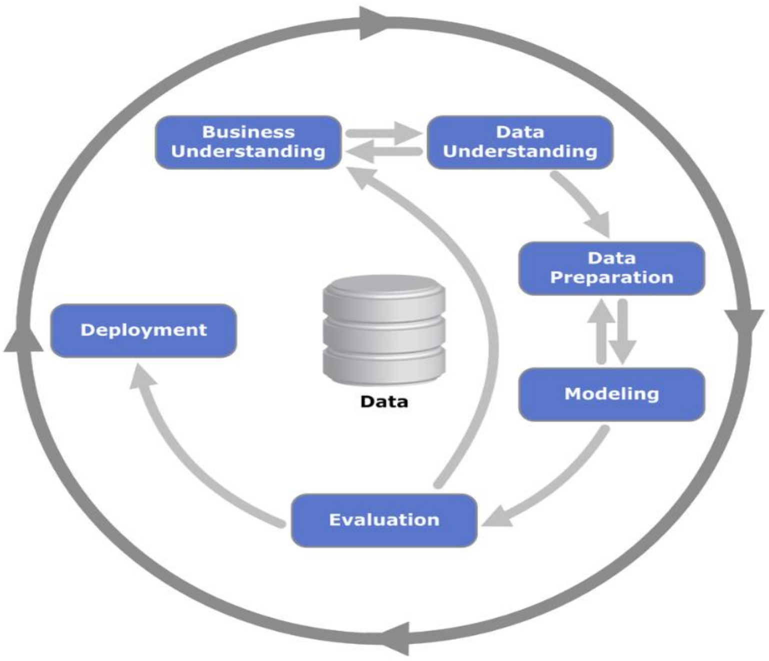
Chapter 3

Investigative Method

3 CRoss-Industry Standard Process for Data Mining (CRISP-DM)

The approach adopted for the credit card routing for online purchase via predictive modelling use case is CRISP-DM, a comprehensive data mining process model with six phases: business understanding, data understanding, data preparation, modelling, evaluation, and deployment. Figure 1 depicts the phases of a data mining process. This framework provides a formal plan for the whole project life cycle, guiding both novices and specialists alike. The cyclical structure of CRISP-DM recognizes the iterative process of data mining, with lessons learned during the project and from the deployed solution driving new, typically more focused business concerns (Bhagwat, n.d.).

Figure 1: CRISP-DM Reference Model



Source: Jensen, 2012

3.1 Business Understanding

The first stage of our CRISP-DM approach entails acquiring a comprehensive grasp of the retail company's goals. During the business understanding phase, it is vital to identify the key aspects that may influence the outcome of our project in order to formulate the task and outline the proposed rough method (“CRISP DM Step 1 - Understanding About the Business,” n.d.).

The business understanding phase in the CRISP-DM methodology involves several key components in the context of the provided use case, namely the identification of "Credit Card Routing for Online Purchase via Predictive Modelling", and this phase should end with the creation of a first project plan detailing the resources that are likely needed, any specific requirements, assumptions, constraints, risks, or contingencies, as well as the tools and techniques used in the project (Bhagwat, n.d.).

3.1.1 Project Context:

- In a major retail company's online payment department, a high failure rate in online credit card payments leads to financial losses and consumer dissatisfaction.
- Four payment service providers (PSPs) with different transaction costs are involved in online payments, and credit card routing is currently manual, necessitating automation and predictive modelling.

3.1.2 Project Objectives:

- Creating a predictive model to automate online credit card routing.
- Improving efficiency by increasing payment success rates, optimizing PSP selection, and lowering transaction fees.

3.1.3 Project Plan:

We have two primary objectives based on the project goal: improving payment success rates and minimizing transaction fees. The proposed project plan method entails developing two distinct models to address each of these goals:

1) Two-model approach:

a) Model 1: Success prediction:

- Predicts success probability while focusing primarily on increasing success rates.
- The success probability from Model 1 is used as input for the second model.

b) Model 2: PSP selection:

- Choosing the best payment service provider (PSP) based on success probabilities and transaction fees.
- Allows for dynamic decision-making by assessing success probabilities and costs.

2) Implementation steps:

a) Train Model 1:

- Using transaction-related features (excluding fees) with success as the target variable.
- Objective: Predict success probabilities.

b) Predict success probabilities:

- Appling Model 1 to obtain success probabilities.

c) Train Model 2:

- Utilizing success probabilities from Model 1, transaction fees, and relevant features with PSP as the target variable.
- Objective: Predicting the optimal PSP, considering success probabilities and fees.

d) Routing decision:

- Using success probabilities from Model 1 and Model 2 predictions to determine the best PSP.
- Leverage fee comparison features in case of ties.

3) Benefits:

a) Specialization:

- Model 1 optimizes success rates.
- Model 2 minimizes transaction fees, considering success probabilities.

b) Interpretability:

- Separate models provide transparency for each decision-making aspect.

3.1.4 Requirements, Assumptions, and Constraints:

1) Requirements:

- Access to historical transaction datasets for model training.
- Collaboration with the online payment department and IT teams.

2) Assumptions:

- The prediction model will have access to accurate and complete data.
- Transactions occurring within one minute, for the same amount, and from the same country may suggest several attempts to make the same purchase. This temporal trend will be taken into account by the machine learning model, particularly when transactions initially fail, causing clients to attempt numerous transfers.

3) Constraints:

- Adherence to regulatory limits pertaining to online payment processing.
- The barriers imposed by existing contracts with four separate PSPs.

3.1.5 Risks and Contingencies:

1) Risks:

- Inaccuracies in historical transaction data.
- Regulatory changes impacting credit card routing.

2) Contingencies:

- Implementing data validation checks and cleaning processes.

- Stay informed about regulatory updates and adapt the model accordingly.

3.1.6 Tools and Techniques:

- Automating credit card routing with predictive modelling techniques such as machine learning algorithms.
- Making use of data preparation techniques to handle and clean past transaction data.

3.1.7 Alignment with Business Goals:

The project is in line with the retail company's broader business objectives, which include:

- Improving financial performance by decreasing losses from failed transactions.
- Enhancing customer satisfaction with the online shopping experience.

3.1.8 Stakeholder Communication:

- Establishing effective links with business stakeholders from the online payment department.
- Maintaining constant collaboration and feedback throughout the project.

3.2 Data Understanding

The data understanding phase begins with data collection and proceeds with actions to become acquainted with the data, such as finding data quality issues, uncovering first insights into the data, or detecting interesting subsets to create hypotheses for hidden information. This phase is crucial because it helps the analyst comprehend the data and ensures that the data quality is appropriate to depict relationships and act as a trustworthy basis from which to draw accurate inferences (*MyEducator - CRISP-DM*, n.d.).

Our key objectives in the "Data Understanding" phase of a credit card routing project are to obtain, assess, and study the available data. Here are the steps we can take:

3.2.1 Acquire Relevant Data:

The essential data for this case study is already provided by the file name "PSP_Jan_Feb_2019.xlsx", which consists of a list of credit card transactions for DACH countries (Germany, Switzerland, and Austria). Furthermore, all important business information (such as PSP names and transaction costs) is available. As a result, column names in the context of a dataset frequently indicate features or variables. Each column relates to a distinct attribute or quality of the data. The features in the dataset provide information about the observations or occurrences.

In the provided column description:

- tmsp (timestamp of transaction): This feature represents the timestamp when each transaction occurred.

- country (country of transaction): This feature indicates the country where each transaction took place.
- amount (transaction amount): This feature represents the monetary value associated with each transaction.
- success: This binary feature indicates the success or failure of the payment for each transaction.
- PSP (name of payments service provider): This feature holds the names of the payment service providers associated with each transaction.
- 3D_secured: This binary feature indicates whether the customer is 3D-identified for more secure online credit card payments.
- card (credit card provider: Master, Visa, Diners): This feature specifies the credit card provider associated with each transaction.

Features are important in predictive modelling because they are used to create observations and predictions based on the data supplied.

3.2.2 Data Quality Assessment:

Data Quality Assessment (DQA) is critical for the credit card routing project since it ensures the predictive model's dependability by resolving issues such as missing values and outliers. DQA increases stakeholder trust by ensuring high-quality data for training and testing. Understanding data constraints allows for more informed decision-making, and effective data quality assurance reduces the "garbage in, garbage out" problem (Aziz et al., 2012). It enables strong feature engineering, which is necessary for capturing significant patterns. DQA also reduces the likelihood of model failure by detecting and fixing concerns early in the project (Otten, 2023). The data quality assessment for the given use case is as follows:

a) Check for Missing Values:

Examining each column for missing values and calculating the percentage of missing values in each column. According to the code output, there appear to be no missing values in any of the columns in our dataset, as indicated by the "Missing Values" column showing zeros for each variable, implying that our dataset is complete in terms of values for each field.

b) Verify Data Types:

Confirming that the data types in each column meet the criteria and descriptions. As a result, it appears that the data types match the criteria and descriptions for each column. Here's a breakdown of the data types:

Unnamed: 0: Integer (int64)

tmsp: Date and time (datetime64[ns])

country: Object (typically string or categorical)

amount: Integer (int64)

success: Integer (int64)

PSP: Object (typically string or categorical)

3D_secured: Integer (int64)

card: Object (typically string or categorical)

The data types specified above appear to be appropriate for each column, based on the information provided. The tmsp column contains the proper datetime type, while the other columns include the necessary numeric or category types.

c) Identify Systematic Deviations:

Searching for patterns or systematic deviations that may indicate difficulties with acquisition, processing, or transfer mechanisms. The distinct numbers in each column appear to be reasonable, and there do not appear to be any obvious systematic changes or issues in the acquisition, processing, or transfer operations. Each column has been summarized below:

Unnamed: 0: Represents an index, and the unique values seem to be in ascending order without any irregularities.

tmsp: Timestamps are in datetime format, and the unique values appear to be within the expected date and time range.

country: Contains the expected countries (Germany, Austria, Switzerland) without any unexpected values.

amount: Contains a variety of numeric values for transaction amounts without any obvious issues.

success: Binary variable with values 0 and 1, indicating whether the payment was successful.

PSP: Payment service provider names include 'UK_Card,' 'Simplecard,' 'Moneycard,' and 'Goldcard,' which seem appropriate.

3D_secured: Binary variable with values 0 and 1, indicating whether the customer is 3D identified.

card: Credit card provider names include 'Visa,' 'Diners,' and 'Master,' which also seem appropriate.

Overall, the unique values in each column match expectations, and no systematic deviations appear to pose serious problems.

3.2.3 Data Exploration:

The first step in data analysis is data exploration, which is used to explore and visualize data in order to uncover insights from the start or to discover regions or patterns to investigate further (*Spotfire | Unveiling Data Exploration*, n.d.). This method is not designed to display every piece of information in a dataset but rather to aid in the construction of a broad picture of notable trends and important topics to study further. Furthermore, data exploration can be classified into two types:

a) Preliminary Data Exploration:

This stage focuses on receiving an overview of the dataset and gaining preliminary insights. Creating histograms for each variable to investigate the data distribution.

After removing the unnamed column, the given dataset has 50,410 rows and 7 columns. In addition, the statistics summary of the amount column is shown below Table 1:

Table 1: Summary statistics for the 'amount' column

```
          count        mean       std  min    25%    50%    75%    max
amount  50410.0  202.395715  96.27473  6.0  133.0  201.0  269.0  630.0
```

According to these summary statistics, the majority of transactions (50%) had amounts less than or equal to 201.0. The data, however, varies, with some transactions having higher sums (up to a maximum of 630.0). The histogram can help us visualize this as shown Figure 2.

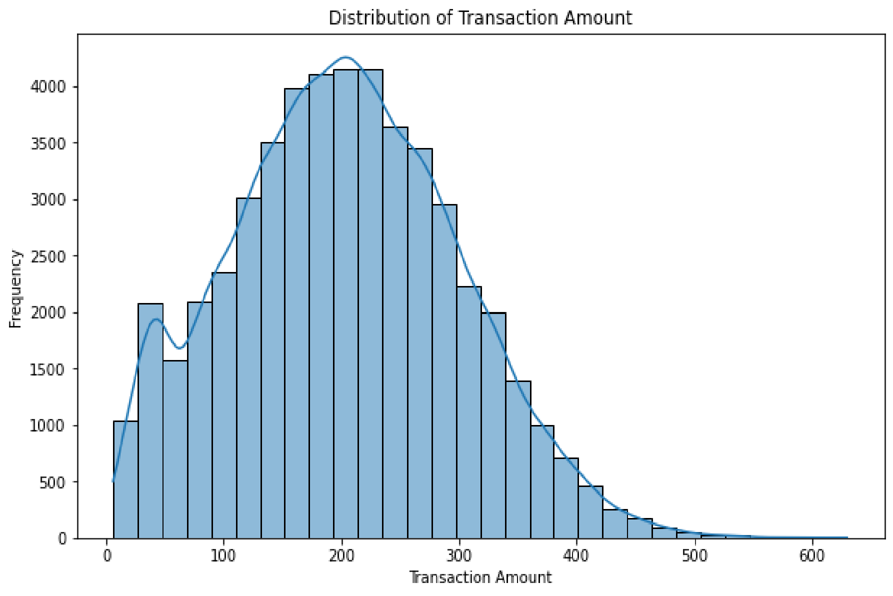


Figure 2: Visualization of transaction amount distribution

The histogram demonstrates a right-skewed distribution of transaction amounts. The majority of transactions have lower sums, generating a peak on the left. A tail on the right, on the other hand, implies that there are fewer transactions with greater sums, which contributes to the distribution's lengthy tail.

Furthermore, the 'success' column has 40,182 unsuccessful (0) transactions and 10,228 successful (1) transactions, according to the summary statistics. The distribution of the 'success' column is visualized in the below Figure 3. Similarly, the summary statistics for the '3D_secured' column show 38,399 transactions with 3D security and 12,011 transactions without 3D security. The visualization of the distribution of the '3D_secured' column is shown below Figure 4.

```
Summary statistics for 'success' column:
0    40182
1    10228
Name: success, dtype: int64
```

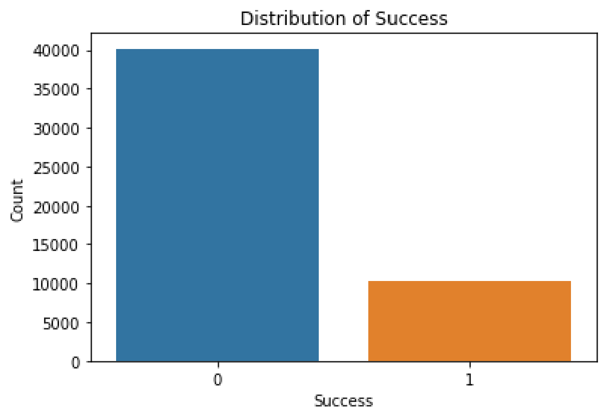


Figure 3: Visualization of success column

```
Summary statistics for '3D_secured' column:
0    38399
1    12011
Name: 3D_secured, dtype: int64
```

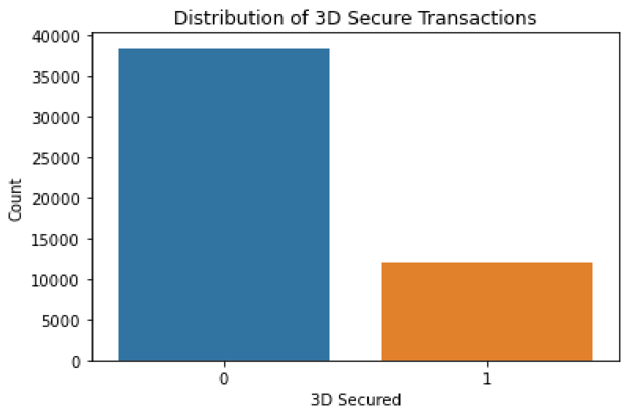


Figure 4: Visualization of 3D_secured column

Moreover, the summary statistics for the 'country' column shows that Germany has 30,233 entries, Switzerland has 10,338 entries, and Austria has 9,839 entries. The distribution of the 'country' column is depicted in the Figure 5 below.

```
Summary statistics for 'country' column:
Germany        30233
Switzerland    10338
Austria         9839
Name: country, dtype: int64
```

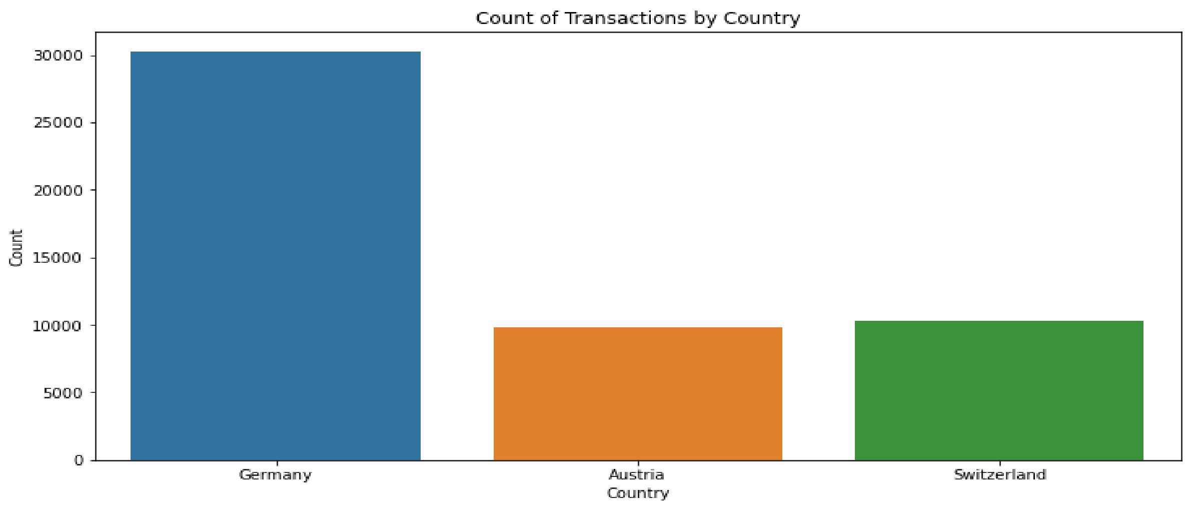


Figure 5: Visualization of country column distribution

According to the summary statistics for the 'PSP' column, there are 26,459 entries for UK_Card, 12,446 for Simplecard, 8,297 for Moneycard, and 3,208 for Goldcard. The distribution of the 'PSP' column is visualized in the below Figure 6.

```
Summary statistics for 'PSP' column:
UK_Card        26459
Simplecard     12446
Moneycard       8297
Goldcard        3208
Name: PSP, dtype: int64
```

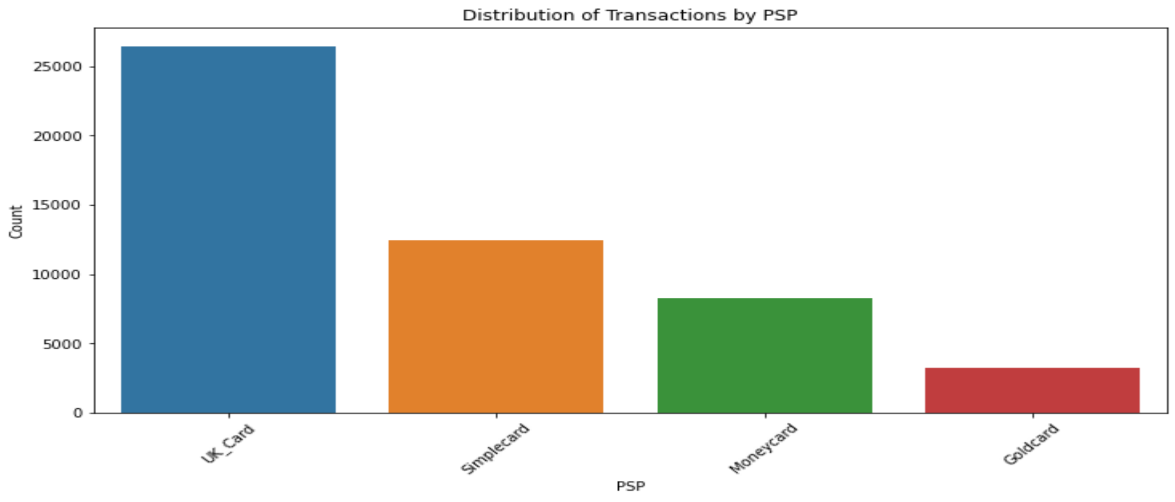


Figure 6: Visualization of PSP column distribution

```
Summary statistics for 'card' column:
Master    29002
Visa      11640
Diners     9768
Name: card, dtype: int64
```

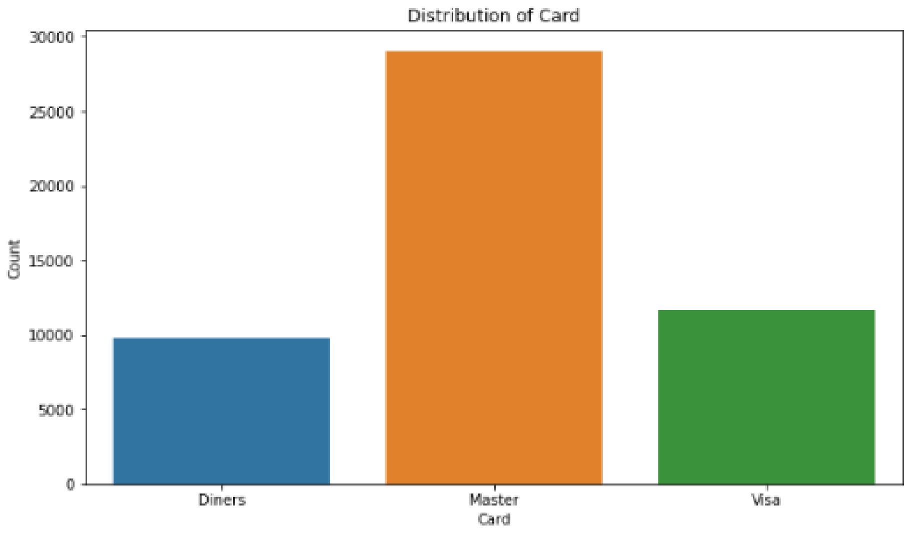


Figure 7: Visualization of card column distribution

According to the 'card' column visualization in Figure 7, the 'card' column has a variable distribution, with MasterCard being the most common option with 29,002 instances. Visa is the second most prevalent card type, with 11,640 reported instances, while Diners card is close behind with 9,768.

b) Detailed Data Exploration:

This step entails a more in-depth review of the data to discover details and patterns. Examining data in greater depth by investigating variable combinations, evaluating variable correlation, and analysing variable behavior in connection to one another.

The use of a boxplot to visualize the relationship between 'amount' and 'success' is vital for understanding how the transaction amount varies with transaction success or failure.

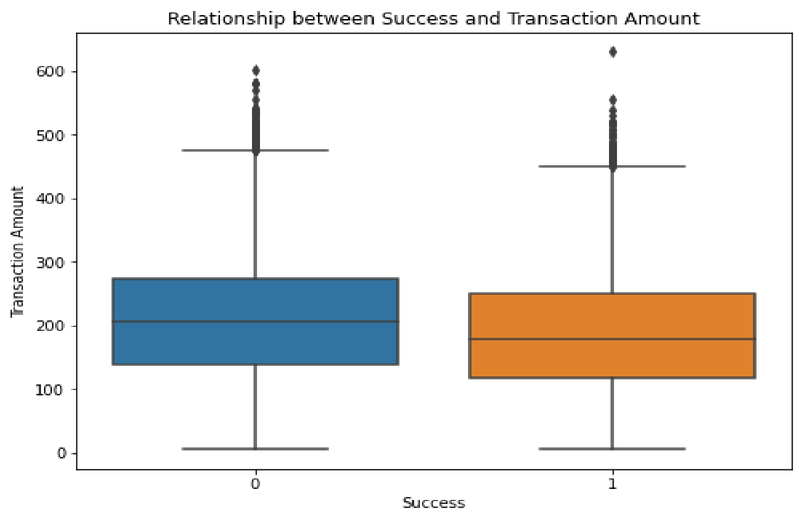


Figure 8: Visualization of relationship between amount and success using boxplot

The variations in median values and the presence of points above the whiskers in Figure 8 suggest that transaction amounts differ between successful and failed transactions. Furthermore, the pair plot is a valuable tool, but its efficacy is dependent on the structure of our dataset and the specific goals of our study. The pair plot visualizes given data to determine the relationship between them, where the variables can be continuous or categorical (SL, 2020).

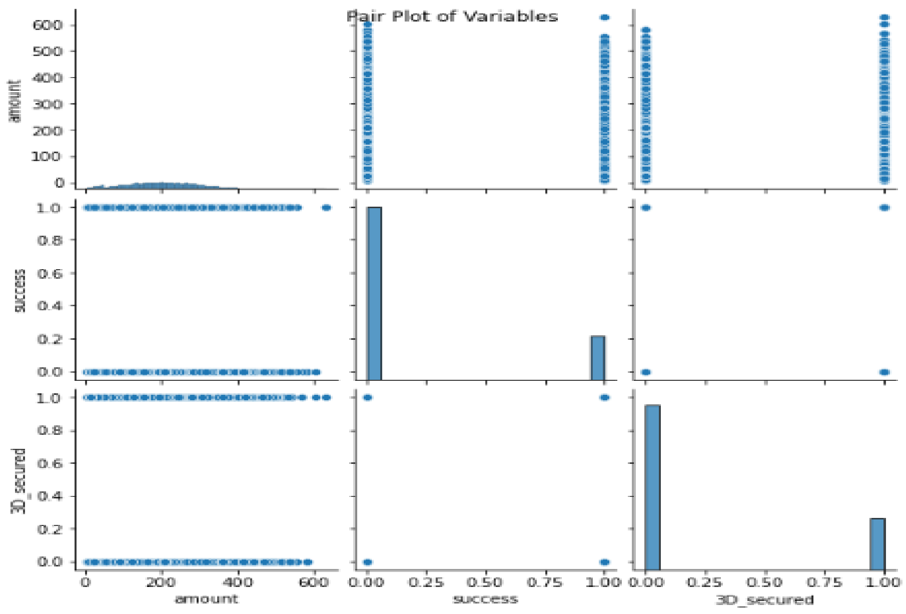


Figure 9: Visualization of relationships between amount, success and 3D_secured using a pair plot

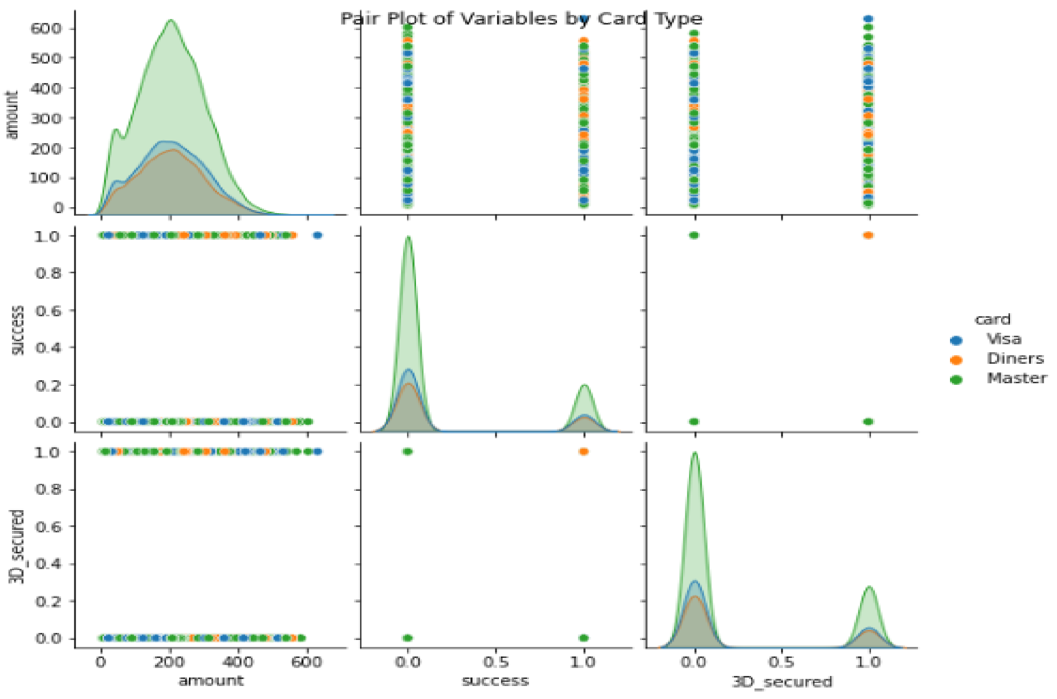


Figure 10: Pair plot including amount, success and 3D_secured for different card types

The interpretation insights from the above Figures 9 and 10 for 'amount vs. success' and 'amount vs. 3D_secured' reveal that specific ranges of transaction amounts may be associated with success/failure and the presence/absence of 3D security. This could be an indication of clear trends in the data.

'Success vs. 3D_secured' having two distinct points underscores that both variables are categorical, with independent values for success (0 or 1) and 3D security (0 or 1).

In addition, the correlation matrix is a (K x K) square and symmetrical matrix whose ij entry represents the correlation between X's columns i and j. Large values in this matrix imply that the variables involved are very collinear (*Correlation Matrix - an Overview | ScienceDirect Topics*, n.d.).

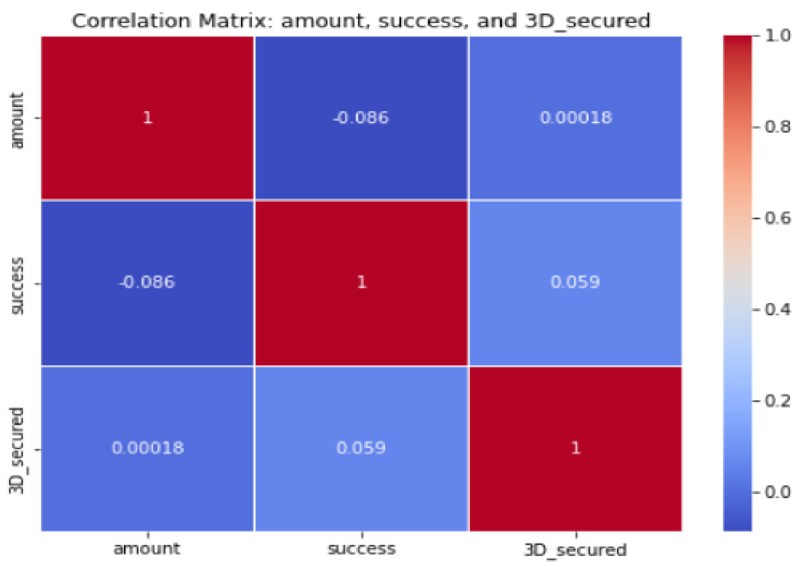


Figure 11: Correlation matrix for amount, success and 3D_secured variables

The weak correlations between these variables in Figure 11 imply that there is no substantial linear link between the transaction amount, transaction success, and the presence of 3D security. These findings highlight the significance of taking into account additional aspects and conducting more in-depth analysis in order to comprehend the complex interactions within the data.
The Success rate by card type bar plot is a critical visualization that provides a clear and concise representation of success rates across different card kinds, providing actionable information and assisting decision-making processes.

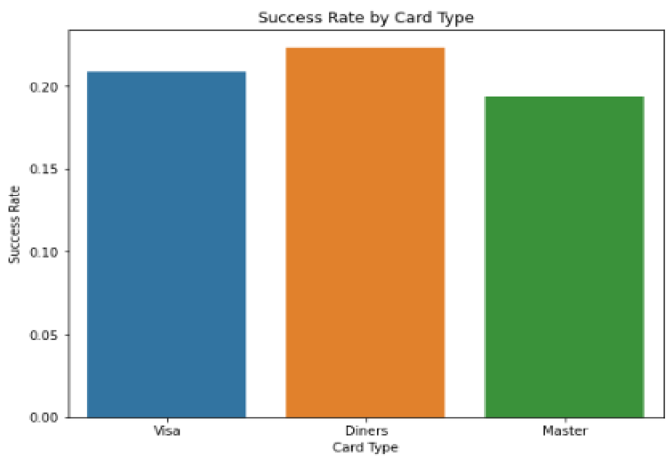


Figure 12: Success rate by card type

The interpretation of the bar graph in Figure 12 shows that success rates vary by card type, with "Dinner" having the highest success rate, followed by "Visa" and then "Master". As a result of the observed hierarchy in success rates among card types, significant insights can be gained that can be used to make informed decisions, optimize processes, and potentially enhance overall transaction success for specific card types.

Time series analysis of transaction data is an effective technique for analysing the temporal behavior of transactions, discovering patterns, and making informed decisions based on previous trends.

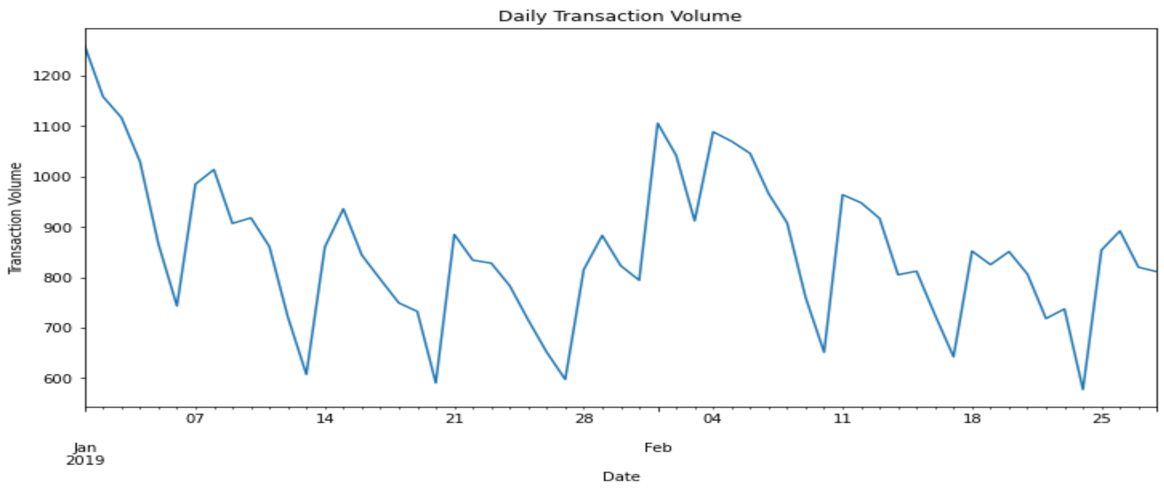


Figure 13: Time series analysis of daily transaction volume

The daily transaction volume graph in the Figure 13 appears as a zig-zag line, indicating that there may not be a distinct trend or seasonality in the daily transaction volume over time. Furthermore, boxplot of transaction amounts by country is an essential visualization that shows how transaction amounts are distributed across countries.

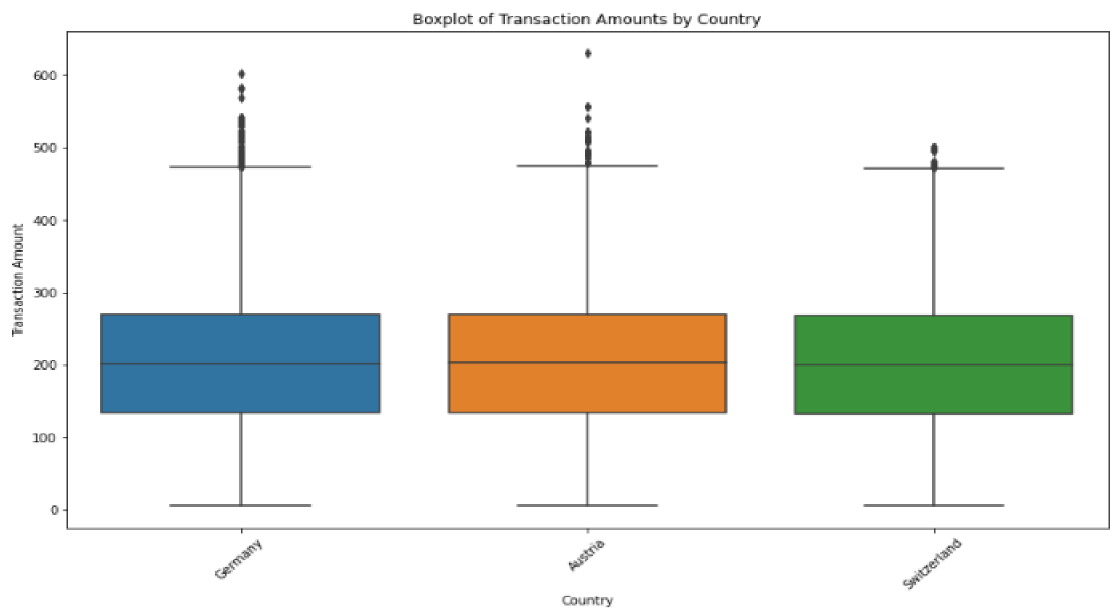


Figure 14: Boxplot of transaction amounts by country

The same medians in Figure 14 reflect equivalent transaction amounts in Germany, Austria, and Switzerland. Differences in the top whisker points show potential variability, with higher points in Germany indicating higher transaction amounts or a wider spread.

The success and failure time series plot is a useful tool for measuring performance, spotting trends, and aiding decision-making processes. It gives a dynamic picture of the system's behavior across time, which aids in the ongoing development of processes and outcomes.

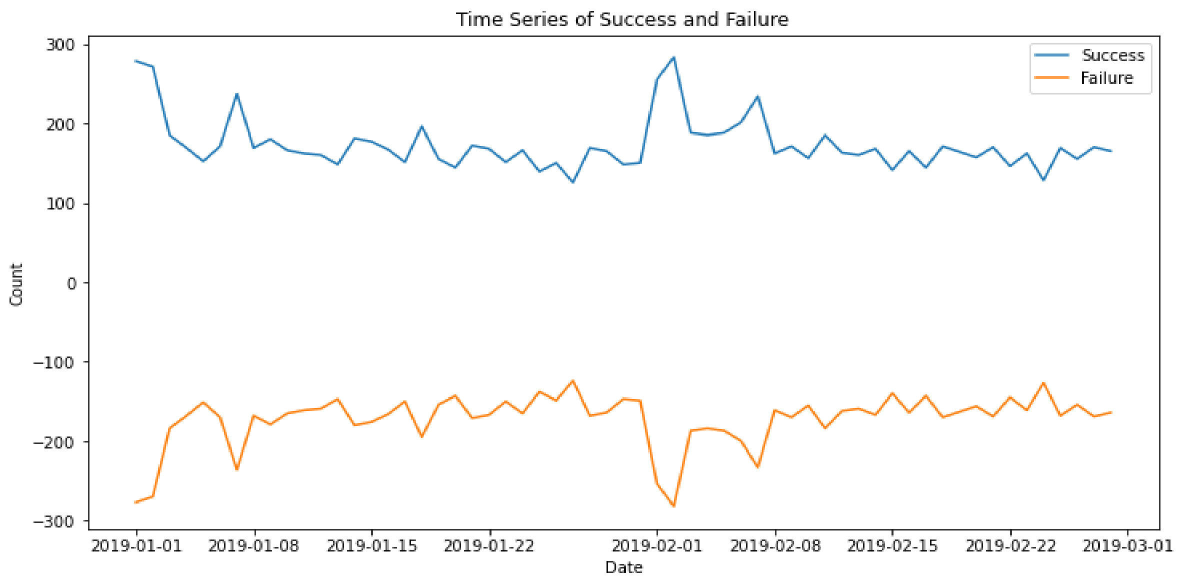


Figure 15: Time series plot of success and failure

The graph in Figure 15 of the time series plot of success and failure could indicate rapid and frequent alternations between success and failure values. This could be related to the nature of our data, in which success and failure occurrences occur often and potentially for brief periods of time. Furthermore, Understanding the temporal dynamics of success and failure in our dataset requires a success rate over time visualization. It is a useful exploratory data analysis tool that can provide actionable insights for decision-making and process improvement.

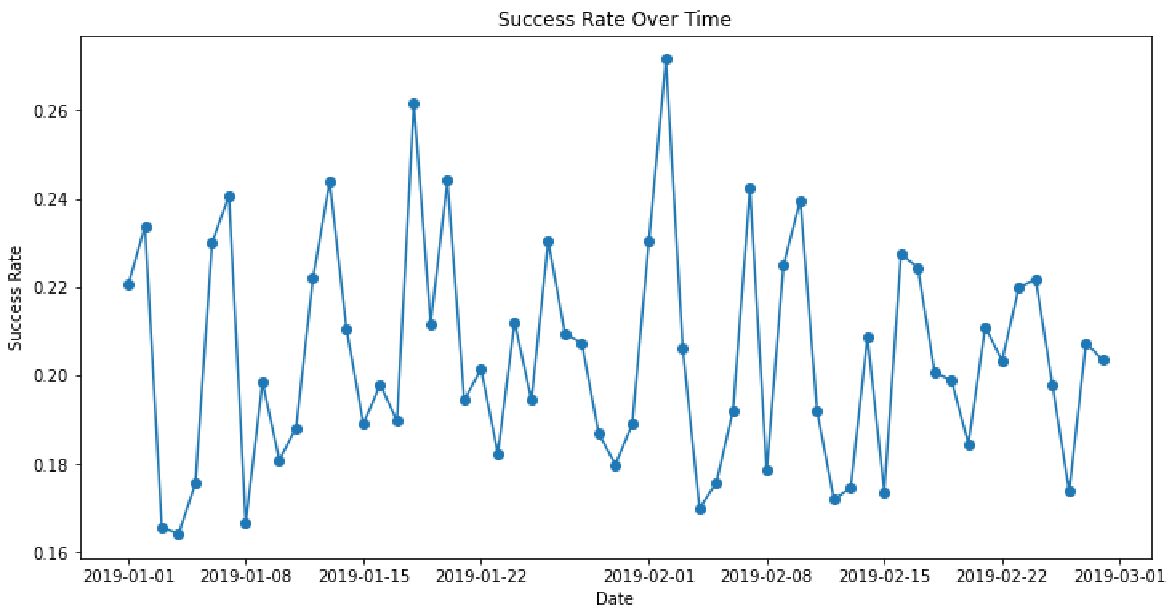


Figure 16: Success rate over time

According to Figure 16, the success rate fluctuates regularly between the dates in the dataset. Each 'o' marker indicates the mean success rate for a certain day, and the lines linking these markers depict the trend over time.

Overall, A preliminary examination of a supplied dataset revealed a right-skewed transaction amount distribution, varied success rates, and categorical 'success' and '3D_secured' variables. Detailed analysis using boxplots, pair plots, and correlation matrices highlighted intricate correlations that necessitated further investigation. Visualizations such as success rates by card type and time series analysis gave useful information on transaction variability and dynamic success rates.

3.3 Data Preparation

The data preparation phase encompasses all operations that result in the final dataset (data that will be supplied into the modelling tool(s)) being constructed from the initial raw data. Data preparation tasks are likely to be completed several times and in no particular order. Table, record, and attribute selection, as well as data transformation and cleansing for modelling tools, are all tasks (*CRISP-DM 1.0. Step-by-Step Data Mining Guide - PDF Free Download*, n.d.). Here are some common activities involved in data preparation:

3.3.1 Data Collection:

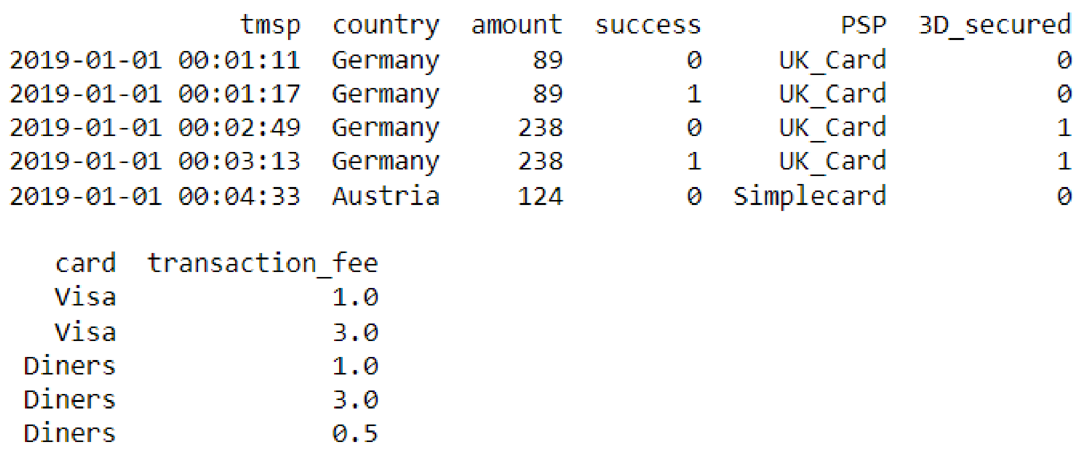
In this context, ensuring that the data obtained covers the required time period and includes all essential variables. As a result, in addition to the provided data, there is a list of payment service providers with service fees that must be collected.

Table 2: List of PSPs and service fees

name	Fee on successful transactions	Fee on failed transactions
Moneycard	5 Euro	2 Euro
Goldcard	10 Euro	5 Euro
UK_Card	3 Euro	1 Euro
Simplecard	1 Euro	0,5 Euro

Furthermore, after gathering payment service providers' service charge data, the relevant fees are mapped to the dataset, resulting in the dataset displayed in Table 3.

Table 3: Mapped PSPs service fee to the dataset by the column name transaction_fee



```
                  tmsp  country  amount  success        PSP  3D_secured
2019-01-01 00:01:11  Germany      89        0    UK_Card           0
2019-01-01 00:01:17  Germany      89        1    UK_Card           0
2019-01-01 00:02:49  Germany     238        0    UK_Card           1
2019-01-01 00:03:13  Germany     238        1    UK_Card           1
2019-01-01 00:04:33  Austria     124        0 Simplecard           0

    card  transaction_fee
    Visa              1.0
    Visa              3.0
  Diners              1.0
  Diners              3.0
  Diners              0.5
```

3.3.2 Data Cleaning:

Data cleaning, also known as data preparation, is an important stage in the data science pipeline that entails discovering and fixing or deleting errors, inconsistencies, and inaccuracies in the data in order to improve its quality and usefulness. Data cleaning is necessary because raw data is frequently noisy, inaccurate, and inconsistent, lowering the accuracy and reliability of the insights obtained from it ("ML | Overview of Data Cleaning," 2018).

The most typical steps in data cleaning are as follows:

a) Detecting and Handling Duplicate Records:
   In the context of the described dataset, duplicate records are rows in which two or more transactions have exactly the same values in the defined columns. However, it is stated in the additional information from the business side that many transactions fail on the first try. As a result, customers attempt to transfer money several times. As a result, for numerous attempts, we will add another

feature to count the multiple attempts in the dataset. So, we handled the duplicate records in this manner.

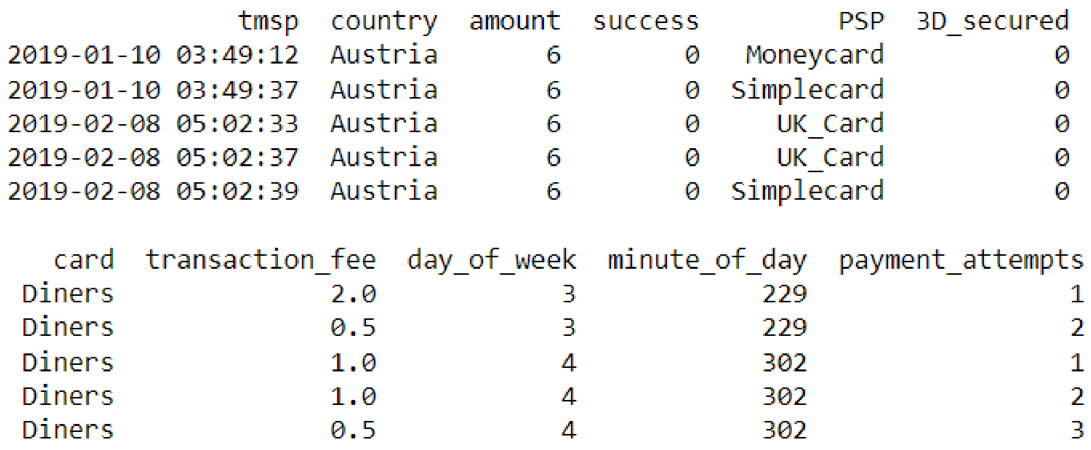
b) Handling Outliers:

Identifying outliers in category variables such as 'country,' 'PSP,' and 'card' does not make as much sense as it does in numerical variables. Outliers are values that differ significantly from the majority of the data. As a result, the numerical column in the provided data is merely the amount, so dealing with outliers is unnecessary.

c) Data Transformation:

Data transformation is the process of changing data from one form to another in order to make it more appropriate for analysis. To transform the data, techniques such as normalization, scaling, or encoding might be utilized ("ML | Overview of Data Cleaning," 2018) column in the dataset must be converted and separated into two independent columns: 'day_of_week' and 'minute_of_day.' The 'day_of_week' column will have values ranging from 0 to 6, reflecting the day of the week for each corresponding timestamp in the 'tmsp' column, where Monday is represented by 0, Tuesday by 1, and so on until Sunday is represented by 6. For each timestamp in the 'tmsp' column, that 'minute_of_day' column will reflect the total number of minutes that have taken place since mid-night. Furthermore, another column "payment_attempts" is added to the dataset to count the many attempts in the dataset, as explained in the detecting and handling duplicate records section. As a result, the modified dataset looks like this:

Table 4: Updated dataset after data transformation



```
                  tmsp  country  amount  success         PSP  3D_secured
2019-01-10 03:49:12  Austria       6        0   Moneycard           0
2019-01-10 03:49:37  Austria       6        0  Simplecard           0
2019-02-08 05:02:33  Austria       6        0     UK_Card           0
2019-02-08 05:02:37  Austria       6        0     UK_Card           0
2019-02-08 05:02:39  Austria       6        0  Simplecard           0

   card  transaction_fee  day_of_week  minute_of_day  payment_attempts
 Diners              2.0            3            229                 1
 Diners              0.5            3            229                 2
 Diners              1.0            4            302                 1
 Diners              1.0            4            302                 2
 Diners              0.5            4            302                 3
```

3.3.3 Feature Selection:

Feature selection methods can be employed in data pre-processing to achieve efficient data reduction. This is useful for locating accurate data models (Jović et al., 2015). To simplify the model and improve its interpretability, remove any unnecessary or redundant features.

Furthermore, the features chosen for analysis in the given dataset include 'country', 'amount', 'success', 'PSP', '3D_secured', 'card', 'transaction_fee', 'day_of_week', 'minute_of_day' and 'payment_attempts'. The 'tmsp' column, which represents timestamps, has been excluded out of the investigation since it wasn't utilized by the features that were chosen. This feature selection technique seeks to focus on essential features while removing timestamp information, which is modified and split into two different columns: 'day_of_week' and 'minute_of_day'. As a result, the modified dataset looks like this:

Table 5: Updated dataset after feature selection

```
country  amount  success         PSP  3D_secured    card  transaction_fee
Austria       6        0   Moneycard           0  Diners              2.0
Austria       6        0  Simplecard           0  Diners              0.5
Austria       6        0     UK_Card           0  Diners              1.0
Austria       6        0     UK_Card           0  Diners              1.0
Austria       6        0  Simplecard           0  Diners              0.5

day_of_week  minute_of_day  payment_attempts
          3            229                 1
          3            229                 2
          4            302                 1
          4            302                 2
          4            302                 3
```

3.3.4 Handling Imbalanced Data:

In many real-world problems, the data sets are often imbalanced, i.e., some classes have far more instances than others. Imbalance has a major impact on classifier performance. Learning algorithms that do not account for class imbalance are likely to be overwhelmed by the dominant class and overlook the minority class (Liu et al., 2009). As a result, in our dataset context, class imbalance is addressed using one-hot encoding for categorical features ('country,' 'PSP,' 'card') and standard scaling for numerical features ('amount', 'minute_of_day', 'day_of_week', '3D_secured', 'payment_attempts'). Additionally, SMOTE generates synthetic samples for the minority class using imbalanced-learn. These strategies work together to provide a more balanced dataset, which improves machine learning model performance, especially in circumstances with considerable class imbalance.

3.3.5 Data Splitting:

One of the most important criteria in machine learning is the ability to develop computational models with excellent prediction and generalization capabilities. A computational model is trained to predict the outputs of an unknown target function in the case of supervised learning (Reitermanova, 2010). Furthermore, the 80-10-10 rule, which allocates 80% of the data to the training set, 10% to the validation set for hyperparameter tuning and model evaluation, and the remaining 10% to the test set for assessing the final model's performance on unknown data, is a typical and commonly used data split. As a result, we split the dataset using the 80-10-10 rule.

3.3.6 Detailed Data Quality Assessment:

Detailed data quality assessment in the data preparation stage entails numerous processes to check for inconsistencies, duplications, and anomalies and to evaluate variable distribution before using the finalized data for building the model.

a) Overview of the newly added data columns:

The modified dataset following feature selection presented in Table 5 comprises 'transaction_fee', 'day_of_week','minute_of_day', and 'payment_attempts'. As a result, the statistics summary for transaction_fee and its distribution is as displayed in Table 6:

Table 6: Statistical summary of transaction_fee column

```
                   count      mean       std  min  25%  50%  75%   max
transaction_fee  50410.0  1.756477  1.814051  0.5  1.0  1.0  2.0  10.0
```

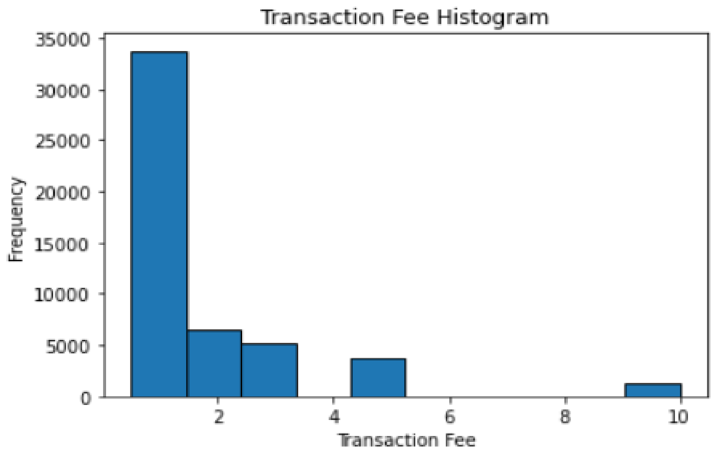


Figure 17: Histogram of transaction fee

In addition, for columns like 'day_of_week', 'minute_of_day,' and 'payment_attempts,' we may want to concentrate on understanding the distribution rather than standard statistical summarization. Histograms, for example, may be more informative.

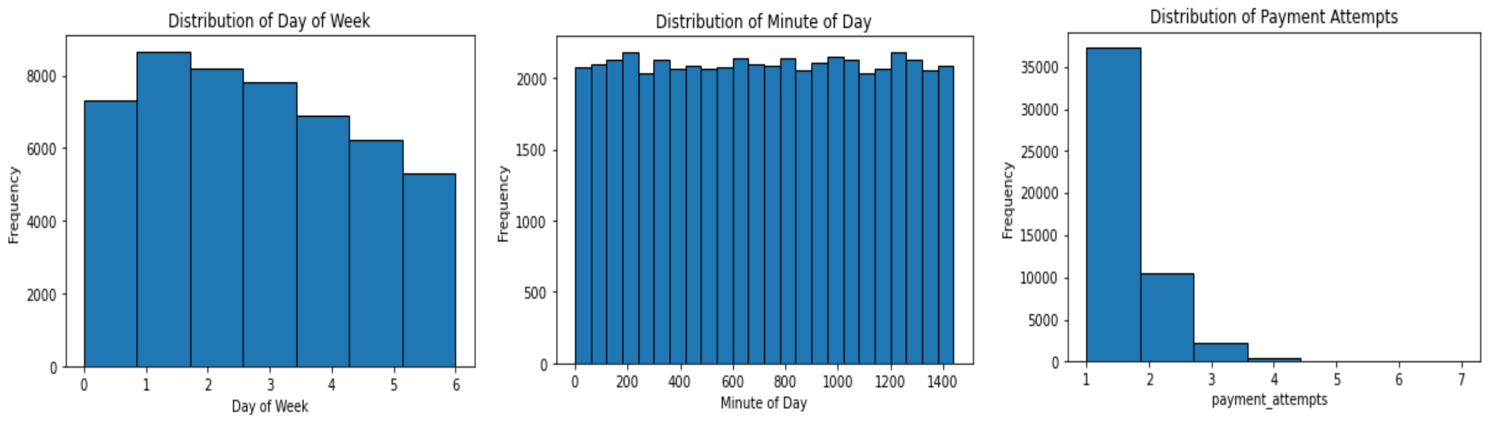


Figure 18: Distribution of day of week, minute of day and payment attempts columns

According to the code output and histograms presented in Figure 18, the most transactions occur on Tuesday, followed by Wednesday, Thursday, Monday, Friday, Saturday, and Sunday. Furthermore, a histogram for the distribution of minutes of the day offers information on which transactions are more frequent throughout the course of a 24-hour period. Furthermore, there are 37,227 entries with one payment attempt, 10,463 entries with two attempts, 2,228 entries with three attempts, 412 entries with four attempts, 67 entries with five attempts, ten entries with six attempts, and three entries with seven attempts.

b) Check for Inconsistencies:

Examining unique values in category columns (country, PSP, card) for unexpected or inconsistent values. As a result, based on the code output, unique values appear reasonable, and there do not appear to be any unexpected values or inconsistencies in these columns.

c) Evaluate Numeric Variables:

Evaluating numeric variables (amount, transaction_fee) using box plots, and below is the box plot.

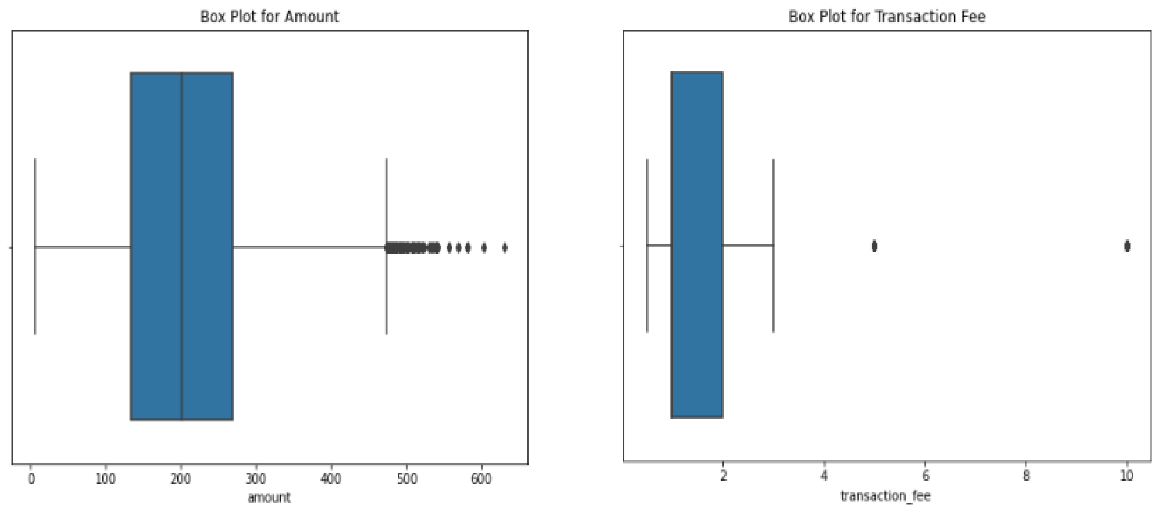


Figure 19: Boxplot of amount and transaction fee column

As a result, according to the boxplot output in Figure 19, the 'amount' has a right-skewed distribution, with higher values above the top whisker, indicating a prevalence of larger transactions. 'transaction_fee', on the other hand, has fewer higher values above the upper whisker, indicating a preference for smaller transaction costs.

d) Evaluate Categorical Variables:

Assessing the distribution of categorical variables (country, PSP, card) is necessary to ensure an appropriate distribution of categories. Checking for unusual categories and deciding whether to group or leave them alone. So, as the code output indicates that there are no rare categories in this situation, we may not need to execute additional grouping for these variables. The absence of unusual categories is a positive finding because it suggests a reasonable distribution of categories in our collection.

e) Correlation Analysis:

The correlation analysis of the modified dataset is shown below:

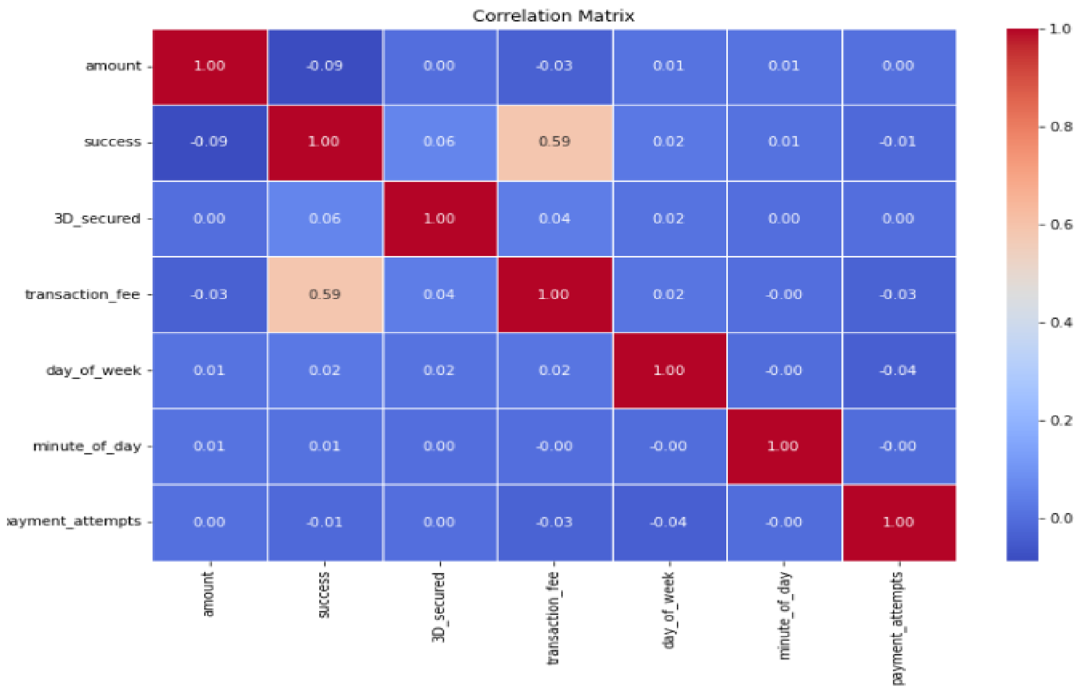


Figure 20: Correlation analysis of updated dataset

A correlation coefficient of 0.59 indicates a somewhat favorable link between 'transaction_fee' and 'success.' As 'transaction_fee' rises, there is a tendency for 'success' to also increase. However, the connection strength is not exceptionally strong. Furthermore, there is no strong correlation between the other variables.

3.4 Modeling

The fourth phase of CRISP-DM is data modeling. This is the stage of the project in which we will apply a mathematical or visual model to the data in order to complete a task, answer a question, or solve a specific problem (Conroy, 2023).

The modelling step consists of four tasks. These are the (*Phase 4 of the CRISP-DM Process Model*, n.d.):

3.4.1 Selecting Modeling Techniques:

In this stage, we will begin model development by creating a baseline model and then developing models using two linear model techniques, Logistic Regression and Support Vector Machines (SVM), and one non-linear model technique, Random Forest Classification, with the goal of selecting the best model.

3.4.2 Designing Test:

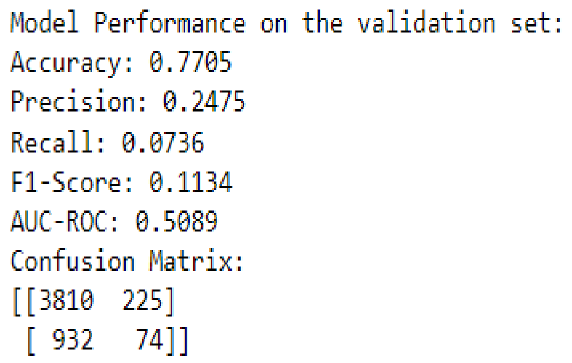
The test in this task is the one we'll use to see how well our model performs. It might be as simple as dividing our data into two groups: cases for model training and cases for model testing (*Phase 4 of the CRISP-DM Process Model*, n.d.). As a result, we are using the 80-10-10 rule to split the dataset, as mentioned in the data splitting section 3.3.5, where we are using 80% of the data for training, 10% for validation and model evaluation, and the remaining 10% for testing the final model's performance on unseen data.

3.4.3 Building Model(s):

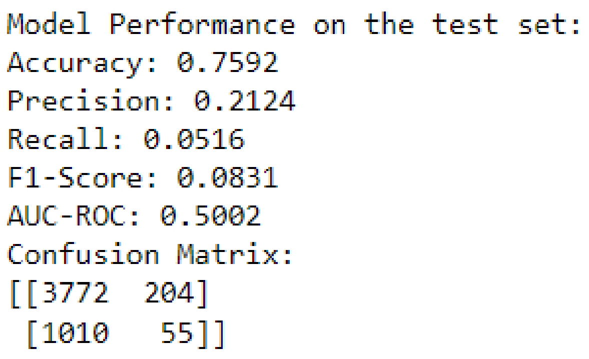
At this step, we build a baseline model as well as accurate prediction models that meet the business needs outlined in the project plan. First, we are developing Model 1 for predicting success probability and giving its outputs to Model 2. Second, Model 2 dynamically selects the best payment service provider (PSP) based on success probabilities and transaction fees, allowing for cost-effective decision-making.

a) Baseline Model 1:

A baseline model is a simple model that is used to measure the effect and possibly justify the adoption of a more complex model (Conroy, 2023). Typically, a baseline model is created before dealing with data imbalances. As a result, the K-Nearest Neighbors (KNN) technique can be used as a baseline model. Because of its simplicity and ease of implementation, it is a strong choice for a baseline model. The following is the baseline model performance Output 1 before dealing with data imbalances, according to the code output:



```
Model Performance on the validation set:
Accuracy: 0.7705
Precision: 0.2475
Recall: 0.0736
F1-Score: 0.1134
AUC-ROC: 0.5089
Confusion Matrix:
[[3810  225]
 [ 932   74]]
```



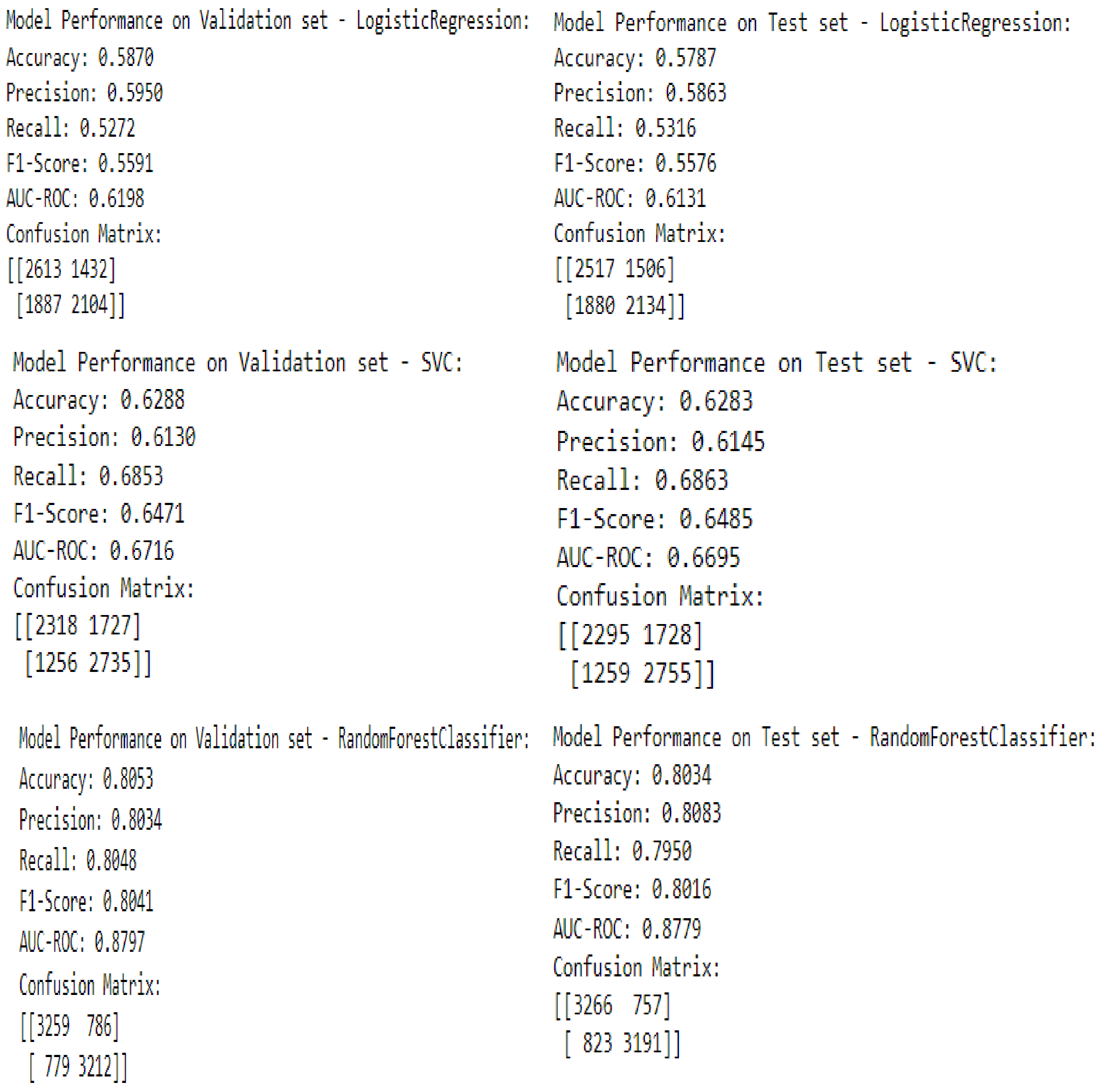
```
Model Performance on the test set:
Accuracy: 0.7592
Precision: 0.2124
Recall: 0.0516
F1-Score: 0.0831
AUC-ROC: 0.5002
Confusion Matrix:
[[3772  204]
 [1010   55]]
```

Output 1: Baseline model 1 performance output on validation and test set

In summary, both sets exhibit poor model performance, with low precision and recall. The AUC-ROC values indicate that the model is close to random guessing. Additional enhancements or different modelling methodologies may be required.

b) Development of Model 1:

Model 1's goal is to predict success probabilities. This model employs given transaction-related features (excluding transaction_fee) as the target variable, with success as the outcome. As a result, Model 1 was used to calculate success probabilities. We used two linear model techniques, Logistic Regression and Support Vector Machines (SVM), and one non-linear model technique, Random Forest Classification, to develop Model 1. The data imbalance is handled before building Model 1, as mentioned in section 3.3.4. As a result, the model's output is as shown in Output 2:



```
Model Performance on Validation set - LogisticRegression:
Accuracy: 0.5870
Precision: 0.5950
Recall: 0.5272
F1-Score: 0.5591
AUC-ROC: 0.6198
Confusion Matrix:
[[2613 1432]
 [1887 2104]]
```

```
Model Performance on Test set - LogisticRegression:
Accuracy: 0.5787
Precision: 0.5863
Recall: 0.5316
F1-Score: 0.5576
AUC-ROC: 0.6131
Confusion Matrix:
[[2517 1506]
 [1880 2134]]
```

```
Model Performance on Validation set - SVC:
Accuracy: 0.6288
Precision: 0.6130
Recall: 0.6853
F1-Score: 0.6471
AUC-ROC: 0.6716
Confusion Matrix:
[[2318 1727]
 [1256 2735]]
```

```
Model Performance on Test set - SVC:
Accuracy: 0.6283
Precision: 0.6145
Recall: 0.6863
F1-Score: 0.6485
AUC-ROC: 0.6695
Confusion Matrix:
[[2295 1728]
 [1259 2755]]
```

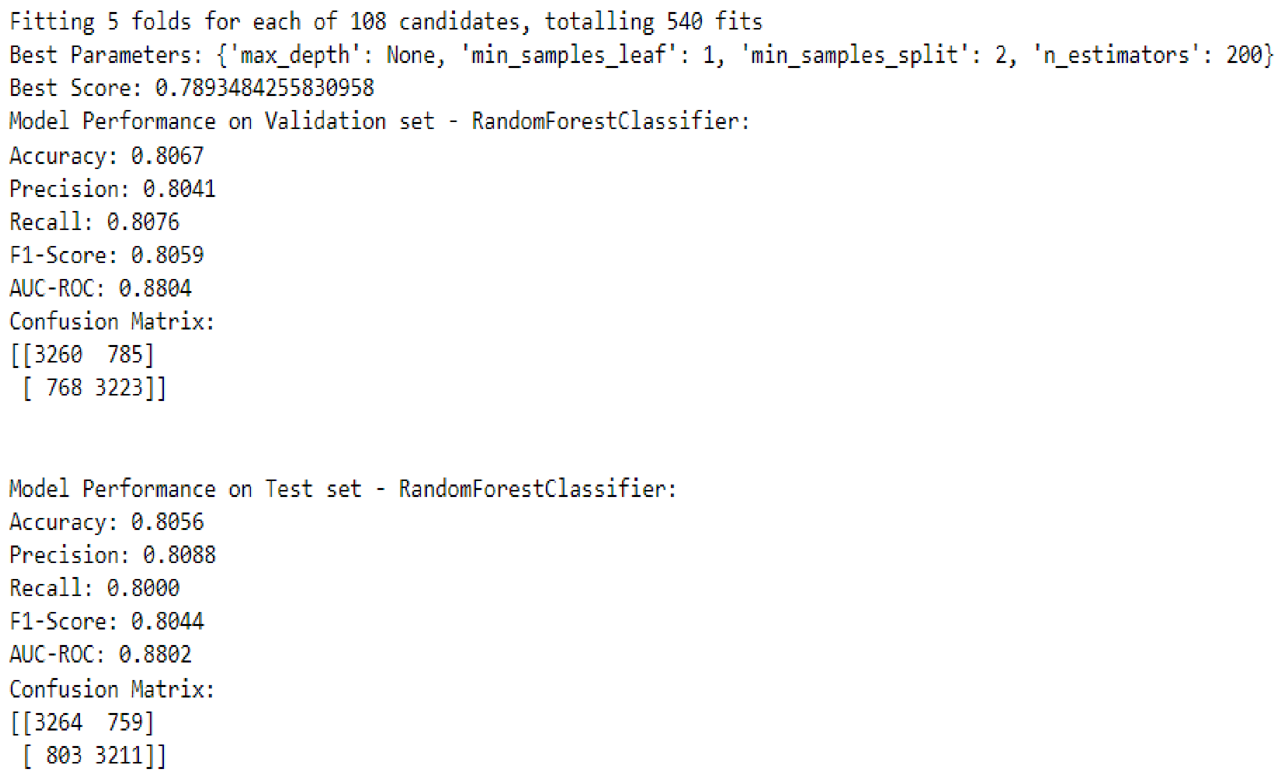
```
Model Performance on Validation set - RandomForestClassifier:
Accuracy: 0.8053
Precision: 0.8034
Recall: 0.8048
F1-Score: 0.8041
AUC-ROC: 0.8797
Confusion Matrix:
[[3259  786]
 [ 779 3212]]
```

```
Model Performance on Test set - RandomForestClassifier:
Accuracy: 0.8034
Precision: 0.8083
Recall: 0.7950
F1-Score: 0.8016
AUC-ROC: 0.8779
Confusion Matrix:
[[3266  757]
 [ 823 3191]]
```

Output 2: Model performance output on the validation and test sets of Model 1

According to the aforementioned results, the Random Forest Classifier outperforms Logistic Regression and SVC on both validation and test sets, while Logistic Regression and SVC have comparable metrics. Models perform marginally better on the validation set, indicating possible overfitting. Choosing the Random Forest Classifier because of its superior performance on both validation and test sets and enhancing its robustness through cross-validation and hyperparameter tuning because cross-validation provides a more robust estimate of a model's performance than a single train-test split, and hyperparameter tuning helps to find the optimal set of hyperparameters for a model. We can develop a more accurate and trustworthy machine learning model by employing these strategies (Singh, 2023).

Furthermore, Random Forest Classifier uses GridSearchCV from scikit-learn for hyperparameter tuning and cross-validation, printing the best hyperparameters and accuracy score. On validation and test sets, the best model is retrieved and evaluated. As indicated in Output 3, the output after hyperparameter tuning and cross-validation is as follows:



```
Fitting 5 folds for each of 108 candidates, totalling 540 fits
Best Parameters: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200}
Best Score: 0.7893484255830958
Model Performance on Validation set - RandomForestClassifier:
Accuracy: 0.8067
Precision: 0.8041
Recall: 0.8076
F1-Score: 0.8059
AUC-ROC: 0.8804
Confusion Matrix:
[[3260  785]
 [ 768 3223]]


Model Performance on Test set - RandomForestClassifier:
Accuracy: 0.8056
Precision: 0.8088
Recall: 0.8000
F1-Score: 0.8044
AUC-ROC: 0.8802
Confusion Matrix:
[[3264  759]
 [ 803 3211]]
```

Output 3: Model 1 performance output on validation and test set after hyperparameter tuning and cross-validation

According to Output 3, hyperparameter adjustment significantly improved Random Forest Classifier metrics (Model 1) such as F1-Score, Recall, and AUC-ROC on both validation and test sets. The modified model demonstrated set consistency, implying robust generalization. Confusion matrices reduced erroneous positives and negatives, improving total predictive accuracy. Optimal hyperparameters such as no maximum depth, one sample per leaf, two samples per split, and 200 trees were critical in obtaining reported performance increases.

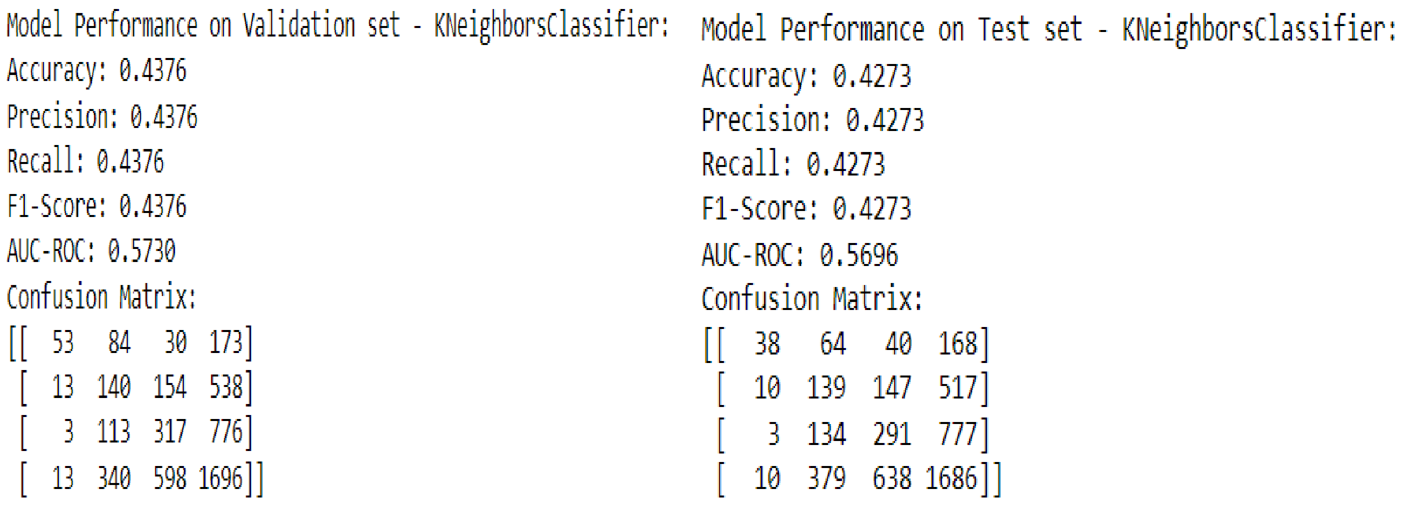
Furthermore, after hyperparameter tuning and cross-validation, we are now predicting success probabilities using the trained best Random Forest Classifier model (best_rf_model). Success probabilities are predicted for the original data and appended to the data frame as 'success_probabilities'. Finally, the modified DataFrame is provided in Table 7, together with the original features and predicted success probabilities.

Table 7: Updated DataFrame after predicting success probabilities using Random Forest Classifier Model 1

```
country  amount  success         PSP  3D_secured    card  transaction_fee
Austria       6        0   Moneycard           0  Diners              2.0
Austria       6        0  Simplecard           0  Diners              0.5
Austria       6        0     UK_Card           0  Diners              1.0
Austria       6        0     UK_Card           0  Diners              1.0
Austria       6        0  Simplecard           0  Diners              0.5

day_of_week  minute_of_day  payment_attempts  success_probabilities
          3            229                 1                  0.450
          3            229                 2                  0.425
          4            302                 1                  0.450
          4            302                 2                  0.420
          4            302                 3                  0.410
```

c) Baseline Model 2:

In this stage also, before the development of Model 2, the baseline Model 2 is build using the KNN algorithm, and below is the model performance output as shown in Output 4:



```
Model Performance on Validation set - KNeighborsClassifier:
Accuracy: 0.4376
Precision: 0.4376
Recall: 0.4376
F1-Score: 0.4376
AUC-ROC: 0.5730
Confusion Matrix:
[[  53   84   30  173]
 [  13  140  154  538]
 [   3  113  317  776]
 [  13  340  598 1696]]
```

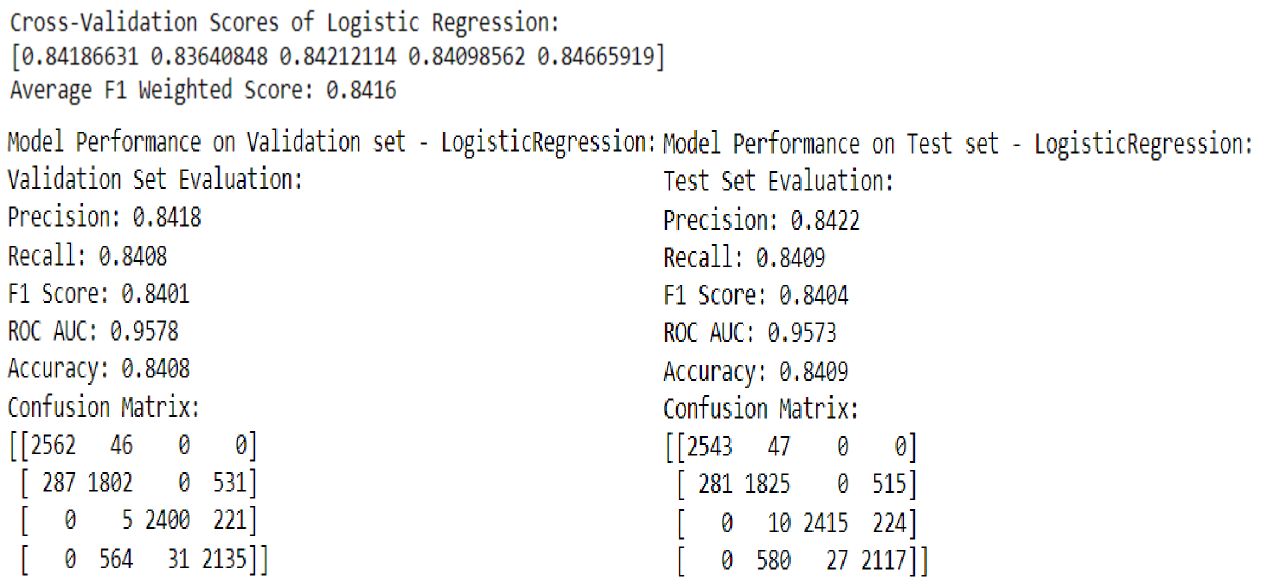
```
Model Performance on Test set - KNeighborsClassifier:
Accuracy: 0.4273
Precision: 0.4273
Recall: 0.4273
F1-Score: 0.4273
AUC-ROC: 0.5696
Confusion Matrix:
[[  38   64   40  168]
 [  10  139  147  517]
 [   3  134  291  777]
 [  10  379  638 1686]]
```

Output 4: Baseline Model 2 performance output on validation and test sets

On both validation and test sets, the baseline K-Neighbors classifier Model 2 performs well, with an Accuracy, Precision, Recall, and F1-score of roughly 43.76%. The consistency of sets shows sustained generalization. However, the AUC-ROC values of roughly 57% indicate that there is space for improvement in class discrimination. In subsequent iterations, we should investigate alternate techniques for improved model performance.

d) Development of Model 2:

The goal of building Model 2 is to predict an optimal PSP based on the success probabilities from Model 1 prediction. Additional transaction fees and features that are relevant. In the development of Model 2, we chose two linear model techniques, Logistic Regression and Support Vector Machines (SVM), and one non-linear model technique. Random Forest Classification is used, and the data imbalance is addressed as discussed in section 3.3.4. As a result, the model's performance output using cross-validation is as follows:

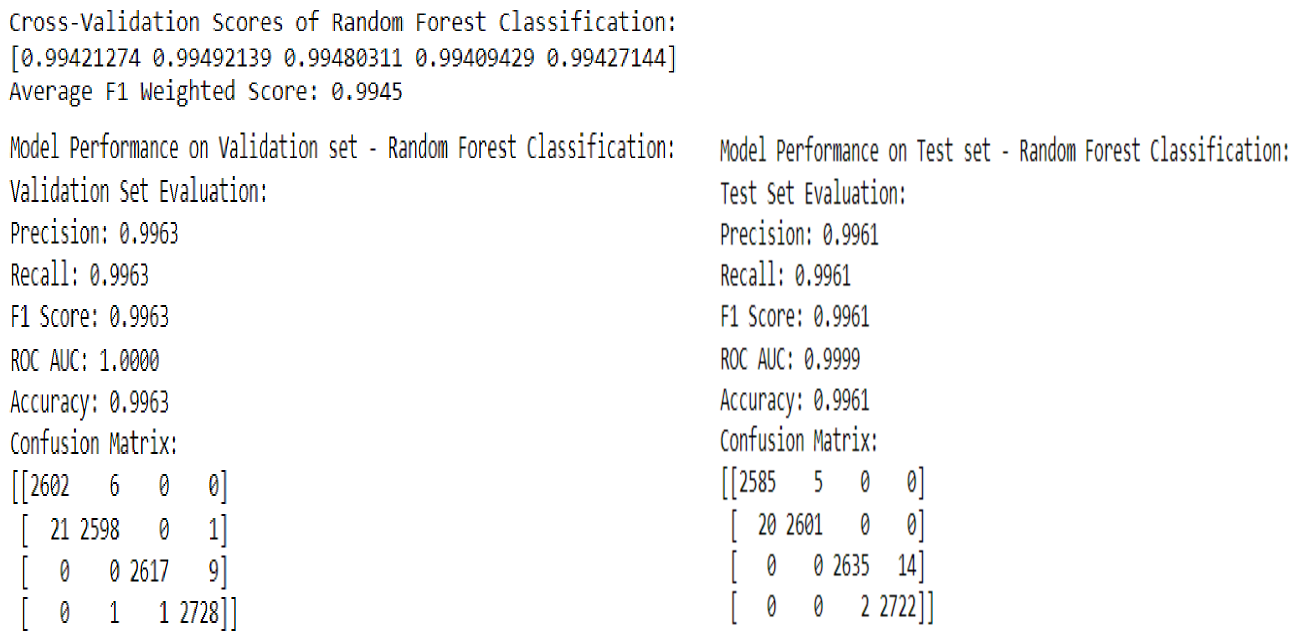


```
Cross-Validation Scores of Logistic Regression:
[0.84186631 0.83640848 0.84212114 0.84098562 0.84665919]
Average F1 Weighted Score: 0.8416
```

```
Model Performance on Validation set - LogisticRegression:
Validation Set Evaluation:
Precision: 0.8418
Recall: 0.8408
F1 Score: 0.8401
ROC AUC: 0.9578
Accuracy: 0.8408
Confusion Matrix:
[[2562   46    0    0]
 [ 287 1802    0  531]
 [   0    5 2400  221]
 [   0  564   31 2135]]
```

```
Model Performance on Test set - LogisticRegression:
Test Set Evaluation:
Precision: 0.8422
Recall: 0.8409
F1 Score: 0.8404
ROC AUC: 0.9573
Accuracy: 0.8409
Confusion Matrix:
[[2543   47    0    0]
 [ 281 1825    0  515]
 [   0   10 2415  224]
 [   0  580   27 2117]]
```

Output 5: Logistic Regression model performance output with cross-validation on validation and test set of Model 2

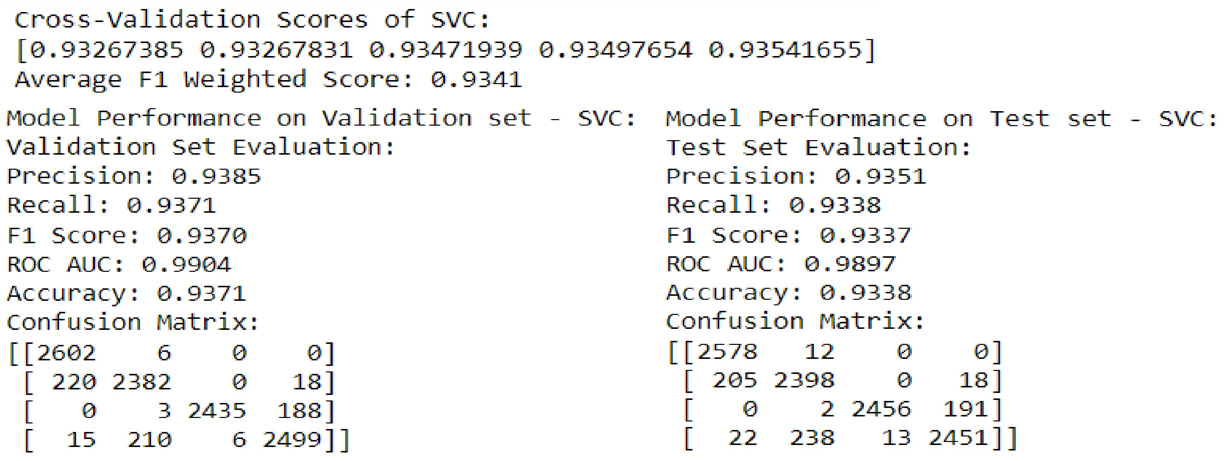


```
Cross-Validation Scores of Random Forest Classification:
[0.99421274 0.99492139 0.99480311 0.99409429 0.99427144]
Average F1 Weighted Score: 0.9945
```

```
Model Performance on Validation set - Random Forest Classification:
Validation Set Evaluation:
Precision: 0.9963
Recall: 0.9963
F1 Score: 0.9963
ROC AUC: 1.0000
Accuracy: 0.9963
Confusion Matrix:
[[2602    6    0    0]
 [  21 2598    0    1]
 [   0    0 2617    9]
 [   0    1    1 2728]]
```

```
Model Performance on Test set - Random Forest Classification:
Test Set Evaluation:
Precision: 0.9961
Recall: 0.9961
F1 Score: 0.9961
ROC AUC: 0.9999
Accuracy: 0.9961
Confusion Matrix:
[[2585    5    0    0]
 [  20 2601    0    0]
 [   0    0 2635   14]
 [   0    0    2 2722]]
```

Output 6: Random Forest Classification model performance output with cross-validation on validation and test set of Model 2



```
Cross-Validation Scores of SVC:
[0.93267385 0.93267831 0.93471939 0.93497654 0.93541655]
Average F1 Weighted Score: 0.9341
```

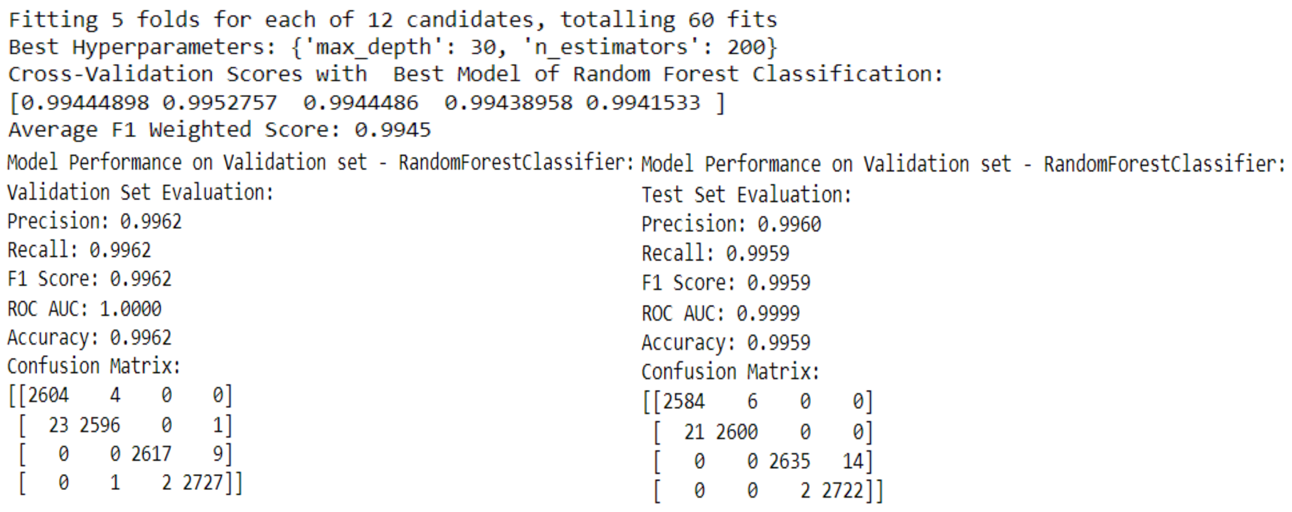
```
Model Performance on Validation set - SVC:
Validation Set Evaluation:
Precision: 0.9385
Recall: 0.9371
F1 Score: 0.9370
ROC AUC: 0.9904
Accuracy: 0.9371
Confusion Matrix:
[[2602    6    0    0]
 [ 220 2382    0   18]
 [   0    3 2435  188]
 [  15  210    6 2499]]
```

```
Model Performance on Test set - SVC:
Test Set Evaluation:
Precision: 0.9351
Recall: 0.9338
F1 Score: 0.9337
ROC AUC: 0.9897
Accuracy: 0.9338
Confusion Matrix:
[[2578   12    0    0]
 [ 205 2398    0   18]
 [   0    2 2456  191]
 [  22  238   13 2451]]
```

Output 7: Support Vector Machines model performance output with cross-validation on validation and test set of Model 2

The classification models perform well across multiple metrics, according to Outputs 5, 6, and 7. Logistic Regression performs well in all areas, with high Precision, Recall, and F1 score. Random Forest Classification performs extremely well in terms of Precision, Recall, and F1 score, demonstrating its robust and accurate predictive skills. Support Vector Machines perform well as well, with good Precision, Recall, and F1 score, indicating their competence in identifying cases. Overall, Random Forest Classification is chosen for predicting PSPs because it demonstrates consistent and competitive performance, each with its own set of strengths, making them excellent choices for our specific use-case goals and requirements.

Furthermore, the Random Forest Classification model is optimized using GridSearchCV hyperparameter tuning, which explores combinations of the number of estimators and maximum tree depth. For robust performance evaluation, the optimal model is chosen based on the F1-weighted score, trained on the complete set, and tested on both the validation and test sets. As a result, the Model 2 outputs are as follows, as shown in Output 8.



```
Fitting 5 folds for each of 12 candidates, totalling 60 fits
Best Hyperparameters: {'max_depth': 30, 'n_estimators': 200}
Cross-Validation Scores with  Best Model of Random Forest Classification:
[0.99444898 0.9952757  0.9944486  0.99438958 0.9941533 ]
Average F1 Weighted Score: 0.9945
```

```
Model Performance on Validation set - RandomForestClassifier:
Validation Set Evaluation:
Precision: 0.9962
Recall: 0.9962
F1 Score: 0.9962
ROC AUC: 1.0000
Accuracy: 0.9962
Confusion Matrix:
[[2604    4    0    0]
 [  23 2596    0    1]
 [   0    0 2617    9]
 [   0    1    2 2727]]
```

```
Model Performance on Validation set - RandomForestClassifier:
Test Set Evaluation:
Precision: 0.9960
Recall: 0.9959
F1 Score: 0.9959
ROC AUC: 0.9999
Accuracy: 0.9959
Confusion Matrix:
[[2584    6    0    0]
 [  21 2600    0    0]
 [   0    0 2635   14]
 [   0    0    2 2722]]
```

Output 8: Model performance on validation and test sets of Model 2 after hyperparameter tuning and cross-validation

Based on the Output 8, hyperparameter tuning enhanced the Model 2, resulting in minor F1 weighted score improvements. With 'max_depth' 30 and 'n_estimators' 200, the optimized model exhibits outstanding stability and generalizability across validation and test sets. While the initial Model 2 functioned well, fine-tuning improved it just a little.

3.4.4 Assessing model(s):

We will analyse the models that we have developed, both technically and business (sometimes with assistance from business specialists on our project team). We evaluated the models using a variety of metrics, including Accuracy, Precision, Recall, F1 Score, ROC AUC, and Confusion Matrices (Wardhani et al., 2019). Logistic Regression, Random Forest Classification, and Support Vector Machines have been evaluated for Model 1, which predicts success probabilities. Random Forest Classification outperformed on both the validation and test sets. The Random Forest Classification algorithm is implemented as Model 1 for predicting success probabilities after further hyperparameter tuning enhanced its metrics, showing strong generalization.

In Model 2, Logistic Regression, Random Forest Classification, and Support Vector Machines were used to determine the best Payment Service Provider (PSP). Random Forest Classification outperformed the other models once more, with high Precision, Recall, and F1 Score. The Random Forest Classification technique is implemented as Model 2 for predicting PSPs after hyperparameter optimization improved its performance while retaining consistency across validation and test sets.

The major goal from a business standpoint is to automate credit card routing, increase payment success rates, and reduce transaction fees. The chosen models, particularly Random Forest Classification after hyperparameter tuning, exhibit robust and competitive performances that are in-line with the business objectives. The Random Forest model was chosen because of its capacity to balance success probability and transaction fees, which contribute to cost-effective decision-making. The model's stability, generalizability, and enhanced forecast accuracy make it a good fit for use in business workflows.

3.5 Evaluation:

At this point in the study, we developed models that appear to be of excellent quality based on data analysis. Before proceeding with the final deployment of the model, it is critical to thoroughly examine the model and review the procedures used to develop the model to ensure that it meets the business objectives. One significant goal is to identify whether or not an important business problem has been adequately examined (Nadali et al., 2011).

Let us now look at the important areas of evaluation under the CRISP-DM framework:

3.5.1 Discussion of Individual Features:

Identifying and discussing the significance of particular features is critical for business goals. Highlighting each feature's contribution fosters transparency and builds stakeholder trust. This method connects predictions with business goals, improving knowledge of the model's functionality. In our use case, understanding why Model 1 and Model 2 produce a particular prediction can be as important as the prediction's accuracy. However, getting the best accuracy for huge current datasets is frequently difficult due to complex models that experts struggle to grasp. In response, several strategies have recently been presented to assist users in interpreting the predictions of complex models. It is frequently unclear how various strategies are related and whether one way is superior to another. To solve this issue, we propose SHAP (SHapley Additive exPlanations), a unified framework for interpreting predictions. SHAP offers an importance rating for each feature for a specific prediction. Its novel components include (1) the identification of a new class of additive feature significance measures and (2) theoretical studies demonstrating the existence of a single solution in this class with a set of desirable qualities (Lundberg & Lee, 2017). We created a bar graph using SHAP to highlight the feature relevance for Models 1 and 2 versus the SHAP value.

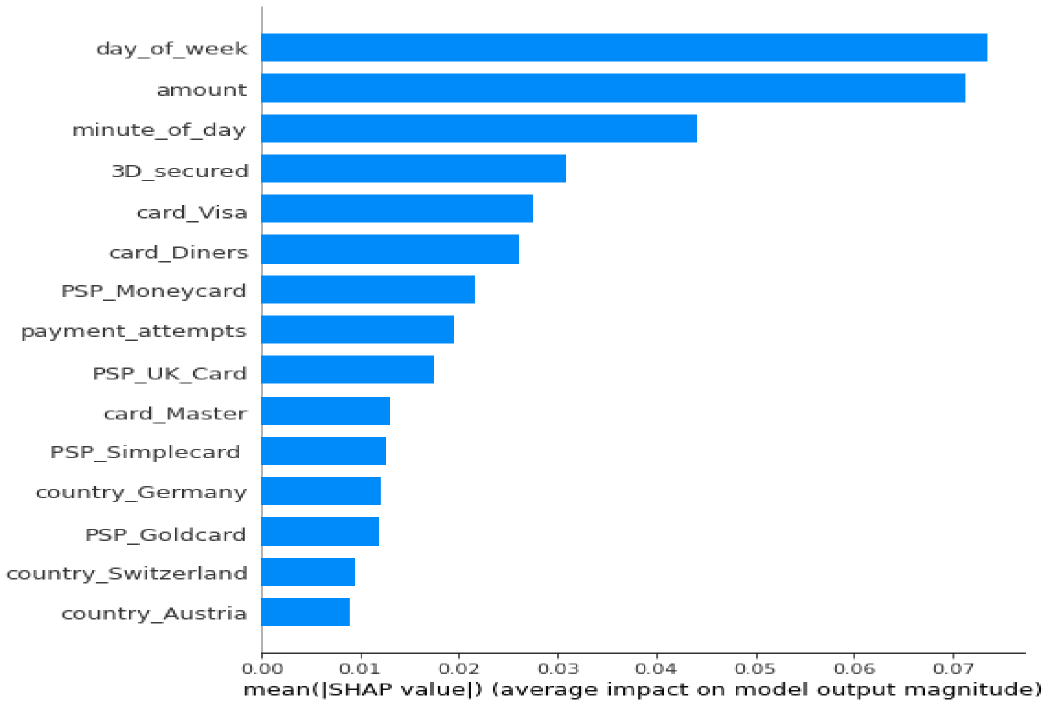


Figure 21: Summary bar plot of the importance of individual features in Model 1

From the above Figure 21, the SHAP analysis shows "day_of_week" as a critical variable with a SHAP value greater than 0.07, demonstrating its strong impact on Model 1 predictions. "amount", "minute_of_day" and "3D_secured" all contribute significantly. Payment-related variables ("card_Visa", "card_Diners", "PSP_Moneycard" and so on) emphasize the relevance of transaction-specific properties, whereas geographical elements ("country_Germany", "country_Switzerland", "country_Austria") emphasize regional importance in determining Model 1 predictions. Overall, our findings highlight the crucial impact of temporal, transactional, and spatial factors in shaping Model 1 results.

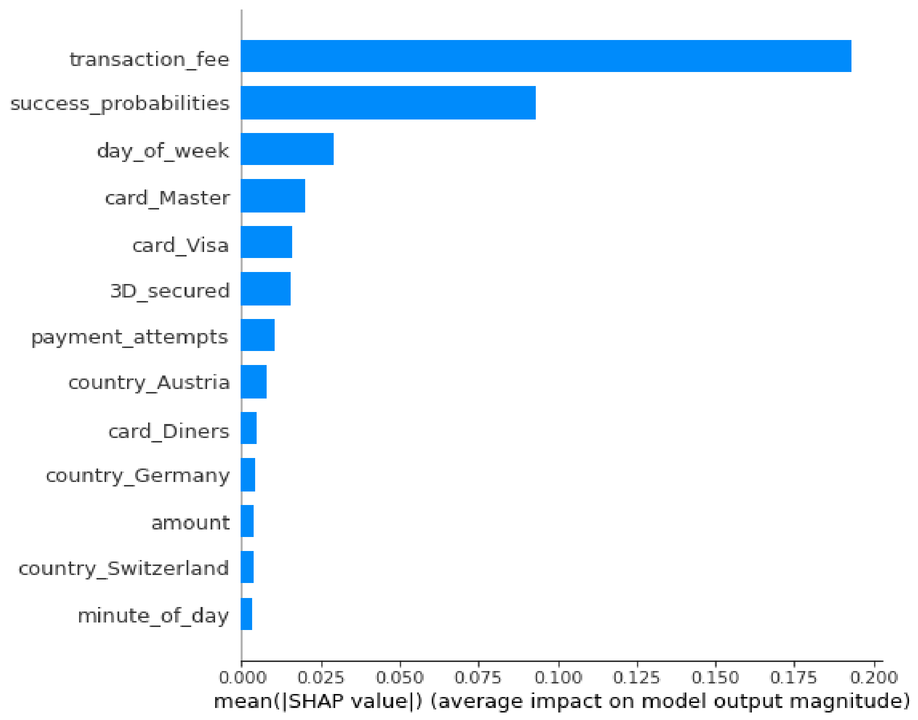


Figure 22: Summary bar plot of the importance of individual features in Model 2

According to Figure 22, the SHAP analysis for Model 2 emphasizes the crucial significance of "transaction_fee", with a significant SHAP value surpassing 0.200, emphasizing its influence on model outcomes. This emphasizes the importance of transaction fees in the financial dynamics represented by the algorithm. Furthermore, characteristics such as "success_probabilities", "day_of_week", and specific card kinds ("card_Master," "card_Visa") play an important role, demonstrating the model's consideration of multiple transactional and temporal elements.

The study additionally examines the influence of transactional and geographical aspects such as "3D_secured," "payment_attempts", and country-specific features ("country_Austria", "country_Germany", and "country_Switzerland"). This highlights the model's ability to recognize both transaction-related variables and geographical aspects when shaping predictions. To summarize, the performance of Model 2 is significantly influenced by a combination of transaction-related variables, success probability, temporal factors, and individual card kinds.

3.5.2 Interpretability of Results:

The capacity to interpret outcomes is critical to ensuring that both technical and non-technical stakeholders understand and trust the model's predictions. Logistic Regression, Random Forest Classification, and Support Vector Machines were used in the technical evaluation for both Model 1 (predicting success probabilities) and Model 2 (predicting optimal PSPs). Random Forest Classifier outperformed both models, obtaining high accuracy, precision, recall, F1 Score, and AUC-ROC on both validation and test sets. Hyperparameter adjustment increased measurements even further, ensuring robust generalization. The cross-validation and evaluation processes resulted in refined and stable models for both Model 1 and Model 2. Random Forest Classifier was finally chosen because of its stability and competitive performance in forecasting success probability and optimal PSPs, with tuning improving overall model metrics.

In terms of business interpretation, the chosen Random Forest Classifier models, particularly after hyperparameter tuning, significantly align with the main goals of automating credit card routing, increasing success rates, and lowering transaction fees. These models exhibit stability, generalizability, and increased predictive accuracy, with the Random Forest Classifier consistently outperforming on both validation and test sets. The importance of Model 2, which uses Random Forest Classifier, is in its ability to predict optimal PSPs while taking success probability and transaction fees into account, making it a significant asset for easy integration into business workflows.

3.5.3 Sophisticated Error Analysis:

Sophisticated error analysis entails looking further into model performance to identify its strengths and limitations. This technique aids in discovering patterns and gaining insights into areas where the model can be enhanced or fine-tuned. Here are some sophisticated error analysis steps we can take (Karani, 2022).

a) Confusion Matrix Analysis:

Let's look at the confusion matrices for Model 1 and Model 2's Random Forest Classifiers on the validation and test sets.

Model 1 performs better in predicting success probability, with F1 scores of 0.805 (validation) and 0.804 (test). On the validation and test sets, both accuracy and precision are about 80.7% and 80.9%, respectively. Although recall is slightly lower on the test set at 79.9%, Model 1 performs well in the context of transaction success prediction.

The Model 2 confusion matrices for both the validation and test sets show high True Positives, indicating effective identification of instances in each class. The model has low False Positives and False Negatives across most classes, indicating high performance in classifying occurrences. Overall, the results show that the models are effective at appropriately classifying cases in both datasets.

b) Precision-Recall Trade-off:

Precision-Recall is a valuable measure of prediction success when the classes are very unbalanced. Precision is a measure of result relevancy in information retrieval, whereas recall is a measure of how many truly relevant results are returned (*Precision-Recall*, n.d.).

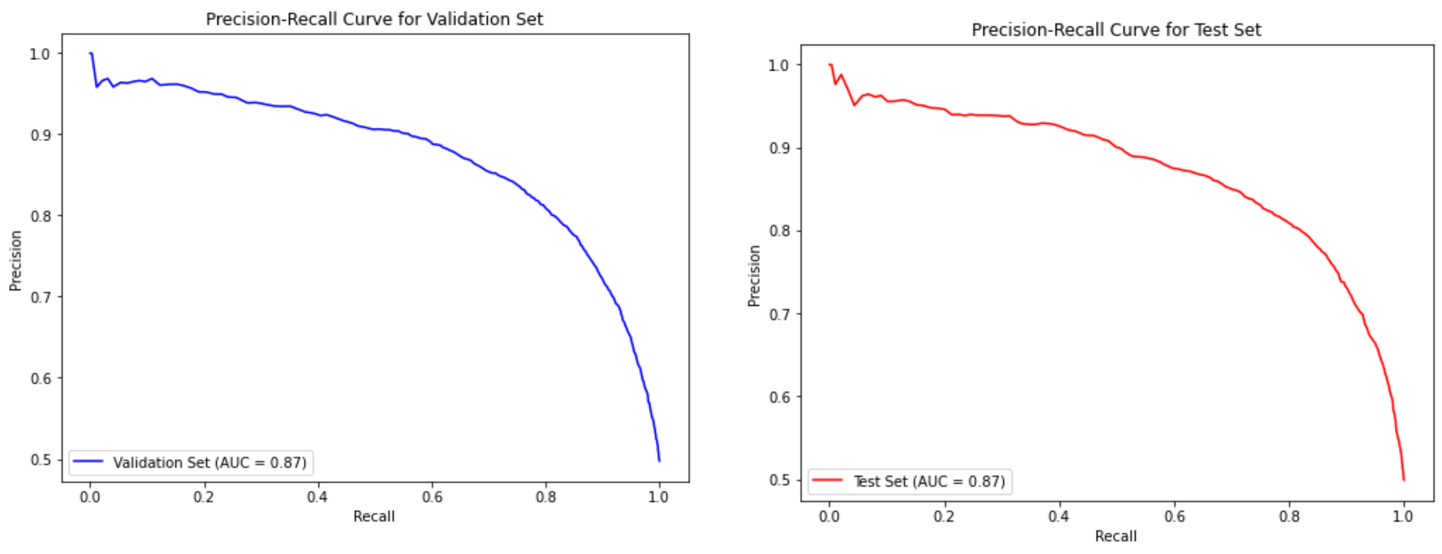
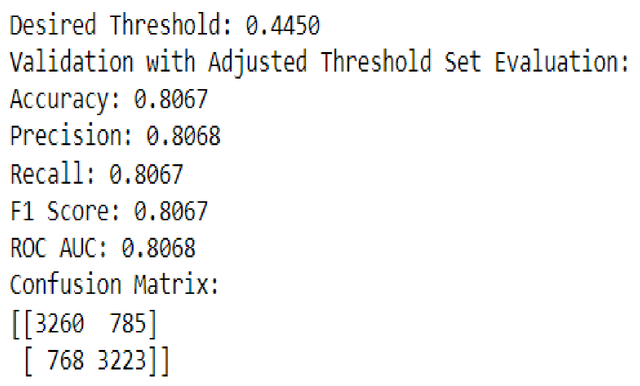
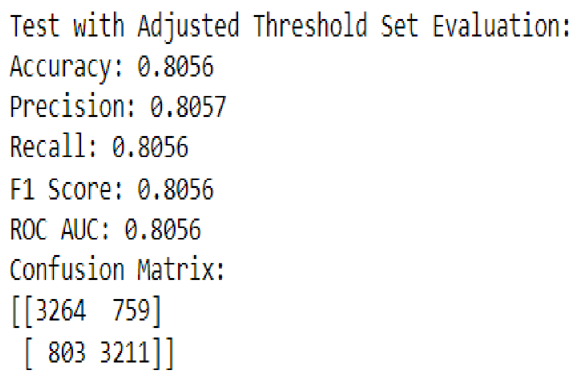


Figure 23: Precision-Recall Curve for validation and test set of Model 1

The Precision-Recall curve for Model 1's validation and test sets gave a significant AUC-PR of 0.87 for both validation and test sets, demonstrating a favorable balance between Precision and Recall (see Figure 23). The threshold was changed to reach the necessary Precision and Recall values by identifying a suitable trade-off point. As a result, Random Forest Model 1 is evaluated on validation and test sets, first using the default threshold and then dynamically adjusting the decision threshold to achieve the required trade-off between Precision and Recall. The Precision-Recall curve is used to determine the optimal threshold for balancing the provided accuracy and recall values. Model 1 is then evaluated using the altered predictions based on this threshold on both the validation and test sets.



```
Desired Threshold: 0.4450
Validation with Adjusted Threshold Set Evaluation:
Accuracy: 0.8067
Precision: 0.8068
Recall: 0.8067
F1 Score: 0.8067
ROC AUC: 0.8068
Confusion Matrix:
[[3260  785]
 [ 768 3223]]
```



```
Test with Adjusted Threshold Set Evaluation:
Accuracy: 0.8056
Precision: 0.8057
Recall: 0.8056
F1 Score: 0.8056
ROC AUC: 0.8056
Confusion Matrix:
[[3264  759]
 [ 803 3211]]
```

Output 9: Impact of threshold adjustment on Precision and Recall metrics of Model 1

The Precision-Recall trade-off study in Output 9 demonstrated that changing the threshold to 0.4450 had little effect on the model's performance parameters, as Accuracy, Precision, Recall, F1-score, and ROC AUC remained constant. The original Model 1, trained with optimized hyperparameters, performed well on both validation and test sets, demonstrating a balanced trade-off between Precision and Recall. The fact that threshold adjustment has such a small effect shows that the model was previously well calibrated.

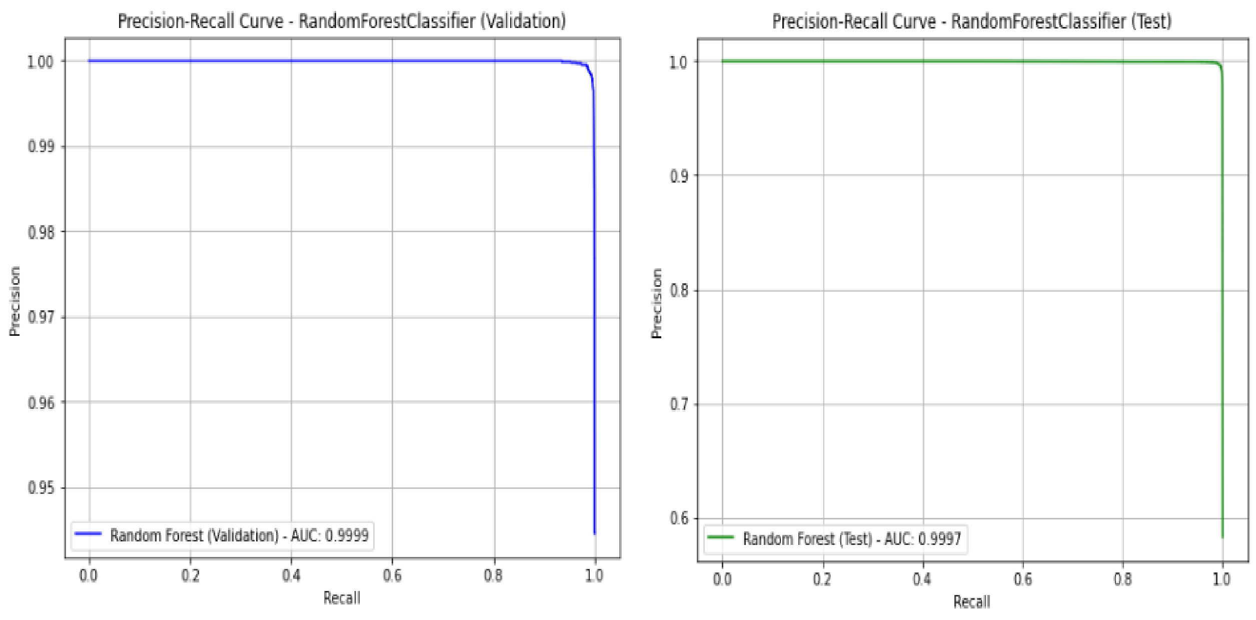


Figure 24: Precision-Recall curve for validation and test set of Model 2

Furthermore, the above Figure 24 shows that our Model 2 is likewise performing very well, with high Precision, Recall, and F1 scores on both the validation and test sets. Furthermore, the ROC AUC values are near one, indicating outstanding discriminative performance.

c) ROC Curve Analysis:

A ROC Curve is a depiction of the True Positive rate (Sensitivity) vs. the False Positive rate (1-Specificity) for various parameter cut-off points. Each point on the ROC curve corresponds to a sensitivity/specificity pair that corresponds to a specific decision threshold. The region AUC is a measure of how well a metric distinguishes between two diagnostic groups (diseased/normal) (Schoonjans, n.d.).

As a result, for Model 1, performing ROC Curve analysis by adjusting the threshold, but AUC=0.88 values remain the same for both validation and test sets after adjusting the threshold, as shown in Figure 25 below, suggests that the model's ability to discriminate between positive and negative instances (as measured by the AUC) is unaffected by the threshold adjustment. The AUC measures the overall discriminatory power of the model and is less sensitive to changes in the decision threshold.

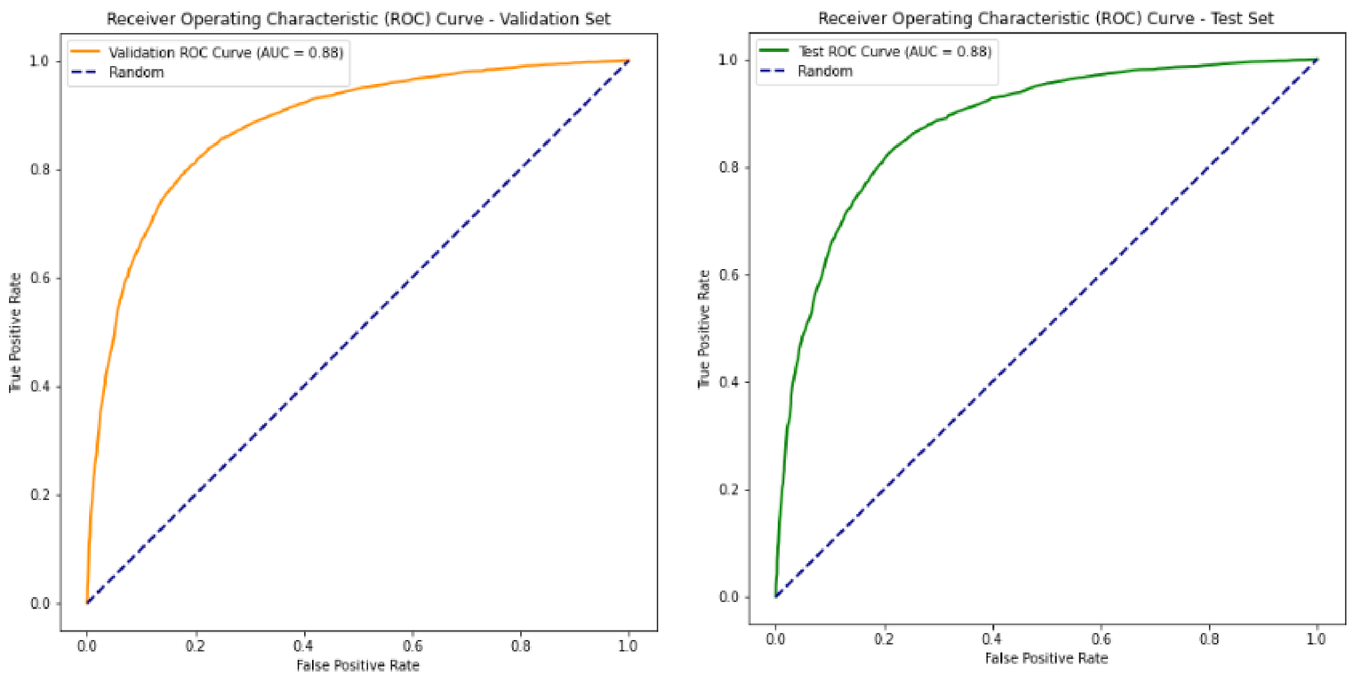


Figure 25: ROC Curve on validation and test set of Model 1

Furthermore, as seen in Figure 26 below, Model 2 has excellent predictive ability, as evidenced by ROC AUC values of 1.00 on the validation set and 1.00 on the test set.

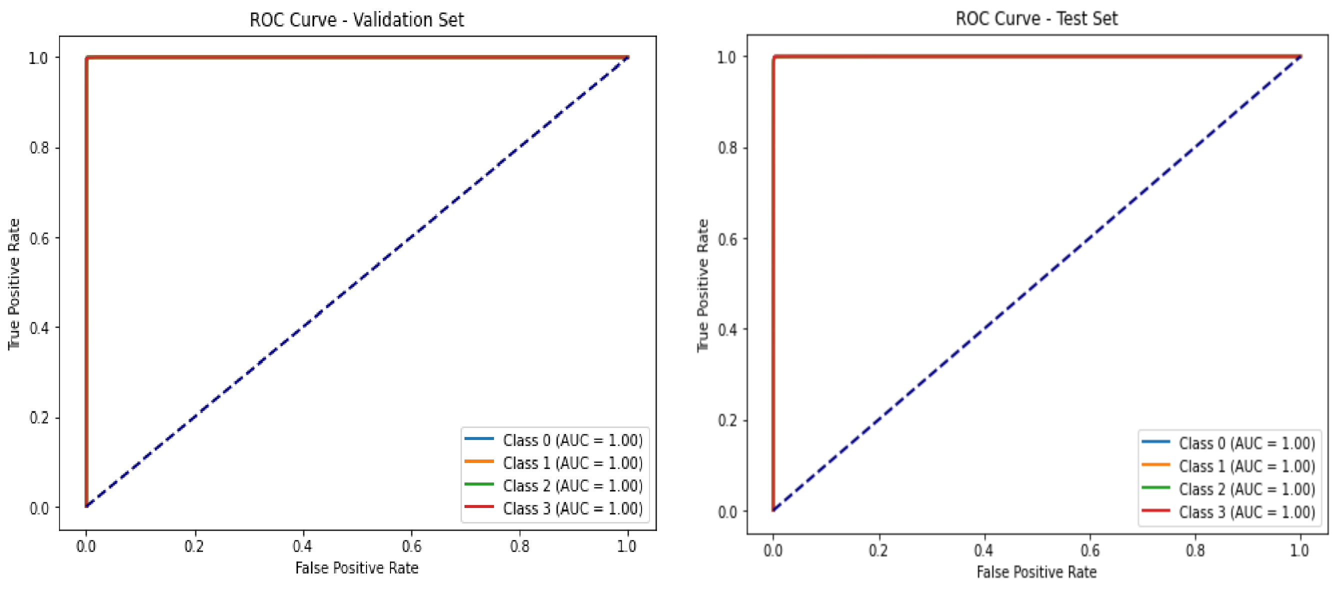
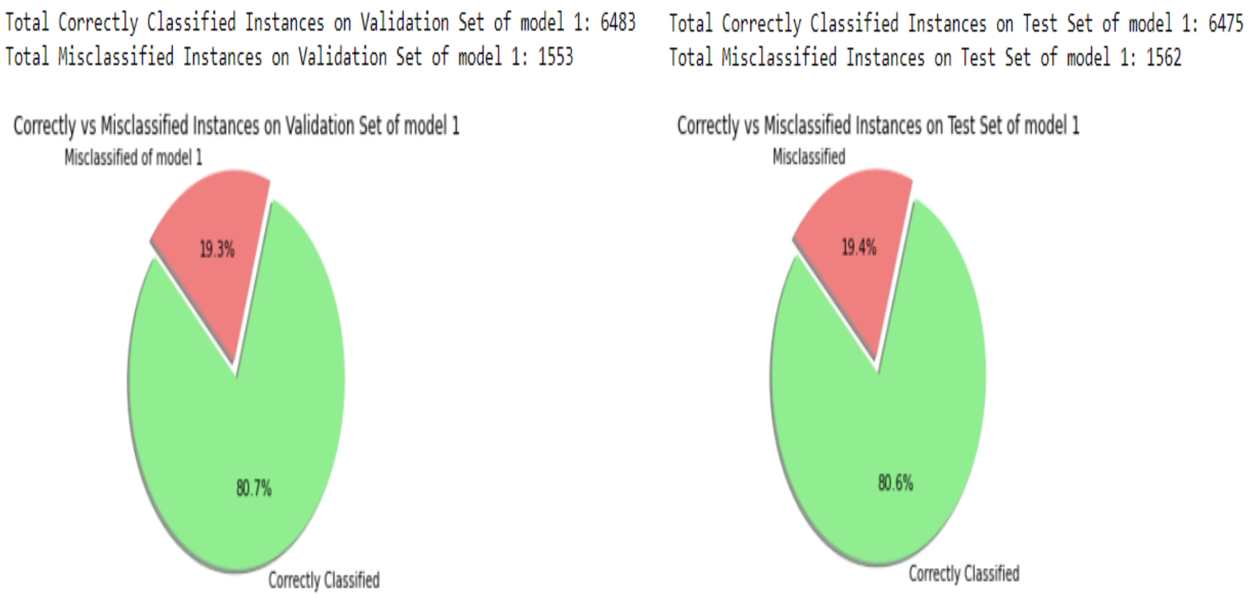


Figure 26: ROC Curve on validation and test set of Model 2

d) Misclassified Samples Inspection:

Examining misclassified instances requires searching for patterns or common characteristics among the misclassified samples. This research provides insight into areas in which the model may fail. As a result, for Model 1, the total number of misclassified instances in the validation set is 1553 and 1562 in the test set, while the total number of correctly classified instances in the validation set is 6483 and 6475 in the test set, indicating that Model 1 exhibits consistent performance in both the validation (80.7% correct classification, 19.3% misclassification) and test sets (80.6% correct classification, 19.4% misclassification). This consistency shows robust generalization and reduces concerns about overfitting to the validation and test sets.

However, there is still room to improve Model 1's performance in the future by investigating specific misclassifications. Investigation into misclassified samples can help inform targeted improvements.



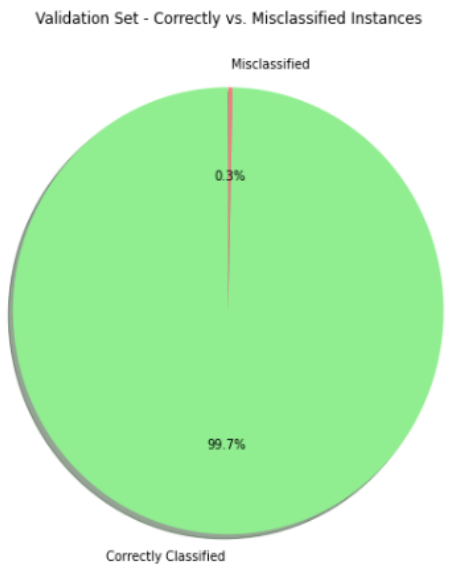
```
Total Correctly Classified Instances on Validation Set of model 1: 6483
Total Misclassified Instances on Validation Set of model 1: 1553
```

```
Total Correctly Classified Instances on Test Set of model 1: 6475
Total Misclassified Instances on Test Set of model 1: 1562
```

Figure 27: Pie chart for analysing correctly classified and misclassified data on the validation and test sets of Model 1

Similarly, for Model 2, the total correctly classified instances in the validation set are 10551 and 10540 in the test set, respectively, while the total misclassified instances in the validation set are 33 and 44 in the test set, indicating that the Model 2 performed well on both the validation and test sets. It obtained great Accuracy in the validation set, with 10551 properly classified cases out of 10584 and only 33 misclassifications. Similarly, on the test set, the model maintained a high degree of Accuracy, accurately classifying 10540 out of 10584 cases with only 44 misclassifications. This consistency shows that the model is capable of making accurate predictions on new, previously unseen data. Therefore, the correctly and misclassified instances on the validation and test sets for both Model 1 and Model 2 are visualized using a pie chart, as shown in Figures 27 and 28.

```
Total Correctly Classified Instances on validation set of model 2: 10551
Total Misclassified Instances on validation set of model 2: 33
```



```
Total Correctly Classified Instances on test set of model 2: 10540
Total Misclassified Instances on test set of model 2: 44
```

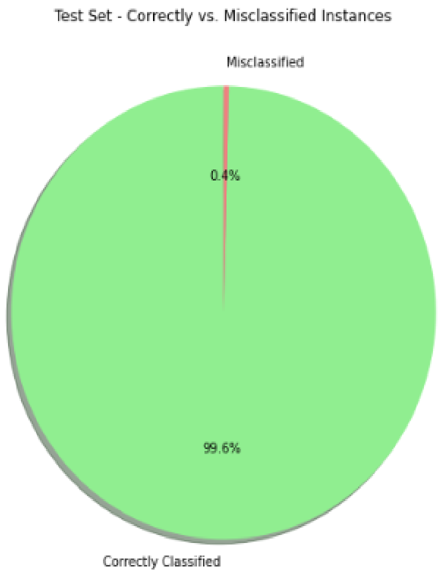


Figure 28: Pie chart for analysing correctly classified and misclassified data on the validation and test set of Model 2

e) Cross-Validation Stability:

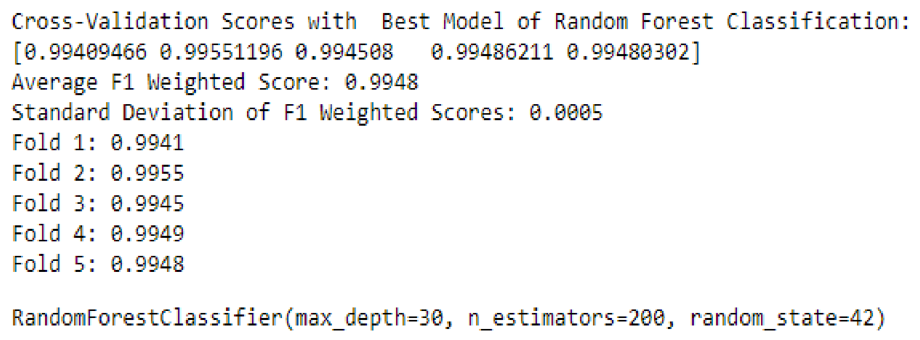
Cross-validation (CV) is a technique for evaluating and testing a machine learning model's performance. CVs are frequently utilized in applied machine learning applications. It facilitates the comparison and selection of an appropriate model for the particular predictive modelling challenge (Lyashenko, 2022). As a result, cross-validation was employed to verify Models 1 and 2's stability across numerous folds. If there is a lot of variation, it may indicate that Models 1 and 2 are sensitive to training data.

```
Fitting 5 folds for each of 108 candidates, totalling 540 fits

Results for Fold 1:
Mean Test Score: 0.7855687586109152
Standard Deviation Test Score: 0.0023204113828104735
Params: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 50}

Results for Fold 2:
Mean Test Score: 0.7883840502691462
Standard Deviation Test Score: 0.003112363723738587
Params: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100}

Results for Fold 3:
Mean Test Score: 0.7893484255830958
Standard Deviation Test Score: 0.0030605605241868054
Params: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200}

Results for Fold 4:
Mean Test Score: 0.7825356601460949
Standard Deviation Test Score: 0.0029116576737104887
Params: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 5, 'n_estimators': 50}

Results for Fold 5:
Mean Test Score: 0.7851176340255112
Standard Deviation Test Score: 0.002555382292353474
Params: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 5, 'n_estimators': 100}
```

Output 10: Cross-Validation stability results of Model 1

Initially, Model 1 is tested by identifying optimal hyperparameters, as shown in Output 10, which demonstrates consistent performance on validation and test sets. Furthermore, accessing Model 1 using GridSearchCV revealed modest mean test score changes and small standard deviations, strengthening Model 1 stability and indicating robust generalization, as indicated by negligible sensitivity to training data subsets. Overall, Model 1 exhibits cross-validation stability, indicating a high potential for generalization to previously unexplored data.

Furthermore, investigating the Model 2's stability by examining individual cross-validation scores and computing standard deviations such that the Random Forest Model 2, configured with hyperparameters max_depth=30 and n_estimators=200, exhibits stable and high-performance characteristics across various folds during cross-validation, as shown in Output 11. The average F1 weighted score is 0.9948, with a standard deviation of 0.0005, showing steady performance. Individual fold scores consistently vary between 0.9941 and 0.9955. These data indicate that the model is resilient and trustworthy, making it a desirable choice.



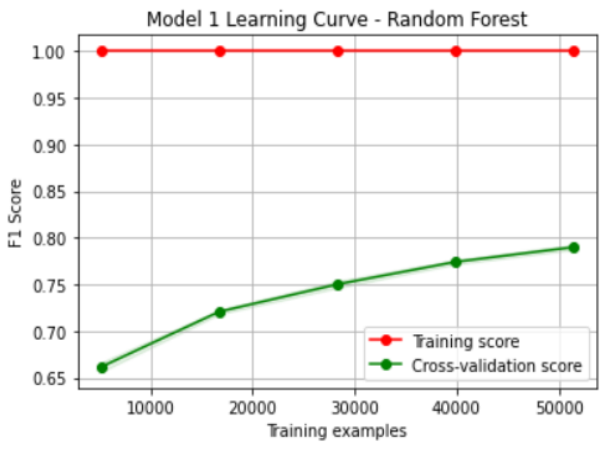
```
Cross-Validation Scores with  Best Model of Random Forest Classification:
[0.99409466 0.99551196 0.994508   0.99486211 0.99480302]
Average F1 Weighted Score: 0.9948
Standard Deviation of F1 Weighted Scores: 0.0005
Fold 1: 0.9941
Fold 2: 0.9955
Fold 3: 0.9945
Fold 4: 0.9949
Fold 5: 0.9948

RandomForestClassifier(max_depth=30, n_estimators=200, random_state=42)
```

Output 11: Cross-Validation stability results of Model 2

f) Overfitting Analysis:

Overfitting is a basic problem in supervised machine learning since it stops us from completely generalizing the models to fit observed data on training data as well as unseen data on testing data. Investigate the possibility of overfitting using approaches such as learning curves to assess overfitting (Ying, 2019).

The learning curve in our scenario displays the F1 score for both the training and cross-validation sets. As illustrated in Figure 29, the training score for Model 1 indicates that the model can precisely predict the labels of the training data, and the cross-validation score curve illustrates how effectively the model generalizes to unknown data. The rising curve indicates that the model is learning from the data and becoming better at generalizing to previously unseen examples.



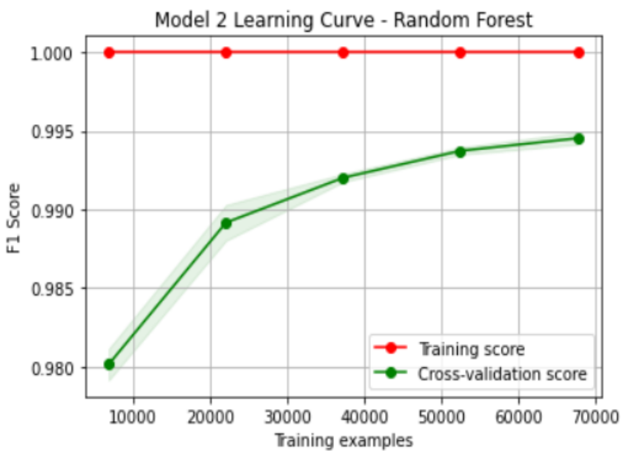


Figure 29: Model 1 learning curve

Figure 30: Model 2 learning curve

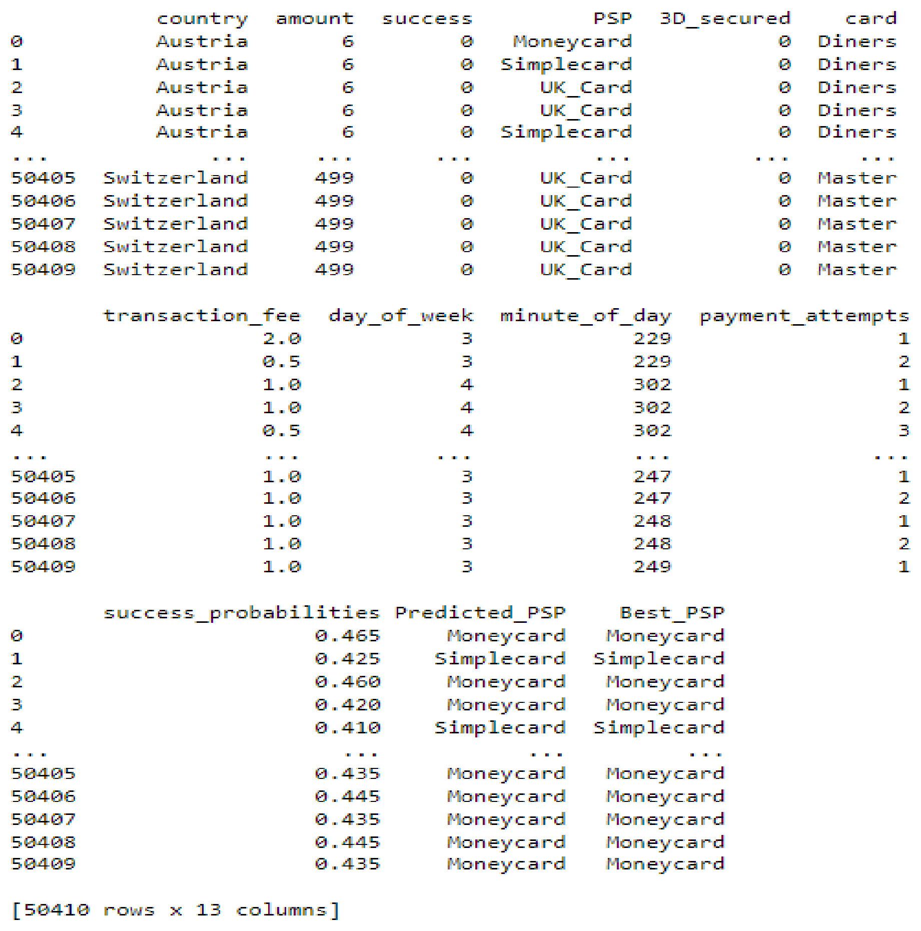
Similarly, the slight difference between the training and cross-validation scores in the above Figure 30 Model 2 learning curve often indicates that our Model 2 is also generalizing effectively to new, unseen data. It means Model 2 isn't overfitting the training data and can perform well in cases it hasn't seen before.

3.5.4 Link to Business Understanding:

It is critical at this stage to ensure that the evaluation criteria are consistent with the initial business understanding and objectives. As a result, a retail company's credit card routing project takes a two-model approach, with the goal of automating credit card routing and improving success rates while minimizing transaction expenses. The detailed project plan covers needs, assumptions, and risk management techniques, with a focus on communication and collaboration with the online payment and IT teams. The criteria for evaluation are tightly aligned with business goals, ensuring that the models match objectives and solve potential concerns. The SHAP analysis provides more insight into the significance of specific features in Models 1 and 2. The interpretability of results is a significant priority, with consideration given to both technical and non-technical stakeholders. For Model 1 and Model 2, error analysis is performed, which includes a confusion matrix, precision-recall trade-off, ROC curve, and misclassified sample inspection. The models' robust generalization is confirmed by cross-validation stability analysis. The techniques place an emphasis on linkage with larger business goals, emphasizing the project's strategic value to the retail organization.

3.6 Deployment

The CRISP-DM methodology's deployment phase refers to the process of putting the solutions created in the modelling phase into action. It is the final step in the data mining process and involves placing the solutions into a production environment where they can be used to produce value for businesses. Furthermore, since we developed Model 2 for predicting optimal PSP, the Output 13 of the predicted PSP with the column name "Predicted_PSP" for each existing transaction with the lowest transaction cost and high success probabilities is shown below. In addition, the code is designed to handle ties in predicted PSPs by selecting which is the most cost-effective PSP based on transaction costs, as indicated by the column Best_PSP. However, no examples of ties were found in our situation.



```
           country  amount  success         PSP  3D_secured    card
0          Austria       6        0   Moneycard           0  Diners
1          Austria       6        0  Simplecard           0  Diners
2          Austria       6        0     UK_Card           0  Diners
3          Austria       6        0     UK_Card           0  Diners
4          Austria       6        0  Simplecard           0  Diners
...            ...     ...      ...         ...         ...     ...
50405  Switzerland     499        0     UK_Card           0  Master
50406  Switzerland     499        0     UK_Card           0  Master
50407  Switzerland     499        0     UK_Card           0  Master
50408  Switzerland     499        0     UK_Card           0  Master
50409  Switzerland     499        0     UK_Card           0  Master

       transaction_fee  day_of_week  minute_of_day  payment_attempts
0                  2.0            3            229                 1
1                  0.5            3            229                 2
2                  1.0            4            302                 1
3                  1.0            4            302                 2
4                  0.5            4            302                 3
...                ...          ...            ...               ...
50405              1.0            3            247                 1
50406              1.0            3            247                 2
50407              1.0            3            248                 1
50408              1.0            3            248                 2
50409              1.0            3            249                 1

       success_probabilities Predicted_PSP    Best_PSP
0                      0.465     Moneycard   Moneycard
1                      0.425    Simplecard  Simplecard
2                      0.460     Moneycard   Moneycard
3                      0.420     Moneycard   Moneycard
4                      0.410    Simplecard  Simplecard
...                      ...           ...         ...
50405                  0.435     Moneycard   Moneycard
50406                  0.445     Moneycard   Moneycard
50407                  0.435     Moneycard   Moneycard
50408                  0.445     Moneycard   Moneycard
50409                  0.435     Moneycard   Moneycard

[50410 rows x 13 columns]
```

Output 13: Data frame with optimal PSP prediction for each exiting transaction

Furthermore, the CRISP-DM deployment step is important to the success of the data mining process. All of the work that has gone into the business understanding, data understanding, data preparation, modelling, and evaluation phases will be wasted without effective deployment. The modelling phase's solutions are only valuable if they are implemented, which is what the deployment step ensures (Yennhi95zz, 2023). Here is a proposal for deployment in the business production environment (Heymann et al., 2022).

a) Step 1: Technical Setup

To begin, build the technological infrastructure required to host the predictive model, assuring a smooth deployment. Configuring hardware and software settings, as well as ensuring compatibility with machine learning libraries, are all part of the process. Serialise models (Models 1 and 2) for simple loading, and describe dependencies to ensure consistency across development and production environments.

b) Step 2: Data Pipeline Integration

The machine learning pipeline automates the process of building prediction models, increasing reusability and improving optimization. Integrating predictive models into the data processing pipeline, establishing connections, implementing real-time data handling, and incorporating appropriate pre-processing procedures for increased efficiency. This modular approach ensures that models are versatile and efficient to create (“Data Pipeline Machine Learning Architecture,” 2023).

c) Step 3: GUI Development

Create a user-friendly dashboard using Gradio to facilitate easy transaction input and routing suggestion visualizations in machine learning. Ensure secure application access using user authentication, which improves total user involvement and comprehension of machine learning outputs. Integrate Gradio's capabilities to ensure a smooth deployment and an excellent user experience (*Build a GUI for Your Machine Learning Models | HackerNoon*, n.d.).

d) Step 4: Key Feature Integration

Integrating real-time decision-making by incorporating Model 1 and Model 2 predictions and dynamically selecting payment service providers depending on success probabilities and transaction fees. Implementing alerts or notifications to address crucial occurrences like model failures as soon as possible guarantees proactive system monitoring and maintenance.

e) Step 5: Real-time Updates and Monitoring

Model monitoring is a continuous process that involves tracking, analysing, and assessing machine learning models in real-world scenarios. It ensures long-term accuracy and dependability by tracking numerous data and model metrics over time (*Model Monitoring for ML in Production*, n.d.). Creating a monitoring dashboard to measure important performance parameters such as success rates and fees, as well as establishing an automatic method for updating models in real-time when new data becomes available, enables adaptability to changing transaction patterns through regular retraining, permitting efficient monitoring and seamless Model 1 and Model 2 modifications for optimal performance.

Chapter 4

Conclusion and Future Work

4.1 Conclusion

The completion of the credit card routing predictive modelling project represents an extraordinary achievement, demonstrating a comprehensive and systematic methodology from the start of the project to deployment. The proposed Git repository structure is a well-organized framework that allows for better project management, code organisation, and collaboration. The project maintains transparency, consistency, and accessibility for users through directory separation, a "requirements.txt" file, and a complete "README.md" documentation. The investigative method implemented in Chapter 3 adheres to the CRISP-DM model and indicates a comprehensive data mining procedure. The business knowledge phase sets a solid foundation by expressing the project's aims and objectives using a strategic two-model approach for success prediction and PSP selection. The following understanding of the data phase goes into relevant data collection, quality evaluation, and feature exploration, boosting project insights through visualisations and analytics.

The extensive data assessment includes a complete examination of transaction details, success rates, security features, card types, and temporal dynamics, all of which contribute to a more sophisticated knowledge of the credit card routing project. This understanding is further developed by the identification of needs, assumptions, limitations, risks, and contingencies, resulting in a solid basis for project planning and risk mitigation that is aligned with broader corporate goals. The data preparation step is critical, requiring meticulous tasks such as data collection, cleaning, transformation, and feature selection. The generated dataset is thoroughly cleaned and transformed once it has been augmented with information on payment service providers and their fees. Feature selection focuses on critical variables, and imbalanced data is addressed using advanced techniques such as SMOTE, laying the groundwork for the modelling step.

The modelling phase offers two well-crafted models, Model 1 and Model 2, that demonstrate the Random Forest Classifier's supremacy. Model 1 forecasts success probabilities, but Model 2 broadens the scope to forecast ideal payment service providers. The models chosen, particularly the Random Forest Classifier, are well aligned with the project's goals of automating credit card routing, improving success rates, and lowering transaction fees. The models are suitable for incorporation into business operations due to their solid performance, stability, and generalizability. The evaluation step, which is rigorously carried out within the CRISP-DM framework, reveals the importance of temporal, transactional, and spatial aspects in influencing outcomes. The superiority of the Random Forest Classifier is emphasised, emphasising stability, generalizability, and higher predictive accuracy in connection with corporate goals.

Sophisticated error analysis approaches validate both models' robust performance in classifying occurrences, suggesting their efficacy in predicting success probability and optimal PSPs. The deployment plan describes a thorough strategy for assuring successful installation into the business environment for improved credit card routing efficiency, success rates, and transaction costs.

Overall, the project's success can be attributed to its thorough approach, competent project management, and the smooth integration of models into the operations of the retail organisation. The models produced not only meet business objectives but also display interpretability and robust performance, greatly contributing to improved credit card routing operations and aligning with the ultimate aims of financial improvement and customer satisfaction.

4.2 Future Work

Future work could include investigating advanced techniques for dealing with data imbalances in Model 1, researching alternative algorithms for Model 2 to improve class discrimination, conducting in-depth analyses of misclassified instances, and implementing continuous model improvement strategies based on real-time monitoring and feedback. Furthermore, investigating the impact of external factors on model performance and increasing the system's scalability and responsiveness to changing business requirements would add to the credit card routing system's continual refinement.

Chapter 5

Bibliography

4. Basic Git Concepts—Version Control with Git, 2nd Edition [Book]. (n.d.). Retrieved January 8, 2024, from https://www.oreilly.com/library/view/version-control-with/9781449345037/ch04.html

Aziz, A., Saman, M. Y. M., & Jusoh, J. (2012). Data investigation: Issues of data quality and implementing base analysis technique to evaluate quality of data in heterogeneous databases. *Journal of Theoretical and Applied Information Technology*, *45*, 360–372.

Bhagwat, A. (n.d.). *CRISP-DM The New Blueprint for Data Mining Colin Shearer (Fall 2000)*. Retrieved December 21, 2023, from https://www.academia.edu/42079490/CRISP_DM_The_New_Blueprint_for_Data_Mining_Colin_Shearer_Fall_2000_

Build a GUI for Your Machine Learning Models | HackerNoon. (n.d.). Retrieved January 4, 2024, from https://hackernoon.com/build-a-gui-for-your-machine-learning-models

Conroy, M. (2023, May 2). CRISP-DM, Phase 4: Model Building. *Data Lab Notes*. https://datalabnotes.com/crisp-dm-model-building/

Correlation Matrix—An overview | ScienceDirect Topics. (n.d.). Retrieved December 24, 2023, from https://www-sciencedirect-com.pxz.iubh.de:8443/topics/mathematics/correlation-matrix

CRISP DM Step 1—Understanding About the Business. (n.d.). *PGBS*. Retrieved December 21, 2023, from https://www.proglobalbusinesssolutions.com/business-understanding/

CRISP-DM 1.0. Step-by-step data mining guide—PDF Free Download. (n.d.). Retrieved December 25, 2023, from https://docplayer.net/202628-Crisp-dm-1-0-step-by-step-data-mining-guide.html

Data Pipeline Machine Learning Architecture. (2023, January 11). *StreamSets*. https://streamsets.com/blog/basics-of-data-pipeline-architecture-for-machine-learning/

Heymann, H., Kies, A. D., Frye, M., Schmitt, R. H., & Boza, A. (2022). Guideline for Deployment of Machine Learning Models for Predictive Quality in Production. *Procedia CIRP*, *107*, 815–820. https://doi.org/10.1016/j.procir.2022.05.068

Jović, A., Brkić, K., & Bogunović, N. (2015). A review of feature selection methods with applications. *2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 1200–1205. https://doi.org/10.1109/MIPRO.2015.7160458

Karani, D. (2022, July 21). *Deep Dive Into Error Analysis and Model Debugging in Machine Learning (and Deep Learning)*. Neptune.Ai. https://neptune.ai/blog/deep-dive-into-error-analysis-and-model-debugging-in-machine-learning-and-deep-learning

Liu, X.-Y., Wu, J., & Zhou, Z.-H. (2009). Exploratory Undersampling for Class-Imbalance Learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, *39*(2), 539–550. https://doi.org/10.1109/TSMCB.2008.2007853

Lundberg, S. M., & Lee, S.-I. (2017). A Unified Approach to Interpreting Model Predictions. *Advances in Neural Information Processing Systems*, *30*. https://papers.nips.cc/paper_files/paper/2017/hash/8a20a8621978632d76c43dfd28b67767-Abstract.html

Lyashenko, V. (2022, July 21). *Cross-Validation in Machine Learning: How to Do It Right*. Neptune.Ai. https://neptune.ai/blog/cross-validation-in-machine-learning-how-to-do-it-right

ML | Overview of Data Cleaning. (2018, May 15). *GeeksforGeeks*. https://www.geeksforgeeks.org/data-cleansing-introduction/

Model monitoring for ML in production: A comprehensive guide. (n.d.). Retrieved January 4, 2024, from https://www.evidentlyai.com/ml-in-production/model-monitoring

MyEducator - CRISP-DM: Data Mining Process. (n.d.). Retrieved December 22, 2023, from https://app.myeducator.com/reader/web/1421a/2/qk5s5/

Nadali, A., Kakhky, E. N., & Nosratabadi, H. E. (2011). Evaluating the success level of data mining projects based on CRISP-DM methodology by a Fuzzy expert system. *2011 3rd International Conference on Electronics Computer Technology*, *6*, 161–165. https://doi.org/10.1109/ICECTECH.2011.5942073

Otten, N. V. (2023, April 7). *Data Quality In Machine Learning—Explained, Issues, How To Fix Them & Python Tools*. Spot Intelligence. https://spotintelligence.com/2023/04/07/data-quality-machine-learning/

Phase 4 of the CRISP-DM Process Model: Modeling. (n.d.). Dummies. Retrieved December 27, 2023, from https://www.dummies.com/article/technology/information-technology/data-science/general-data-science/phase-4-of-the-crisp-dm-process-model-modeling-148176/

Precision-Recall. (n.d.). Scikit-Learn. Retrieved January 2, 2024, from https://scikit-learn/stable/auto_examples/model_selection/plot_precision_recall.html

Reitermanova, Z. (2010). Data splitting. *WDS*, *10*, 31–36. https://www.mff.cuni.cz/veda/konference/wds/proc/pdf10/WDS10_105_i1_Reitermanova.pdf

Schoonjans, F. (n.d.). *ROC curve analysis*. MedCalc. Retrieved January 3, 2024, from https://www.medcalc.org/manual/roc-curves.php

Singh, S. (2023, February 15). Cross Validation and Hyperparameter Tuning: A Beginner's Guide. *Medium*. https://medium.com/@sandeepmaths04/cross-validation-and-hyperparameter-tuning-a-beginners-guide-96d258eedee7

SL, S. (2020, July 2). PAIRPLOT VISUALIZATION. *Analytics Vidhya*. https://medium.com/analytics-vidhya/pairplot-visualization-16325cd725e6

Spotfire | Unveiling Data Exploration: A Gateway to Informed Business Analysis. (n.d.). Spotfire. Retrieved December 24, 2023, from https://www.spotfire.com/glossary/what-is-data-exploration.html

Wardhani, N. W. S., Rochayani, M. Y., Iriany, A., Sulistyono, A. D., & Lestantyo, P. (2019). Cross-validation Metrics for Evaluating Classification Performance on Imbalanced Data. *2019 International Conference on Computer, Control, Informatics and Its Applications (IC3INA)*, 14–18. https://doi.org/10.1109/IC3INA48034.2019.8949568

Yennhi95zz. (2023, March 13). #6. The Deployment Phase of the CRISP-DM Methodology: A Detailed Discussion. *Medium*. https://medium.com/@yennhi95zz/6-the-deployment-phase-of-the-crisp-dm-methodology-a-detailed-discussion-f802a7cb9a0f

Ying, X. (2019). An Overview of Overfitting and its Solutions. *Journal of Physics: Conference Series*, *1168*(2), 022022. https://doi.org/10.1088/1742-6596/1168/2/022022

Appendix A: Python Program

```
# Note: Before running this code, ensure that you have installed the required libraries.

# Note: Replace the file path with the correct path to the "PSP_Jan_Feb_2019.xlsx" Excel file.

# This code reads data from the specified Excel file into a DataFrame.


# Import necessary libraries

import pandas as pd  # For data manipulation and analysis

import matplotlib.pyplot as plt  # For data visualization

import seaborn as sns  # For statistical data visualization

import numpy as np  # For numerical operations

from sklearn.model_selection import train_test_split, GridSearchCV, StratifiedKFold,
cross_val_score  # For model selection and evaluation

from sklearn.neighbors import KNeighborsClassifier  # For k-nearest neighbors classification

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score, preci-
sion_score, recall_score, f1_score, roc_auc_score, make_scorer  # For model evaluation met-
rics

from sklearn.preprocessing import StandardScaler, LabelBinarizer, OneHotEncoder, label_bina-
rize  # For data preprocessing

from imblearn.over_sampling import SMOTE  # For handling imbalanced datasets

from sklearn.linear_model import LogisticRegression  # For logistic regression modeling

from sklearn.ensemble import RandomForestClassifier  # For random forest classification mod-
eling

from sklearn.svm import SVC  # For support vector machine classification modeling

import shap  # For SHAP (SHapley Additive exPlanations) values

from sklearn.metrics import precision_recall_curve, auc, roc_curve  # For precision-recall and
ROC curve analysis

from sklearn.compose import ColumnTransformer  # For applying transformers to columns in a
dataset

from sklearn.model_selection import learning_curve  # For plotting learning curves

# Read data from Excel file into a DataFrame

df = pd.read_excel(r"C:\Users\HP\OneDrive\Desktop\Model Engineering\Dataset_ex-
cel\PSP_Jan_Feb_2019.xlsx")


# Print column names of the DataFrame

print(df.columns)
```

```
# Check the number of missing values for each column
missing_values = df.isnull().sum()

# Check the percentage of missing values for each column
percentage_missing = (missing_values / len(df)) * 100

# Create a DataFrame to display the missing values information
missing_data = pd.DataFrame({
    'Missing Values': missing_values,
    'Percentage Missing': percentage_missing
})

# Print the missing values information
print(missing_data)
```

```
# Determine the data types for each column in the DataFrame
data_types = df.dtypes

# Print the data types information
print(data_types)
```

```
# Iterate through each column in the DataFrame
for column in df.columns:
    # Get unique values in the current column
    unique_values = df[column].unique()
    # Print the column name and its unique values
    print(f"Column: {column}\nUnique Values: {unique_values}\n")
```

```
# Print the number of rows in the DataFrame
print("Number of rows in ideal_data set:", len(df))

# Print the number of columns in the DataFrame
print("Number of columns in ideal_data set:", len(df.columns))
```

```
# Get the DataFrame's first five rows
first_five_rows = df.head(5)
# Print the first five rows
print(first_five_rows)
```

```
# Get the last five rows of the DataFrame
last_five_rows = df.tail(5)
# Print the last five rows
print(last_five_rows)
```

```
# Print information about the DataFrame, including data types and memory usage
print(df.info())
```

```
# Drop the 'Unnamed: 0' column from the DataFrame
df = df.drop('Unnamed: 0', axis=1)
# Print the remaining columns after dropping 'Unnamed: 0'
print(df.columns)
```

```
# Print descriptive statistics for the 'amount' column
print(df['amount'].describe().to_frame().T)

# Visualization of transaction amount distribution
plt.figure(figsize=(10, 6))

# Plot a histogram with 30 bins and a kernel density estimate
sns.histplot(df['amount'], bins=30, kde=True)

# Set plot title and axis labels
plt.title('Distribution of Transaction Amount')
plt.xlabel('Transaction Amount')
plt.ylabel('Frequency')

# Show the plot of transaction amount distribution
plt.show()
```

```
# Print summary statistics for the 'success' column
print("Summary statistics for 'success' column:")
print(df['success'].value_counts())
# Visualization of the distribution of 'success' column
plt.figure(figsize=(6, 4))
# Create a countplot for the 'success' column
sns.countplot(x='success', data=df)
# Set plot title and axis labels
plt.title('Distribution of Success')
plt.xlabel('Success')
plt.ylabel('Count')
# Show the plot distribution of success
plt.show()
```

```
# Print summary statistics for the '3D_secured' column
print("\nSummary statistics for '3D_secured' column:")
print(df['3D_secured'].value_counts())

# Visualization of the distribution of '3D_secured' column
plt.figure(figsize=(6, 4))

# Create a countplot for the '3D_secured' column
sns.countplot(x='3D_secured', data=df)

# Set plot title and axis labels
plt.title('Distribution of 3D Secure Transactions')
plt.xlabel('3D Secured')
plt.ylabel('Count')

# Show the plot distribution of 3D_secured
plt.show()
```

```
# Print summary statistics for the 'country' column
print("\nSummary statistics for 'country' column:")
print(df['country'].value_counts())

# Visualization of the count of transactions by country
plt.figure(figsize=(12, 6))

# Create a countplot for the 'country' column
sns.countplot(x='country', data=df)

# Set plot title and axis labels
plt.title('Count of Transactions by Country')
plt.xlabel('Country')
plt.ylabel('Count')
# Show the plot count of transactions by country
plt.show()
```

```
# Print summary statistics for the 'PSP' column
print("\nSummary statistics for 'PSP' column:")
print(df['PSP'].value_counts())

# Visualization of the distribution of transactions by PSP
plt.figure(figsize=(12, 6))
# Create a countplot for the 'PSP' column
sns.countplot(x='PSP', data=df)

# Set plot title, axis labels, and rotate x-axis labels for better readability
plt.title('Distribution of Transactions by PSP')
plt.xlabel('PSP')
plt.ylabel('Count')
plt.xticks(rotation=45)

# Show the plot distribution of transactions by PSP
plt.show()
```

```
# Print summary statistics for the 'card' column
print("\nSummary statistics for 'card' column:")
print(df['card'].value_counts())

# Visualization of the distribution of cards
plt.figure(figsize=(10, 6))
# Create a countplot for the 'card' column
sns.countplot(x='card', data=df)
# Set plot title and axis labels
plt.title('Distribution of Card')
plt.xlabel('Card')
plt.ylabel('Count')
# Show the plot distribution of cards
plt.show()
```

```
# Visualization of the relationship between success and transaction amount using a boxplot
plt.figure(figsize=(8, 6))

# Create a boxplot with 'success' on the x-axis and 'amount' on the y-axis
sns.boxplot(x='success', y='amount', data=df)
# Set plot title and axis labels
plt.title('Relationship between Success and Transaction Amount')
plt.xlabel('Success')
plt.ylabel('Transaction Amount')

# Show the plot relationship between success and transaction amount
plt.show()
```

```
# Create a pair plot for selected variables: 'amount', 'success', and '3D_secured'
sns.pairplot(df[['amount', 'success', '3D_secured']])
# Set the overall title for the pair plot
plt.suptitle('Pair Plot of Variables')
# Show the plot
plt.show()
```

```
# Create a pair plot for selected variables: 'amount', 'success', and '3D_secured', with hue
based on 'card' column

sns.pairplot(df, hue='card', vars=['amount', 'success', '3D_secured'])

# Set the overall title for the pair plot

plt.suptitle('Pair Plot of Variables by Card Type')

# Show the plot

plt.show()
```

```
# Select relevant columns for correlation analysis

selected_columns = ['amount', 'success', '3D_secured']

# Calculate the correlation matrix

correlation_matrix = df[selected_columns].corr()

# Create a heatmap to visualize the correlation matrix

plt.figure(figsize=(8, 6))

sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=.5)

# Set plot title

plt.title('Correlation Matrix: Amount, Success, and 3D_secured')

# Show the plot

plt.show()
```

```
# Create a bar plot to visualize success rate by card type

plt.figure(figsize=(8, 6))

# Use a bar plot with 'card' on the x-axis, 'success' on the y-axis, and no confidence interval
(ci=None)

sns.barplot(x='card', y='success', data=df, ci=None)


# Set plot title and axis labels

plt.title('Success Rate by Card Type')

plt.xlabel('Card Type')

plt.ylabel('Success Rate')


# Show the plot

plt.show()
```

```
# Convert 'tmsp' column to datetime format
df['tmsp'] = pd.to_datetime(df['tmsp'])

# Set 'tmsp' as the index of the DataFrame
df.set_index('tmsp', inplace=True)

# Resample the data by day and plot the daily transaction volume
df.resample('D').size().plot(title='Daily Transaction Volume', figsize=(12, 6))

# Set x-axis and y-axis labels
plt.xlabel('Date')
plt.ylabel('Transaction Volume')

# Show the plot
plt.show()
```

```
# Create a boxplot to visualize transaction amounts by country
plt.figure(figsize=(14, 8))

# Use a boxplot with 'country' on the x-axis and 'amount' on the y-axis
sns.boxplot(x='country', y='amount', data=df)

# Set plot title, axis labels, and rotate x-axis labels for better readability
plt.title('Boxplot of Transaction Amounts by Country')
plt.xlabel('Country')
plt.ylabel('Transaction Amount')
plt.xticks(rotation=45)

# Show the plot
plt.show()
```

```
# Resample the DataFrame by day and calculate the sum for each day
df_resampled = df.resample('D').sum()

# Create a line plot for success and failure over time
plt.figure(figsize=(12, 6))
# Use lineplot to plot 'success' and 'failure' against the resampled dates
sns.lineplot(x=df_resampled.index, y='success', data=df_resampled, label='Success')
sns.lineplot(x=df_resampled.index, y=1 - df_resampled['success'], label='Failure')

# Set plot title, x-axis label, y-axis label, and add legend
plt.title('Time Series of Success and Failure')
plt.xlabel('Date')
plt.ylabel('Count')
plt.legend()
# Show the plot
plt.show()
```

```
# Create a figure
plt.figure(figsize=(12, 6))

# Calculate the daily success rate and plot it over time
success_rate_by_date = df.resample('D')['success'].mean()

# Use a line plot with markers to visualize the success rate over time
success_rate_by_date.plot(marker='o', linestyle='-')

# Set plot title, x-axis label, y-axis label
plt.title('Success Rate Over Time')
plt.xlabel('Date')
plt.ylabel('Success Rate')

# Show the plot
plt.show()
```

```
# Reset the index of the DataFrame
df.reset_index(inplace=True)
# Define dictionaries for mapping success and failure transaction fees based on card type
success_fee_mapping = {
    'Moneycard': 5,
    'Goldcard': 10,
    'UK_Card': 3,
    'Simplecard': 1
}
failed_fee_mapping = {
    'Moneycard': 2,
    'Goldcard': 5,
    'UK_Card': 1,
    'Simplecard': 0.5
}
# Create a new column 'transaction_fee' based on success or failure, using np.where
df['transaction_fee'] = np.where(df['success'] == 1, df['PSP'].map(success_fee_mapping),
df['PSP'].map(failed_fee_mapping))


# Print the first few rows of the updated DataFrame
print(df.head())
```

```
# Convert 'tmsp' column to datetime format
df['tmsp'] = pd.to_datetime(df['tmsp'])


# Add 'day_of_week' column with numerical mapping (Monday 0, ..., Sunday 6)
df['day_of_week'] = df['tmsp'].dt.dayofweek


# Add 'minute_of_day' column
df['minute_of_day'] = df['tmsp'].dt.hour * 60 + df['tmsp'].dt.minute


# Print the first few rows of the updated DataFrame
print(df.head())
```

```
# Sort the DataFrame by 'country', 'amount', 'day_of_week', 'minute_of_day'
df.sort_values(by=['country', 'amount', 'minute_of_day'], inplace=True)


# Create a new column 'payment_attempts' and initialize it with 1
df['payment_attempts'] = 1
# Identify rows where consecutive attempts have the same 'country', 'amount', 'day_of_week', 
and 'minute_of_day'
# Increment the 'payment_attempts' for those rows
df['payment_attempts'] = df.groupby(['country', 'amount', 'minute_of_day']).cumcount() + 1
# Reset the DataFrame index
df.reset_index(drop=True, inplace=True)


# Display the updated DataFrame
print(df.head())
```

```
# Drop the 'tmsp' column
df.drop('tmsp', axis=1, inplace=True)
print(df.head())
```

```
# Select numeric columns for summary statistics
numeric_columns = ['transaction_fee']
# Calculate descriptive statistics for numeric columns
numeric_summary = df[numeric_columns].describe()
# Transpose the summary for better readability
transposed_summary = numeric_summary.transpose()
# Print the transposed summary
print(transposed_summary)
# Create a histogram to visualize the distribution of transaction fees
plt.hist(df['transaction_fee'], bins=10, edgecolor='black')
# Set plot title, x-axis label, y-axis label
plt.title('Transaction Fee Histogram')
plt.xlabel('Transaction Fee')
plt.ylabel('Frequency')
# Show the plot
plt.show()
```

```
# Create a histogram to visualize the distribution of 'day_of_week'
plt.hist(df['day_of_week'], bins=7, edgecolor='black')
# Set x-axis label, y-axis label, and plot title
plt.xlabel('Day of Week')
plt.ylabel('Frequency')
plt.title('Distribution of Day of Week')
# Show the plot
plt.show()
```

```
# Create a histogram to visualize the distribution of 'minute_of_day'
plt.hist(df['minute_of_day'], bins=24, edgecolor='black')
# Set x-axis label, y-axis label, and plot title
plt.xlabel('Minute of Day')
plt.ylabel('Frequency')
plt.title('Distribution of Minute of Day')
# Show the plot
plt.show()
```

```
# Count the occurrences of each value in the 'payment_attempts' column and sort by index
attempt_counts = df['payment_attempts'].value_counts().sort_index()

# Print the count of payment attempts for each value
print(attempt_counts)

# Create a histogram to visualize the distribution of 'payment_attempts'
plt.hist(df['payment_attempts'], bins=7, edgecolor='black')
# Set x-axis label, y-axis label, and plot title
plt.xlabel('Payment Attempts')
plt.ylabel('Frequency')
plt.title('Distribution of Payment Attempts')

# Show the plot
plt.show()
```

```
# Examine unique values in 'country' column
unique_countries = df['country'].unique()
print("Unique values in 'country' column:", unique_countries)
# Examine unique values in 'PSP' column
unique_psps = df['PSP'].unique()
print("Unique values in 'PSP' column:", unique_psps)
# Examine unique values in 'card' column
unique_cards = df['card'].unique()
print("Unique values in 'card' column:", unique_cards)
```

```
# Box plot for 'amount'
plt.figure(figsize=(8, 6))
sns.boxplot(x=df['amount'])
plt.title('Box Plot for Amount')
plt.show()
# Box plot for 'transaction_fee'
plt.figure(figsize=(8, 6))
sns.boxplot(x=df['transaction_fee'])
plt.title('Box Plot for Transaction Fee')
plt.show()
```

```
# Set a threshold for identifying rare categories
threshold = 10

# Identify rare categories for 'country', 'PSP', and 'card'
rare_country = df['country'].value_counts()[df['country'].value_counts() < threshold].index
rare_PSP = df['PSP'].value_counts()[df['PSP'].value_counts() < threshold].index
rare_card = df['card'].value_counts()[df['card'].value_counts() < threshold].index

# Print the rare categories
print("Rare countries:", rare_country)
print("Rare PSPs:", rare_PSP)
print("Rare cards:", rare_card)
```

```
# Calculate correlations between variables
correlation_matrix = df.corr()
# Visualize the correlation matrix using a heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=.5)
plt.title('Correlation Matrix')
plt.show()
```

```
# Create a copy of the DataFrame
df_1 = df.copy()
# Drop the 'transaction_fee' column from the copied DataFrame
df_1 = df_1.drop('transaction_fee', axis=1)
# Print the modified DataFrame without the 'transaction_fee' column
print(df_1)
```

```
# Building Baseline Model 1
def evaluate_model(model, X, y, set_name):
    # Make predictions
    y_pred = model.predict(X)
    # Calculate performance metrics
    accuracy = accuracy_score(y, y_pred)
    precision = precision_score(y, y_pred)
    recall = recall_score(y, y_pred)
    f1 = f1_score(y, y_pred)
    roc_auc = roc_auc_score(y, y_pred)
    conf_matrix = confusion_matrix(y, y_pred)
    # Print the performance metrics with set name
    print(f'Model Performance on the {set_name}:')
    print(f'Accuracy: {accuracy:.4f}')
    print(f'Precision: {precision:.4f}')
    print(f'Recall: {recall:.4f}')
    print(f'F1-Score: {f1:.4f}')
    print(f'AUC-ROC: {roc_auc:.4f}')
    print('Confusion Matrix:')
    print(conf_matrix)
    print('\n')
```

```
# Encode categorical variables using one-hot encoding

df_1_encoded = pd.get_dummies(df_1, columns=['country', 'PSP', 'card'])

# Split the dataset into train, validation, and test sets

X_original = df_1_encoded.drop('success', axis=1)

y_original = df_1_encoded['success']

X_train_original, X_temp_original, y_train_original, y_temp_original = train_test_split(X_original,
y_original, test_size=0.2, random_state=42)

X_validation_original, X_test_original, y_validation_original, y_test_original =
train_test_split(X_temp_original, y_temp_original, test_size=0.5, random_state=42)

# Create and train a baseline KNN model

baseline_knn_model = KNeighborsClassifier()

baseline_knn_model.fit(X_train_original, y_train_original)

# Evaluate the model on the validation set

evaluate_model(baseline_knn_model, X_validation_original, y_validation_original, "validation
set")

# Evaluate the model on the test set

evaluate_model(baseline_knn_model, X_test_original, y_test_original, "test set")
```

```
# Encode categorical variables using one-hot encoding

df_1 = pd.get_dummies(df_1, columns=['country', 'PSP', 'card'])


# Print the DataFrame after one-hot encoding

print(df_1)
```

```
# Drop the target variable 'success' from the features

X = df_1.drop('success', axis=1)

# Initialize the StandardScaler

scaler = StandardScaler()

# Fit and transform the features using StandardScaler

X_scaled = scaler.fit_transform(X)

# Create a DataFrame with scaled features and include the 'success' column

df_1_scaled = pd.DataFrame(X_scaled, columns=X.columns)

df_1_scaled['success'] = df_1['success']

# Print the DataFrame with scaled features

print(df_1_scaled)
```

```
# Extract features and target variable for SMOTE
X_smote = df_1_scaled.drop('success', axis=1)
y_smote = df_1_scaled['success']
# Apply SMOTE
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_smote, y_smote)
# Create a new DataFrame with the resampled features and the target variable
df_1_resampled = pd.DataFrame(X_resampled, columns=X_smote.columns)
df_1_resampled['success'] = y_resampled
# Display the updated DataFrame with SMOTE
print(df_1_resampled)
```

```
# Development of Model 1
# Split the resampled dataset into features (X_1) and target variable (y_1)
X_1 = df_1_resampled.drop('success', axis=1)
y_1 = df_1_resampled['success']
# Split the dataset into train, validation, and test sets
X_train_1, X_temp, y_train_1, y_temp = train_test_split(X_1, y_1, test_size=0.2, ran-
dom_state=42)
X_valid_1, X_test_1, y_valid_1, y_test_1 = train_test_split(X_temp, y_temp, test_size=0.5, ran-
dom_state=42)
# Initialize a StandardScaler and scale the features for training, validation, and test sets
scaler = StandardScaler()
X_train_scaled_1 = scaler.fit_transform(X_train_1)
X_valid_scaled_1 = scaler.transform(X_valid_1)
X_test_scaled_1 = scaler.transform(X_test_1)
# Create and train Logistic Regression model
logreg_model_1 = LogisticRegression(random_state=42)
logreg_model_1.fit(X_train_scaled_1, y_train_1)
# Create and train Random Forest model
rf_model_1 = RandomForestClassifier(random_state=42)
rf_model_1.fit(X_train_scaled_1, y_train_1)
# Create and train Support Vector Machine (SVM) model
svm_model_1 = SVC(probability=True, random_state=42)
svm_model_1.fit(X_train_scaled_1, y_train_1)
```

```
# Define a function to evaluate the performance of a model
def evaluate_model(model, X_scaled, y, set_name):
    y_pred = model.predict(X_scaled)
    y_pred_proba = model.predict_proba(X_scaled)[:, 1]

    precision = precision_score(y, y_pred)
    recall = recall_score(y, y_pred)
    f1 = f1_score(y, y_pred)
    roc_auc = roc_auc_score(y, y_pred_proba)
    accuracy = accuracy_score(y, y_pred)
    conf_matrix = confusion_matrix(y, y_pred)

    print(f'Model Performance on {set_name} set - {type(model).__name__}:')
    print(f'Accuracy: {accuracy:.4f}')
    print(f'Precision: {precision:.4f}')
    print(f'Recall: {recall:.4f}')
    print(f'F1-Score: {f1:.4f}')
    print(f'AUC-ROC: {roc_auc:.4f}')
    print('Confusion Matrix:')
    print(conf_matrix)
    print('\n')

# Evaluate the models on the validation set
evaluate_model(logreg_model_1, X_valid_scaled_1, y_valid_1, set_name="Validation")
evaluate_model(rf_model_1, X_valid_scaled_1, y_valid_1, set_name="Validation")
evaluate_model(svm_model_1, X_valid_scaled_1, y_valid_1, set_name="Validation")

# Evaluate the models on the test set
evaluate_model(logreg_model_1, X_test_scaled_1, y_test_1, set_name="Test")
evaluate_model(rf_model_1, X_test_scaled_1, y_test_1, set_name="Test")
evaluate_model(svm_model_1, X_test_scaled_1, y_test_1, set_name="Test")
```

```
# Random Forest Classification with Cross-Validation and Hyperparameter Tuning

# Define the Random Forest model
rf_model = RandomForestClassifier(random_state=42)

# Define the parameter grid for hyperparameter tuning
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Create GridSearchCV
grid_search = GridSearchCV(estimator=rf_model, param_grid=param_grid, cv=5, scoring='ac-
curacy', n_jobs=-1, verbose=2)

# Fit the model on the training data
grid_search.fit(X_train_scaled_1, y_train_1)

# Print the best parameters and best score
print("Best Parameters:", grid_search.best_params_)
print("Best Score:", grid_search.best_score_)

# Get the best model from GridSearchCV
best_rf_model = grid_search.best_estimator_

# Evaluate the best model on the validation and test sets
evaluate_model(best_rf_model, X_valid_scaled_1, y_valid_1, set_name="Validation")
evaluate_model(best_rf_model, X_test_scaled_1, y_test_1, set_name="Test")
```

```
#Summary bar plot of the importance of individual features in Model 1

# Create a SHAP TreeExplainer for the best random forest Model 1

explainer = shap.TreeExplainer(best_rf_model)

# Obtain SHAP values for the validation set

shap_values = explainer.shap_values(X_valid_scaled_1)

# Define feature names for summary plot

feature_names = ["amount", "3D_secured", "day_of_week", "minute_of_day", "payment_at-
tempts",

            "country_Austria", "country_Germany", "country_Switzerland",

            "card_Diners", "card_Master", "card_Visa",

            "PSP_Moneycard", "PSP_Simplecard", "PSP_UK_Card", "PSP_Goldcard", "suc-
cess"]

# Create a summary plot using SHAP values

shap.summary_plot(shap_values[1], X_valid_scaled_1, feature_names=feature_names,
plot_type="bar", show=False)

# Show the plot
```

```
# Precision-Recall Curve for validation and test set of Model 1

# Predict probabilities on the validation set

y_valid_pred_proba = best_rf_model.predict_proba(X_valid_scaled_1)[:, 1]

# Predict probabilities on the test set

y_test_pred_proba = best_rf_model.predict_proba(X_test_scaled_1)[:, 1]

# Compute precision-recall curve values for validation set

precision_valid, recall_valid, thresholds_valid = precision_recall_curve(y_valid_1,
y_valid_pred_proba)

# Compute area under the curve (AUC) for validation set

pr_auc_valid = auc(recall_valid, precision_valid)

# Plot the precision-recall curve for validation set

plt.figure(figsize=(8, 6))

plt.plot(recall_valid, precision_valid, label=f'Validation Set (AUC = {pr_auc_valid:.2f})', color='b')

plt.xlabel('Recall')

plt.ylabel('Precision')

plt.title('Precision-Recall Curve for Validation Set')

plt.legend(loc='lower left')

plt.show()
```

```
# Compute precision-recall curve values for test set
precision_test, recall_test, thresholds_test = precision_recall_curve(y_test_1,
y_test_pred_proba)
# Compute area under the curve (AUC) for test set
pr_auc_test = auc(recall_test, precision_test)
# Plot the precision-recall curve for test set
plt.figure(figsize=(8, 6))
plt.plot(recall_test, precision_test, label=f'Test Set (AUC = {pr_auc_test:.2f})', color='r')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve for Test Set')
plt.legend(loc='lower left')
plt.show()
```

```
# Impact of threshold adjustment on Precision and Recall metrics of Model 1


def evaluate_model(model, X, y, set_name=""):
    """
    Evaluate the performance of a classification model on a given dataset.


    Parameters:
    - model: The trained classification model.
    - X: The feature matrix of the dataset.
    - y: The true labels of the dataset.
    - set_name: The name of the dataset (e.g., "Train", "Validation", "Test").


    Prints:
    - Accuracy, Precision, Recall, F1 Score, ROC AUC Score, Confusion Matrix.
    """
    # Make predictions
    y_pred = model.predict(X)


    # Convert y to 1D array if it's a DataFrame
    if hasattr(y, 'values'):
        y = y.values.ravel()
```

```
  # Calculate and print relevant evaluation metrics
   accuracy = accuracy_score(y, y_pred)
   precision = precision_score(y, y_pred, average='weighted')
   recall = recall_score(y, y_pred, average='weighted')
   f1 = f1_score(y, y_pred, average='weighted')
   roc_auc = roc_auc_score(y, y_pred)
   confusion_mat = confusion_matrix(y, y_pred)
   print(f"{set_name} Set Evaluation:")
   print(f"Accuracy: {accuracy:.4f}")
   print(f"Precision: {precision:.4f}")
   print(f"Recall: {recall:.4f}")
   print(f"F1 Score: {f1:.4f}")
   print(f"ROC AUC: {roc_auc:.4f}")
   print("Confusion Matrix:")
   print(confusion_mat)
   print("\n")
# Predict probabilities on the validation set
y_valid_pred_proba = best_rf_model.predict_proba(X_valid_scaled_1)[:, 1]
# Calculate precision and recall for different thresholds
precision, recall, thresholds = precision_recall_curve(y_valid_1, y_valid_pred_proba)
# Find the threshold for a desired trade-off (e.g., balance between precision and recall)
desired_precision = 0.85
desired_recall = 0.85
# Find the index of the point on the curve closest to the desired trade-off
closest_point_index = np.argmin(np.abs(precision - desired_precision) + np.abs(recall - de-
sired_recall))
# Get the corresponding threshold
desired_threshold = thresholds[closest_point_index]
# Print the desired threshold
print(f"Desired Threshold: {desired_threshold:.4f}")
# Adjust the decision threshold for the validation set
adjusted_predictions_valid = (y_valid_pred_proba >= desired_threshold).astype(int)
# Now, use the adjusted predictions in your evaluation or downstream tasks for the validation
set
evaluate_model(best_rf_model, X_valid_scaled_1, y_valid_1, set_name="Validation with Ad-
justed Threshold")
```

```
# Predict probabilities on the test set

y_test_pred_proba = best_rf_model.predict_proba(X_test_scaled_1)[:, 1]

# Adjust the decision threshold for the test set

adjusted_predictions_test = (y_test_pred_proba >= desired_threshold).astype(int)

# Now, use the adjusted predictions in your evaluation or downstream tasks for the test set

evaluate_model(best_rf_model, X_test_scaled_1, y_test_1, set_name="Test with Adjusted
Threshold")
```

```
# ROC Curve on validation and test set of Model 1

# Get probabilities on the validation set

y_valid_pred_proba = best_rf_model.predict_proba(X_valid_scaled_1)[:, 1]

# Get probabilities on the test set

y_test_pred_proba = best_rf_model.predict_proba(X_test_scaled_1)[:, 1]

# Adjust the decision threshold for both validation and test sets

adjusted_threshold = 0.445

adjusted_predictions_valid = (y_valid_pred_proba >= adjusted_threshold).astype(int)

adjusted_predictions_test = (y_test_pred_proba >= adjusted_threshold).astype(int)

# Compute ROC Curve for both validation and test sets

fpr_valid, tpr_valid, _ = roc_curve(y_valid_1, y_valid_pred_proba)

roc_auc_valid = auc(fpr_valid, tpr_valid)

fpr_test, tpr_test, _ = roc_curve(y_test_1, y_test_pred_proba)

roc_auc_test = auc(fpr_test, tpr_test)

# Plot ROC Curves separately

import matplotlib.pyplot as plt

# Plot ROC Curve for Validation set

plt.figure(figsize=(8, 8))

plt.plot(fpr_valid, tpr_valid, color='darkorange', lw=2, label=f'Validation ROC Curve (AUC =
{roc_auc_valid:.2f})')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--', label='Random')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic (ROC) Curve - Validation Set')

plt.legend()

plt.show()

plt.show()
```

```
# Plot ROC Curve for Test set

plt.figure(figsize=(8, 8))

plt.plot(fpr_test, tpr_test, color='green', lw=2, label=f'Test ROC Curve (AUC =
{roc_auc_test:.2f})')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--', label='Random')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic (ROC) Curve - Test Set')

plt.legend()

plt.show()
```

```
# Pie chart for analysing correctly classified and misclassified on validation Set of Model 1

# Confusion matrix of Model 1 on Validation Set

confusion_matrix = np.array([[3260, 785], [768, 3223]])

# Calculate total correctly classified and misclassified instances

correctly_classified = np.trace(confusion_matrix)

misclassified = np.sum(confusion_matrix) - correctly_classified


# Print the results

print("Total Correctly Classified Instances on Validation Set of model 1:", correctly_classified)

print("Total Misclassified Instances on Validation Set of model 1:", misclassified)


# Plot a pie chart

labels = ['Correctly Classified', 'Misclassified of model 1']

sizes = [correctly_classified, misclassified]

colors = ['lightgreen', 'lightcoral']

explode = (0.1, 0)  # explode the first slice


plt.pie(sizes, explode=explode, labels=labels, colors=colors, autopct='%1.1f%%', shadow=True,
startangle=140)

plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.


# Display the pie chart

plt.title('Correctly vs Misclassified Instances on Validation Set of model 1')

plt.show()
```

```
# Pie chart for analysing correctly classified and misclassified  on Test Set of Model 1

# Confusion matrix of Model 1 on Test Set

confusion_matrix = np.array([[3264, 759], [803, 3211]])

# Calculate total correctly classified and misclassified instances

correctly_classified = np.trace(confusion_matrix)

misclassified = np.sum(confusion_matrix) - correctly_classified

# Print the results

print("Total Correctly Classified Instances on Test Set of model 1:", correctly_classified)

print("Total Misclassified Instances on Test Set of model 1:", misclassified)

# Plot a pie chart

labels = ['Correctly Classified', 'Misclassified']

sizes = [correctly_classified, misclassified]

colors = ['lightgreen', 'lightcoral']

explode = (0.1, 0)  # explode the first slice

plt.pie(sizes, explode=explode, labels=labels, colors=colors, autopct='%1.1f%%',
shadow=True, startangle=140)

plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.

# Display the pie chart

plt.title('Correctly vs Misclassified Instances on Test Set of model 1')

plt.show()
```

```
#Evaluate Cross-Validation Stability of Model 1

# Define the parameter grid for hyperparameter tuning

param_grid = {

   'n_estimators': [50, 100, 200],

   'max_depth': [None, 10, 20, 30],

   'min_samples_split': [2, 5, 10],

   'min_samples_leaf': [1, 2, 4]

}

# Create GridSearchCV

grid_search = GridSearchCV(estimator=rf_model, param_grid=param_grid, cv=5, scoring='ac-
curacy', n_jobs=-1, verbose=2)

# Fit the model on the training data

grid_search.fit(X_train_scaled_1, y_train_1)

# Access cross-validation results

cv_results = grid_search.cv_results_
```

```
# Print performance metrics for each fold
for i in range(grid_search.n_splits_):
    print(f"\nResults for Fold {i + 1}:")
    print(f"Mean Test Score: {cv_results[f'mean_test_score'][i]}")
    print(f"Standard Deviation Test Score: {cv_results[f'std_test_score'][i]}")
    print(f"Params: {cv_results['params'][i]}")
```

```
# Overfitting Analysis for Model 1
# Function to plot learning curve
def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None,
                        n_jobs=-1, train_sizes=np.linspace(.1, 1.0, 5)):
    plt.figure()
    plt.title(title)
    if ylim is not None:
        plt.ylim(*ylim)
    plt.xlabel("Training examples")
    plt.ylabel("F1 Score")
    train_sizes, train_scores, test_scores = learning_curve(
        estimator, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes, scoring='f1_macro')
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)
    plt.grid()
    plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                     train_scores_mean + train_scores_std, alpha=0.1,
                     color="r")
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                     test_scores_mean + test_scores_std, alpha=0.1, color="g")
    plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
             label="Training score")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
             label="Cross-validation score")
    plt.legend(loc="best")
    return plt
```

```
# Plot learning curve
title = "Learning Curve (Random Forest)"
cv = 5  # Cross-validation folds
plot_learning_curve(best_rf_model, title, X_train_scaled_1, y_train_1, cv=cv)
plt.show()
```

```
# Drop 'success' and 'transaction_fee' columns from the original DataFrame
X_original = df.drop(['success', 'transaction_fee'], axis=1)

# One-hot encode categorical columns: 'country', 'PSP', 'card'
X_original_encoded = pd.get_dummies(X_original, columns=['country', 'PSP', 'card'])

# Identify additional columns in X_original_encoded not present in X_train_1
additional_columns = set(X_original_encoded.columns) - set(X_train_1.columns)
if additional_columns:
    print(f"Additional columns in X_original_encoded: {additional_columns}")

# Identify missing columns in X_original_encoded compared to X_train_1
missing_columns = set(X_train_1.columns) - set(X_original_encoded.columns)
if missing_columns:
    print(f"Missing columns in X_original_encoded: {missing_columns}")
# Keep only columns present in X_train_1 in X_original_encoded
X_original_encoded = X_original_encoded[X_train_1.columns]

# Scale the features using the previously defined 'scaler'
X_original_scaled = scaler.transform(X_original_encoded)
# Predict success probabilities using the trained random forest Model 1
success_probabilities = best_rf_model.predict_proba(X_original_scaled)[:, 1]

# Add the success probabilities as a new column to the original DataFrame
df['success_probabilities'] = success_probabilities

# Display the updated DataFrame
print(df)
```

```
# Create a copy of the DataFrame
df_2 = df.copy()
# Print the copied DataFrame
print(df_2)
```

```
# Building Baseline Model 2
# Encode categorical variables using one-hot encoding
df_2_encoded = pd.get_dummies(df_2, columns=['country', 'card'])

# Split the dataset into features (X) and target variable (y)
X = df_2_encoded.drop(['PSP', 'success'], axis=1)
y = df_2_encoded['PSP']

# Split the dataset into train, validation, and test sets
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.2, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, ran-
dom_state=42)

# Create and train a KNN model
knn_model = KNeighborsClassifier(n_neighbors=5)
knn_model.fit(X_train, y_train)

# Define a function to evaluate the performance of the model
def evaluate_model(model, X, y, set_name):
    y_pred = model.predict(X)
    y_proba = model.predict_proba(X) if hasattr(model, 'predict_proba') else None

    accuracy = accuracy_score(y, y_pred)
    precision = precision_score(y, y_pred, average='micro')
    recall = recall_score(y, y_pred, average='micro')
    f1 = f1_score(y, y_pred, average='micro')
    roc_auc = roc_auc_score(y, y_proba, multi_class='ovr') if y_proba is not None else None
    conf_matrix = confusion_matrix(y, y_pred)
    print(f'Model Performance on {set_name} set - {type(model).__name__}:')
    print(f'Accuracy: {accuracy:.4f}')
    print(f'Precision: {precision:.4f}')
    print(f'Recall: {recall:.4f}')
    print(f'F1-Score: {f1:.4f}')
    if roc_auc is not None:
        print(f'AUC-ROC: {roc_auc:.4f}')
    print('Confusion Matrix:')
    print(conf_matrix)
    print('\n')
# Evaluate the KNN model on the validation set
evaluate_model(knn_model, X_val, y_val, set_name='Validation')
# Evaluate the KNN model on the test set
evaluate_model(knn_model, X_test, y_test, set_name='Test')
```

```
# Encode categorical variables using one-hot encoding
df_encoded_2 = pd.get_dummies(df_2, columns=['country', 'card'])
# Print the DataFrame after one-hot encoding
print(df_encoded_2)
```

```
# Drop the target variables 'PSP' and 'success' from the features
X_2 = df_encoded_2.drop(['PSP', 'success'], axis=1)

# Initialize the StandardScaler
scaler = StandardScaler()

# Fit and transform the features using StandardScaler
X_scaled_2 = scaler.fit_transform(X_2)

# Create a DataFrame with scaled features and include the 'PSP' column
df_scaled_2 = pd.DataFrame(X_scaled_2, columns=X_2.columns)
df_scaled_2[['PSP']] = df_encoded_2[['PSP']]

# Print the DataFrame with scaled features
print(df_scaled_2)
```

```
# Split the scaled DataFrame into features (X_smote_2) and target variable (y_smote_2)
X_smote_2 = df_scaled_2.drop('PSP', axis=1)
y_smote_2 = df_scaled_2['PSP']
# Initialize the SMOTE with a random state
smote_2 = SMOTE(random_state=42)
# Resample the dataset using SMOTE
X_resampled_2, y_resampled_2 = smote_2.fit_resample(X_smote_2, y_smote_2)
# Create a DataFrame with resampled features and include the 'PSP' column
df_2_resampled = pd.DataFrame(X_resampled_2, columns=X_smote_2.columns)
df_2_resampled['PSP'] = y_resampled_2
# Print the resampled DataFrame
print(df_2_resampled)
```

```
# Split the resampled DataFrame into features (X_2) and target variable (y_2)
X_2 = df_2_resampled.drop('PSP', axis=1)
y_2 = df_2_resampled['PSP']
# Split the dataset into train, validation, and test sets
X_train_2, X_temp_2, y_train_2, y_temp_2 = train_test_split(X_2, y_2, test_size=0.2, ran-
dom_state=42)
X_valid_2, X_test_2, y_valid_2, y_test_2 = train_test_split(X_temp_2, y_temp_2, test_size=0.5,
random_state=42)
# Normalize Data
scaler = StandardScaler()
X_train_scaled_2 = scaler.fit_transform(X_train_2)
X_valid_scaled_2 = scaler.transform(X_valid_2)
X_test_scaled_2 = scaler.transform(X_test_2)
```

```
def evaluate_model(model, X, y, set_name):
    # Make predictions on the data
    y_pred = model.predict(X)
    # Extract the values of the target variable
    y_values = y.values.ravel()  # Convert DataFrame to 1D array
    # Calculate and print relevant evaluation metrics
    precision = precision_score(y_values, y_pred, average='weighted')
    recall = recall_score(y_values, y_pred, average='weighted')
    f1 = f1_score(y_values, y_pred, average='weighted')
    # For binary classification, set multi_class to 'ovr'
    if len(model.classes_) == 2:
        roc_auc = roc_auc_score(y_values, model.predict_proba(X)[:, 1], average='weighted')
    else:
        roc_auc = roc_auc_score(pd.get_dummies(y_values), model.predict_proba(X), average='weighted', multi_class='ovr')
    accuracy = accuracy_score(y_values, y_pred)
    print(f"{set_name} Set Evaluation:")
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"F1 Score: {f1:.4f}")
    print(f"ROC AUC: {roc_auc:.4f}")
    print(f"Accuracy: {accuracy:.4f}")
    print("Confusion Matrix:")
    print(confusion_matrix(y_values, y_pred))
```

```
# Development of Model 2
# Create a Logistic Regression model with specified parameters
logreg_model = LogisticRegression(random_state=42, C=1)
# Perform cross-validation on the training set
cv_scores = cross_val_score(logreg_model, X_train_scaled_2, y_train_2, scoring='f1_weighted', cv=StratifiedKFold(n_splits=5))
# Print cross-validation scores
print("Cross-Validation Scores of Logistic Regression:")
print(cv_scores)
print(f"Average F1 Weighted Score: {cv_scores.mean():.4f}\n")
# Fit the Logistic Regression model on the training set
logreg_model.fit(X_train_scaled_2, y_train_2)
# Print model performance on the validation set
print("Model Performance on Validation set - Logistic Regression:")
evaluate_model(logreg_model, X_valid_scaled_2, y_valid_2, set_name="Validation")
# Print model performance on the test set
print("\nModel Performance on Test set - Logistic Regression:")
evaluate_model(logreg_model, X_test_scaled_2, y_test_2, set_name="Test")
```

```
# Create a Random Forest model with specified parameters
rf_model = RandomForestClassifier(random_state=42, n_estimators=100, max_depth=None)
# Perform cross-validation on the training set
cv_scores_rf = cross_val_score(rf_model, X_train_scaled_2, y_train_2, scoring='f1_weighted',
cv=StratifiedKFold(n_splits=5))
# Print cross-validation scores
print("Cross-Validation Scores:")
print(cv_scores_rf)
print(f"Average F1 Weighted Score: {cv_scores_rf.mean():.4f}\n")
# Fit the Random Forest model on the training set
rf_model.fit(X_train_scaled_2, y_train_2)
# Print model performance on the validation set
print("Model Performance on Validation set - Random Forest:")
evaluate_model(rf_model, X_valid_scaled_2, y_valid_2, set_name="Validation")
# Print model performance on the test set
print("Model Performance on Test set - Random Forest:")
evaluate_model(rf_model, X_test_scaled_2, y_test_2, set_name="Test")
```

```
# Create a Support Vector Machine (SVM) model with specified parameters
svm_model = SVC(probability=True, random_state=42, C=1, gamma='scale')
# Perform cross-validation on the training set
cv_scores_svm = cross_val_score(svm_model, X_train_scaled_2, y_train_2, scor-
ing='f1_weighted', cv=StratifiedKFold(n_splits=5))
# Print cross-validation scores
print("Cross-Validation Scores of SVC:")
print(cv_scores_svm)
print(f"Average F1 Weighted Score: {cv_scores_svm.mean():.4f}\n")
# Fit the SVM model on the training set
svm_model.fit(X_train_scaled_2, y_train_2)
# Print model performance on the validation set
print("Model Performance on Validation set - SVC:")
evaluate_model(svm_model, X_valid_scaled_2, y_valid_2, set_name="Validation")
# Print model performance on the test set
print("Model Performance on Test set - SVC:")
evaluate_model(svm_model, X_test_scaled_2, y_test_2, set_name="Test")
```

```
# Random Forest Model 2 after hyperparameter tuning and cross-validation
# Create a Random Forest model with a set random state
rf_model = RandomForestClassifier(random_state=42)
# Define the parameter grid for hyperparameter tuning
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30]
}

# Use GridSearchCV for hyperparameter tuning
rf_grid = GridSearchCV(rf_model, param_grid, scoring='f1_weighted', cv=StratifiedK-
Fold(n_splits=5), verbose=1)
rf_grid.fit(X_train_scaled_2, y_train_2)
# Get the best model from the grid search
rf_best_model = rf_grid.best_estimator_
# Print the best hyperparameters found
print("Best Hyperparameters:", rf_grid.best_params_)

# Perform cross-validation on the training set with the best model
cv_scores_rf = cross_val_score(rf_best_model, X_train_scaled_2, y_train_2, scor-
ing='f1_weighted', cv=StratifiedKFold(n_splits=5))
# Print cross-validation scores
print("Cross-Validation Scores with Best Model of Random Forest Classification:")
print(cv_scores_rf)
print(f"Average F1 Weighted Score: {cv_scores_rf.mean():.4f}\n")

# Fit the best Random Forest model on the training set
rf_best_model.fit(X_train_scaled_2, y_train_2)
# Print model performance on the validation set
print("Model Performance on Validation set - RandomForestClassifier:")
evaluate_model(rf_best_model, X_valid_scaled_2, y_valid_2, set_name="Validation")
# Print model performance on the test set
print("Model Performance on Test set - RandomForestClassifier:")
evaluate_model(rf_best_model, X_test_scaled_2, y_test_2, set_name="Test")
```

```
#Summary bar plot of the importance of individual features in Model 2
# Create a SHAP TreeExplainer for the best Random Forest Model 2
explainer = shap.TreeExplainer(rf_best_model)
# Calculate SHAP values for the validation set
shap_values = explainer.shap_values(X_valid_scaled_2)
# Define the feature names
feature_names = ["amount", "3D_secured", "transaction_fee", "day_of_week", "minute_of_day",
              "payment_attempts", "success_probabilities", "country_Austria", "country_Ger-
many",
              "country_Switzerland", "card_Diners", "card_Master", "card_Visa", "PSP"]
# Create a summary plot of SHAP values
shap.summary_plot(shap_values[1], X_valid_scaled_2, feature_names=feature_names,
plot_type="bar", show=False)
# Show the plot
```

```
# Precision-Recall curve for validation and test set of Model 2
# Binarize the labels for precision-recall curve
y_valid_2_bin = label_binarize(y_valid_2, classes=np.unique(y_valid_2))
y_test_2_bin = label_binarize(y_test_2, classes=np.unique(y_test_2))
# Predict probabilities on the validation set
y_valid_pred_proba_rf = rf_best_model.predict_proba(X_valid_scaled_2)
# Calculate precision and recall for various thresholds
precision_rf_valid, recall_rf_valid, thresholds_rf_valid = precision_re-
call_curve(y_valid_2_bin.ravel(), y_valid_pred_proba_rf.ravel())
# Calculate AUC for validation set
auc_rf_valid = auc(recall_rf_valid, precision_rf_valid)
# Plot the precision-recall curve for validation set
plt.figure(figsize=(8, 6))
plt.plot(recall_rf_valid, precision_rf_valid, color='blue', label=f'Random Forest (Validation) -
AUC: {auc_rf_valid:.4f}')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve - RandomForestClassifier (Validation)')
plt.legend()
plt.grid(True)
plt.show()
```

```
# Predict probabilities on the test set
y_test_pred_proba_rf = rf_best_model.predict_proba(X_test_scaled_2)
# Calculate precision and recall for various thresholds
precision_rf_test, recall_rf_test, thresholds_rf_test = precision_re-
call_curve(y_test_2_bin.ravel(), y_test_pred_proba_rf.ravel())
# Calculate AUC for test set
auc_rf_test = auc(recall_rf_test, precision_rf_test)
# Plot the precision-recall curve for test set
plt.figure(figsize=(8, 6))
plt.plot(recall_rf_test, precision_rf_test, color='green', label=f'Random Forest (Test) - AUC:
{auc_rf_test:.4f}')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve - RandomForestClassifier (Test)')
plt.legend()
plt.grid(True)
plt.show()
```

```
#ROC Curve on validation and test set of Model 2
def evaluate_model_with_roc(model, X, y, set_name="Set"):
    lb = LabelBinarizer()
    y_bin = lb.fit_transform(y)
    y_pred_prob = model.predict_proba(X)
    # Compute ROC curve and ROC area for each class
    n_classes = y_bin.shape[1]
    fpr = dict()
    tpr = dict()
    roc_auc = dict()
    for i in range(n_classes):
        fpr[i], tpr[i], _ = roc_curve(y_bin[:, i], y_pred_prob[:, i])
        roc_auc[i] = auc(fpr[i], tpr[i])
    # Plot ROC curve
    plt.figure(figsize=(8, 6))
    for i in range(n_classes):
        plt.plot(fpr[i], tpr[i], lw=2, label=f'Class {i} (AUC = {roc_auc[i]:.2f})')
    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'ROC Curve - {set_name}')
    plt.legend(loc='lower right')
    plt.show()
# Evaluate on Validation Set with ROC Curve
print("Model Performance on Validation set - RandomForestClassifier:")
evaluate_model_with_roc(rf_best_model, X_valid_scaled_2, y_valid_2, set_name="Validation
Set")
# Evaluate on Test Set with ROC Curve
print("Model Performance on Test set - RandomForestClassifier:")
evaluate_model_with_roc(rf_best_model, X_test_scaled_2, y_test_2, set_name="Test Set")
```

```
# Pie chart for analysing correctly and misclassified classified on the validation Set of Model  2
conf_matrix = np.array([[2604, 4, 0, 0],
                        [15, 2604, 0, 1],
                        [0, 0, 2616, 10],
                        [0, 0, 3, 2727]])
# Calculate total correctly classified instances
correctly_classified_total = np.sum(np.diag(conf_matrix))

# Calculate total misclassified instances
misclassified_total = np.sum(conf_matrix) - correctly_classified_total
# Print the results
print(f"Total Correctly Classified Instances on validation set of model 2: {correctly_classified_to-
tal}")
print(f"Total Misclassified Instances on validation set of model 2: {misclassified_total}")

# Data for validation set
labels_validation = ['Correctly Classified', 'Misclassified']
sizes_validation = [correctly_classified_total, misclassified_total]
colors_validation = ['lightgreen', 'lightcoral']
```

```
# Plot pie chart for validation set
plt.figure(figsize=(8, 8))
plt.pie(sizes_validation, labels=labels_validation, colors=colors_validation, autopct='%1.1f%%',
startangle=90, shadow=True)
plt.title('Validation Set - Correctly vs. Misclassified Instances')
plt.show()
```

```
# Pie chart for analysing correctly and misclassified classified on the Test Set of Model  2
conf_matrix = np.array([[2584, 6, 0, 0],
                        [17, 2604, 0, 0],
                        [0, 0, 2633, 16],
                        [0, 0, 5, 2719]])

# Calculate total correctly classified instances
correctly_classified_total = np.sum(np.diag(conf_matrix))

# Calculate total misclassified instances
misclassified_total = np.sum(conf_matrix) - correctly_classified_total

# Print the results
print(f"Total Correctly Classified Instances on test set of model 2: {correctly_classified_total}")
print(f"Total Misclassified Instances on test set of model 2: {misclassified_total}")

# Data for test set
labels_test = ['Correctly Classified', 'Misclassified']
sizes_test = [correctly_classified_total, misclassified_total]
colors_test = ['lightgreen', 'lightcoral']  # Match the colors used in code1

# Plot pie chart for test set
plt.figure(figsize=(8, 8))
plt.pie(sizes_test, labels=labels_test, colors=colors_test, autopct='%1.1f%%', startangle=90,
shadow=True)
plt.title('Test Set - Correctly vs. Misclassified Instances')
plt.show()
```

```
# Cross-Validation stability of Model 2

# Cross-validation scores with the best model

cv_scores_rf = cross_val_score(rf_best_model, X_train_scaled_2, y_train_2, scor-
ing='f1_weighted', cv=StratifiedKFold(n_splits=5))

print("Cross-Validation Scores with  Best Model of Random Forest Classification:")

print(cv_scores_rf)

print(f"Average F1 Weighted Score: {cv_scores_rf.mean():.4f}")

print(f"Standard Deviation of F1 Weighted Scores: {cv_scores_rf.std():.4f}")

# Individual cross-validation scores

for i, score in enumerate(cv_scores_rf):

    print(f"Fold {i + 1}: {score:.4f}")

# Train the model on the entire training set

rf_best_model.fit(X_train_scaled_2, y_train_2)
```

```
# Overfitting Analysis of Model 2

def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None, n_jobs=None,
train_sizes=np.linspace(.1, 1.0, 5)):

    plt.figure()

    plt.title(title)

    if ylim is not None:

        plt.ylim(*ylim)

    plt.xlabel("Training examples")

    plt.ylabel("F1 Score")

    train_sizes, train_scores, test_scores = learning_curve(estimator, X, y, cv=cv, n_jobs=n_jobs,
train_sizes=train_sizes, scoring='f1_weighted')

    train_scores_mean = np.mean(train_scores, axis=1)

    train_scores_std = np.std(train_scores, axis=1)

    test_scores_mean = np.mean(test_scores, axis=1)

    test_scores_std = np.std(test_scores, axis=1)

    plt.grid()

    plt.fill_between(train_sizes, train_scores_mean - train_scores_std,

                     train_scores_mean + train_scores_std, alpha=0.1,

                     color="r")

    plt.fill_between(train_sizes, test_scores_mean - test_scores_std,

                     test_scores_mean + test_scores_std, alpha=0.1, color="g")
    plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
             label="Training score")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
             label="Cross-validation score")
    plt.legend(loc="best")
    return plt
# Plot learning curves
plot_learning_curve(rf_best_model, "Model 2 Learning Curve - Random Forest",
X_train_scaled_2, y_train_2, cv=StratifiedKFold(n_splits=5), n_jobs=-1)
plt.show()
```

```
# Optimal PSP prediction for each exiting transaction
# Extract 'PSP' column for comparison
psp_column = df_2['PSP']
# Drop 'PSP' and 'success' columns from the features
X_predict = df_2.drop(['PSP', 'success'], axis=1)
# Define numeric features and apply StandardScaler
numeric_features = ['amount', '3D_secured', 'transaction_fee', 'day_of_week', 'minute_of_day',
'payment_attempts', 'success_probabilities']
numeric_transformer = StandardScaler()
# Define categorical features and apply OneHotEncoder
categorical_features = ['country', 'card']
categorical_transformer = OneHotEncoder()
# Create a ColumnTransformer for preprocessing
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)
    ])

# Transform the features using the preprocessor
X_predict_transformed = preprocessor.fit_transform(X_predict)
# Predict the 'PSP' values using the best Random Forest model
df_2['Predicted_PSP'] = rf_best_model.predict(X_predict_transformed)
```

```
# Restore the original 'PSP' values for comparison
df_2['PSP'] = psp_column
# Print the DataFrame with predicted 'PSP' values
print(df_2)
```

```
# Handle ties and choose the most cost-effective PSP

def choose_best_psp(row):
    if pd.Series(row['Predicted_PSP']).nunique() == 1:
        # No tie, return the predicted PSP
        return row['Predicted_PSP']
    else:
        # There is a tie, choose the most cost-effective PSP based on fee comparison
        min_fee_psp = row.loc[row['transaction_fee'].idxmin()]['Predicted_PSP']

        # Filter rows with tied predictions
        tied_rows = row[row['Predicted_PSP'].duplicated(keep=False)]

        # Check if the row with the minimum fee is part of the tied rows
        if min_fee_psp in tied_rows['Predicted_PSP'].values:
            return min_fee_psp
        else:
            # If not, choose the PSP with the lowest transaction fee
            return tied_rows.loc[tied_rows['transaction_fee'].idxmin()]['Predicted_PSP']

# Apply the function to each row in df_2
df_2['Best_PSP'] = df_2.apply(choose_best_psp, axis=1)

# Display the updated DataFrame with the 'Best_PSP' column
print(df_2)
```

```
# Count occurrences of Predicted_PSP
predicted_psp_counts = df_2['Predicted_PSP'].value_counts()

# Count occurrences of Best_PSP
best_psp_counts = df_2['Best_PSP'].value_counts()

# Combine the counts into a DataFrame
matching_counts = pd.DataFrame({
    'Predicted_PSP_Count': predicted_psp_counts,
    'Best_PSP_Count': best_psp_counts
})

# Display the counts
print(matching_counts)
```