CASE STUDY: Model Engineering
(DLMDSME01)

A Written Assignment on

Credit Card Routing for Online Purchase via Predictive Modelling

Author: Samyak Anand

Matriculation Number: 321149613

Tutor's Name: Mohammad Reza Nilchiyan, Sahar Qaadan

Date: 24 Mar 2024

Place: Berlin, Germany

Content

List of Figures

List of Tables

List of Outputs

1. Introduction

In the dynamic landscape of e-commerce, swift processing of credit card transactions is crucial for both ensuring a positive customer experience and achieving financial success. As a recently appointed data scientist at a leading global retailer, our initial challenge is to address a significant spike in the failure rate of online credit card payments observed over the past year. This issue has resulted in substantial financial losses for the company and has left customers unsatisfied with their online shopping interactions. The existing manual and rule-based credit card routing system managed by payment service providers (PSPs) has proven inadequate in tackling this problem. To respond to this challenge, business stakeholders are seeking a transformative solution through predictive modelling.

The primary objective is clear: develop an automated credit card routing system that not only enhances payment success rates but also strategically reduces transaction costs. This project follows a comprehensive approach based on the Cross-Industry Standard Process for Data Mining (CRISP-DM). It introduces a two-model strategy aimed at achieving the dual objectives of improving success rates and cutting transaction costs. The overarching aim of this initiative is to revolutionize credit card routing in the online retail sector through a blend of investigative methodologies, data analysis, preparation, modelling, evaluation, and deployment. This approach aligns with broader business goals, emphasizing financial efficiency and customer satisfaction.

On your inaugural day as a data scientist at one of the world's largest retail corporations, you are swiftly immersed in a meeting with crucial stakeholders from the online payment department. They urgently seek your assistance in addressing a significant rise in the failure rate of online credit card payments over the past year. This surge in failed transactions not only results in substantial financial losses for the company but also leads to growing dissatisfaction among customers using the online store.

1.1 Problem Definition

The online credit card payments are facilitated through payment service providers (PSPs), and the company currently holds contracts with four different PSPs, incurring transaction fees for each payment. The existing credit card routing system relies on manual and rule-based logic, but the business leaders are optimistic that, with your expertise and the application of predictive modelling, a more intelligent PSP (Payment Service Provider) routing approach can be devised.

1.2 Project Objective

The primary goal is to assist the business in automating credit card routing through the implementation of a predictive model. This model aims to enhance the payment success rate by identifying the most suitable PSP for each transaction while simultaneously managing transaction fees to ensure cost-effectiveness.

1.3 Data Set

The relevant data set, including information about PSPs, transaction fees, and other pertinent details, is provided in a dedicated *.zip folder accessible through myCampus, under the Case Study section.

1.4 Project Approach

1. Project Structure and Git Repository Proposal
   Organize the project following the CRISP-DM or Team DS methodologies to ensure a systematic approach. Propose a Git repository structure that aligns with best practices, considering separate branches for data exploration, model development, and evaluation. Utilize subfolders for datasets, code, and documentation to enhance collaboration and version control.
2. Data Set Quality Assessment and Initial Data Analysis
   Evaluate the quality of the provided dataset, checking for completeness, accuracy, and consistency. Perform an initial data analysis to extract meaningful insights. Present the findings through clear visualizations to facilitate comprehension by business stakeholders. This step is crucial for establishing a foundational understanding of the dataset.
3. Baseline and Predictive Models
   Develop a baseline model to establish a benchmark for comparison. Subsequently, create an accurate predictive model that addresses the business requirements of increasing credit card success rates while minimizing transaction fees. Utilize appropriate machine learning algorithms and hyperparameter tuning to optimize model performance.
4. Feature Importance and Model Interpretability
   Discuss the importance of individual features in the predictive model, emphasizing how they contribute to the overall outcome. Ensure interpretability by using techniques such as feature importance plots or SHAP (SHapley Additive exPlanations) values. Conduct a sophisticated error analysis to provide insights into model limitations, helping business stakeholders understand the potential drawbacks of the approach.
5. Model Integration into Everyday Business Operations
   Propose a practical implementation of the model in the business's daily operations. Consider suggesting a graphical user interface (GUI) to facilitate user interaction, providing an intuitive platform for stakeholders to input data and receive model predictions. This step ensures seamless integration and usability of the developed model.
6. Code Submission
   Include the code as part of the final submission document. Organize the codebase according to best practices, including proper commenting and documentation to enhance readability and reproducibility. Ensure that the code aligns with the proposed Git repository structure, allowing for easy navigation and collaboration.

By addressing these tasks, the final document will provide a comprehensive and actionable framework for implementing the credit card routing predictive model, emphasizing transparency, interpretability, and practical usability for the business stakeholders.

The efficacy of a dual-model strategy, encompassing success prediction (Model A) and payment service provider (PSP) selection (Model B), lies in its potential to enhance payment

success rates and minimize transaction fees. The choice of diverse modelling techniques, including K-Nearest Neighbor (KNN), Logistic Regression, Decision Tree Classification, and Random Forest Classification, significantly influences the effectiveness of achieving these objectives. How can the utilization of these models positively impact payment success rates and transaction fees reduction?

Among the modelling strategies, namely **K-Nearest Neighbor (KNN)**, **Logistic Regression**, **Decision Tree Classification**, **XGBoost** and **Random Forest Classification**, which one excels in predicting success probability and determining the most optimal payment service provider (PSP)? In terms of accuracy, interpretability, and overall effectiveness in credit card routing, how do these selected models compare to a baseline model?

1.5 Scope

- The case study aims to tackle the prevalent issue of high credit card transaction failure rates in online retail. This will be achieved through the utilization of historical data and machine learning techniques, specifically employing two predictive models (**KNN, Logistic Regression, Decision Tree, XGBoost, Random Forest**).
- The primary goals involve enhancing success rates while concurrently reducing transaction costs.
- Collaboration with various departments is essential to thoroughly examine extensive data, emphasizing the importance of interpretability and transparency in the process.

1.6 Significance

- The automation of credit card routing possesses the potential to revolutionize online retail payment systems.
- Implementation of a CRISP-DM-aligned two-model strategy is pivotal for addressing the current challenges.
- The focus is on minimizing financial losses and elevating customer satisfaction levels.
- The initiative contributes significantly to both financial and customer satisfaction objectives.
- Integration of machine learning techniques is intended to optimize the efficiency of online payment processes.
- Contingency plans are imperative for mitigating risks, and effective stakeholder communication is prioritized.
- A deployment strategy is essential for ensuring long-term optimization of the implemented solution.

2. Git Repository Proposal

Git Repository Proposal is a planning document that outlines the organization of a version-controlled codebase using Git. It includes details about branch structure, folder hierarchy, and documentation practices, promoting clarity, collaboration, and adherence to version control best practices. The benefits include efficient development workflows, clear documentation, reproducibility, ease of onboarding for new team members, quality assurance, and preparation for project scalability. Overall, it enhances collaboration, code organization, and project management in the development process. (git-scm.com)

2.1 Git Repository Proposal: Credit Card Routing Optimization

1. **Branch Structure:**
   master: Main branch for stable, production-ready code.
   dev: Development branch for ongoing work and feature integration.
   feature/modeling: Branch for model-specific developments (e.g., KNN, Logistic Regression, XGBoost, Decision Tree and Random Forest).
   feature/evaluation: Branch for ongoing work on model evaluation and analysis.

   data-exploration: Branch for exploratory data analysis and initial data understanding.
2. **Folder Structure:**
   /data/raw: Raw datasets, including 'raw_data_psp_jan_feb_2019.xlsx'.
   /data/processed: Processed datasets, including 'cleaned_data_psp_jan_feb_2019.xlsx' and 'features/selected_features_psp_jan_feb_2019.xlsx'.
   /notebooks: Jupyter notebooks for exploratory data analysis, data preparation, modeling, and evaluation.
   /notebooks/modelling: Model-specific notebooks for training (e.g., 'model_1_training.ipynb', 'model_2_training.ipynb').
   /notebooks/evaluation: Notebooks for detailed evaluation and analysis (e.g., confusion matrix, precision-recall, ROC curve, misclassified samples inspection, cross-validation stability, overfitting analysis).
3. **/src:**
   /src/data: Scripts for data-related tasks (e.g., data_loading.py, data_cleaning.py, feature_engineering.py, data_splitting.py).
   /src/modeling: Scripts for model development (e.g., model_1.py, model_2.py).
   /src/evaluation: Scripts for model evaluation and analysis (e.g., confusion_matrix_analysis.py, precision_recall_tradeoff.py, roc_curve_analysis.py, misclassified_samples_inspection.py, cross_validation_stability.py, overfitting_analysis.py).
4. **Documentation:**
   README.md: Main documentation providing an overview of the project, installation instructions, and usage guidelines.
   requirements.txt: File listing all project dependencies for easy setup.
   .gitignore: File specifying which files and directories to ignore in version control.
   By adopting this Git repository structure, the project will benefit from clear organization, facilitating collaboration, version control, and seamless integration of various components. Regularly merging feature branches into the 'dev' branch ensures continuous development, while the 'master' branch represents a stable, deployable version of the code. Regular tags can be used for versioning and milestones.

3. Cross-Industry Standard Process for Data Mining [CRISP-DM]

CRISP-DM stands as a highly prevalent and firmly established framework for executing data mining initiatives. Offering a systematic methodology, it serves as a guiding tool for practitioners in uncovering valuable insights and knowledge from datasets. The genesis of CRISP-DM lies in a collaborative effort by a consortium of businesses recognized as the

CRISP-DM Consortium, featuring leading entities actively engaged in the realm of data mining.

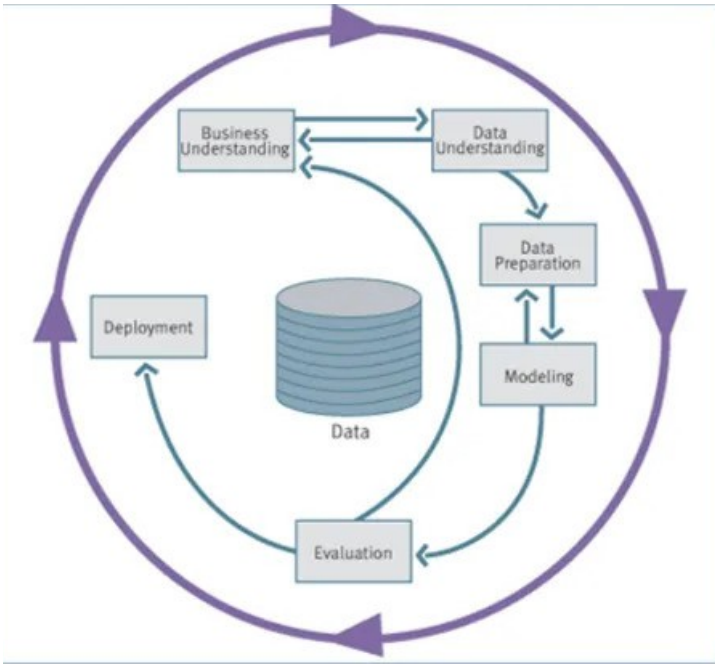


Figure 1: The CRISP-DM methodology.

The CRISP-DM methodology unfolds through six distinct stages:

a) **Comprehending Business Objectives:** In the initial phase, the focus is on defining project goals and requirements from a business standpoint. This entails a thorough understanding of the data mining project's objectives, clarifying stakeholder intentions, and establishing benchmarks for success.
b) **Exploring Data:** The second phase involves the collection and exploration of data, aiming to grasp its quality and structure. Identification of data sources and an initial exploration shed light on the inherent characteristics of the dataset.
c) **Data Refinement:** Within the data preparation phase, emphasis is placed on preprocessing and cleansing to ensure data quality and suitability for analysis. Tasks include addressing missing values, transforming variables, and selecting pertinent features.
d) **Model Crafting:** The modeling phase entails the selection and application of diverse data mining techniques to the prepared dataset. This encompasses the creation of models through algorithms such as decision trees, neural networks, or statistical methods tailored to the project's specific objectives.
e) **Performance Evaluation:** Models developed in the preceding phase undergo rigorous evaluation to verify their efficacy in meeting business goals. This involves scrutinizing performance metrics, fine-tuning parameters, and selecting the optimal model for deployment.
f) **Application of Insights:** In the ultimate deployment phase, the insights derived from the data mining process are strategically applied to inform and shape business decisions. Integration into existing systems, report generation, or the implementation of tailored strategies align with the overall objectives of the project.

3.1 Comprehending Business Objectives

In the initial step of our CRISP-DM approach, a thorough comprehension of the retail company's objectives is paramount. This business understanding phase requires the identification of critical factors that could impact the project's outcome, leading to the formulation of tasks and the outline of a preliminary method ("CRISP DM Step 1 - Understanding About the Business," n.d.).
Within the CRISP-DM methodology's business understanding phase, various essential elements come into play, particularly in the context of the specified use case focused on "Credit Card Routing for Online Purchase via Predictive Modelling." This phase's conclusion should involve the development of an initial project plan. This plan should delineate anticipated resource requirements, specific needs, assumptions, constraints, risks, or contingencies, along with a description of the tools and techniques to be employed in the project (Bhagwat, n.d.).

3.2 Data Understanding

The initiation of the data understanding phase involves an initial data collection, followed by various activities aimed at familiarizing oneself with the data. These actions encompass the identification of data quality issues, uncovering initial insights, and detecting intriguing subsets for the formulation of hypotheses regarding concealed information. This crucial phase is pivotal for analysts as it facilitates a comprehensive understanding of the data and ensures that its quality is suitable for accurately portraying relationships, establishing a reliable foundation for drawing precise inferences (MyEducator - CRISP-DM, n.d.).

3.2.1 Collect Data

Enumerate the datasets procured (including their sources, methods of acquisition, challenges faced, and innovative solutions implemented)

3.2.2 Elaborate on Data

Evaluate the extent of data and scrutinize its fundamental properties. Assess the accessibility and availability of attributes, considering their types, ranges, correlations, and distinctive identities. Decipher the business significance of each attribute and its values. For each attribute, compute fundamental statistics (e.g., distribution, mean, maximum, minimum, standard deviation, variance, mode, skewness).

1. **Explore Data**

   Dive into the intricacies of noteworthy attributes. Analize distributions, relationships among specific pairs of attributes, characteristics of significant sub-populations, and engage in straightforward statistical analyses.

2. **Verify Data Quality**
   Identify distinct values and document their nuanced meanings. Confirm the coverage of all requisite cases and gauge the prevalence of errors. Pinpoint missing attributes and vacant fields, discerning the implications of absent data. Scrutinize the harmony between attribute meanings and their encapsulated values. Validate the consistency

of spelling (e.g., uniform capitalization). Evaluate the plausibility of values, ensuring uniformity across all fields.

3.3 Data Preparation

This encompasses all actions involved in crafting the final dataset from the initial raw data. The transformation of raw data into an analytical dataset holds immense importance, as the quality of the cleansed data directly influences model performance. Data preparation tasks are dynamic, often performed iteratively, and not confined to a specific order. These tasks encompass the selection of tables, records, and attributes, along with the transformation and cleansing of data tailored for modelling tools.

1. **Choose Data**
   - Re-evaluate criteria for data selection.
   - Determine the dataset to be utilized.
   - Gather relevant additional data, whether internal or external.
   - Explore the use of sampling techniques.
   - Provide rationale for the inclusion or exclusion of specific data.

2. **Refine Raw Data:** Refine Raw Data through Pre-processing Techniques such as
   - Analysis of Missing Values
   - Examination of Outliers
   - Implementation of Feature Engineering
   - Scaling of Numerical Variables
   - Sampling
   - Smoothing

3.4 Model Development

In this stage, diverse modelling techniques are chosen and applied, with parameters fine-tuned for optimal values. Typically, multiple techniques are available for the same type of data mining problem. Some techniques impose specific data format requirements, necessitating a potential return to the data preparation phase.
For this project, we created a baseline model and use various machine learning algorithms to evaluate the model. Algorithms used are **Linear Regression, KNN, Decision Tree, XGBoost** and **Random Forest**.

3.4.1. Project Approach

Our project objectives revolve around enhancing payment success rates and reducing transaction fees, prompting the formulation of a distinctive project plan methodology. This approach involves the creation of two independent models, each catering to a specific goal:

- **Single Approach:**
  Model: Success Prediction:
  Predicts success probability with a primary focus on elevating success rates.
  The success probability derived from Model A serves as input for the sub model.
  Sub Model: PSP Selection:

Identifies the optimal payment service provider (PSP) based on success probabilities and transaction fees. Facilitates dynamic decision-making by evaluating both success probabilities and costs.

- **Implementation Steps:**
  Train Model: Utilizes transaction-related features (excluding fees) with success as the target variable. Aims to predict success probabilities.
  Predict Success Probabilities: Applies Model A to generate success probabilities.
  Train Sub Model: Incorporates success probabilities from Model, transaction fees, and relevant features, with PSP as the target variable. Aims to predict the optimal PSP, considering both success probabilities and fees.
  Routing Decision: Utilizes success probabilities from Model A and Sub Model predictions to determine the best PSP. Leverages fee comparison features in case of ties.
- **Benefits:**
  Specialization: Model A optimizes success rates. Sub Model minimizes transaction fees while factoring in success probabilities.
  Interpretability: The segregation of models provides transparency for each facet of decision-making.

3.4.2. Prerequisites, Presumptions, and Limitations

- **Prerequisites**:
  Necessity for historical transaction datasets to facilitate model training.
  Collaboration with both the online payment department and IT teams.
- **Presumptions**:
  The predictive model is assumed to have access to precise and comprehensive data.
  Instances where transactions occur within a minute, involve the same amount, as prompt clients to make numerous transfer attempts.
- **Limitations**:
  Adherence to regulatory constraints governing online payment processing.
  Constraints imposed by existing contractual obligations with four distinct payments service providers.

3.4.3 Potential Risks:

- Potential inaccuracies present in historical transactional data.
- The likelihood of regulatory modifications influencing the routing of credit card transactions.

3.4.4 Precautionary Measures:

- Deployment of stringent data validation checks and comprehensive cleaning processes.
- Maintaining proactive awareness of regulatory shifts and adjusting the model accordingly.

3.4.5 Tools and Methodologies:

- The integration of predictive modeling techniques, particularly machine learning algorithms, to automate credit card routing.
- Implementation of data preparation methodologies for handling and cleansing historical transactional data.

3.4.6 Harmonization with Business Objectives:

- The project seamlessly aligns with the overarching business goals of the retail company, including.
- Boosting financial performance by curbing losses stemming from unsuccessful transactions.
- Elevating customer satisfaction through an enhanced online shopping experience.

3.4.7 Stakeholder Interaction:

- Establishing robust connections with key stakeholders from the online payment department.
- Sustaining continual collaboration and feedback loops throughout the duration of the project.

3.5 Relevant Data

The indispensable data for this case study is encapsulated within the file labelled "**PSP_Jan_Feb_2019.xlsx.**" This file meticulously documents credit card transactions spanning the DACH countries, namely Germany, Switzerland, and Austria. Moreover, it comprehensively includes vital business details such as PSP names and transaction costs. In the realm of datasets, column names serve as markers denoting features or variables, each one indicative of a unique attribute or characteristic within the data. These features within the dataset furnish insights into the various observations or incidents under scrutiny.

3.5.1 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.

3.6 Quality Assessment

Defining, measuring, analysing, and continually enhancing Data Quality (DQ) are crucial for maintaining data of high quality. (Strong, Lee, & Wang, 1997) The process of DQ assessment involves several key steps that organizations, users, and developers should undertake (R. Y. Wang, 1998) (Aljumaili, Rauhala, Tretten, & Karim, 2011):

a) In the definition step, significant DQ dimensions are identified within the relevant context.

b) The measurement step involves defining and generating metrics and measures necessary for evaluating DQ effectively.
c) The analysis step aims to pinpoint the root causes of DQ issues and assess the impact of subpar data quality.
d) Lastly, in the improvement step, appropriate techniques are recommended to enhance and uplift the overall Data Quality.

Examining Data Quality (DQ) dimensions is indispensable for organizations across diverse industries for several compelling reasons. Firstly, it enables the evaluation and quantification of data quality. Secondly, it establishes a structure for formulating DQ guidelines and enhancement strategies. When crafting these measures, a company must ascertain what aspects are to be evaluated and which set of DQ dimensions holds significance aligned with its mission and operational needs. (Y. W. Lee et al., 2006). Assessing the extent to which the dataset lacks missing or incomplete values, ensuring a comprehensive representation of the intended information. Verifying the precision of data entries by comparing them against trusted sources or established benchmarks to identify and rectify any discrepancies. Evaluating the relevance of the data concerning its recency and appropriateness for the current analysis, ensuring that outdated information does not compromise the results. Ensuring uniformity and coherence in data entries across the dataset, identifying and rectifying any discrepancies or contradictions. Verifying the appropriateness of the data for the specific analysis at hand, ensuring that irrelevant or redundant information does not impact the outcomes.

Data quality assessment is an iterative process, often performed in conjunction with data preparation, to address any issues identified and refine the dataset before proceeding with further analysis or modeling. It is a crucial step in establishing the credibility and efficacy of insights drawn from the data. (Atkinson, 2006)

The evaluation of data quality for the provided use case involves the following steps:

3.6.1 Assessment of Missing Values

Each column is scrutinized for any instances of missing values, with the percentage of missing values calculated for each column. Based on the code output, it is observed that there are no missing values across any columns in the dataset. The "Missing Values" column displays zeros for each variable, indicating completeness with respect to field values.

3.6.2 Validation of Data Types

Verification is conducted to ensure that the data types in each column align with the specified criteria and descriptions. The examination indicates that the data types conform to the criteria and descriptions for each respective column. Here is a breakdown of the identified data types:

Data Type Information:

- Unnamed: 0 int64
- tmsp datetime64[ns]
- country object

- amount int64
- success int64
- PSP object
- 3D_secured int64
- card object

The mentioned data types seem fitting for each respective column, considering the provided information. The "**tmsp**" column is appropriately designated as a datetime type, and the remaining columns are assigned the required numeric or categorical types as needed.

3.6.3 Detecting Systematic Deviations

The aim is to uncover patterns or consistent deviations that might signal challenges in acquisition, processing, or transfer mechanisms. Upon inspection, the distinct numbers in each column seem reasonable, and there is no apparent presence of systematic alterations or issues in the acquisition, processing, or transfer processes. A summary of each column is outlined below:

a) Unnamed: 0: This column represents an index, and the unique values exhibit an orderly, ascending sequence without any irregularities.
b) tmsp: Timestamps are formatted as datetime, and the unique values fall within the expected date and time range.
c) country: The column contains the anticipated countries (Germany, Austria, Switzerland) without any unexpected values.
d) amount: Various numeric values for transaction amounts are present without any discernible issues.
e) success: A binary variable with values 0 and 1, indicating the success or failure of the payment.
f) PSP: Payment service provider names include 'UK_Card,' 'Simplecard,' 'Moneycard,' and 'Goldcard,' which appear fitting.
g) 3D_secured: A binary variable with values 0 and 1, denoting whether the customer is 3D identified.
h) card: Credit card provider names such as 'Visa,' 'Diners,' and 'Master' seem appropriate.

The unique values in each column align with expectations, and there are no discernible systematic deviations that could present significant issues.

The initial stage of data analysis involves data exploration, a process utilized to examine data visually and comprehensively, either to unveil immediate insights or to identify specific areas or patterns warranting further investigation (Spotfire | Unveiling Data Exploration, n.d.). This approach is not aimed at displaying every detail within a dataset; instead, it serves to construct a general overview of significant trends and crucial subjects, guiding subsequent in-depth studies. Moreover, data exploration can be categorized into two distinct types.

4. Preliminary Data Exploration

4.1 Comprehensive Understanding of the Dataset

This phase centres on obtaining a comprehensive understanding of the dataset and acquiring initial insights. It involves generating histograms for each variable to examine the distribution of the data. Following the removal of the unnamed column, the provided dataset comprises 50,410 rows and 7 columns.

Table 1: Statistics summary

	count	mean	std	min	25.00%	50.00%	75.00%	max
amount	50410	202.395715	96.27473	6	133	201	269	630

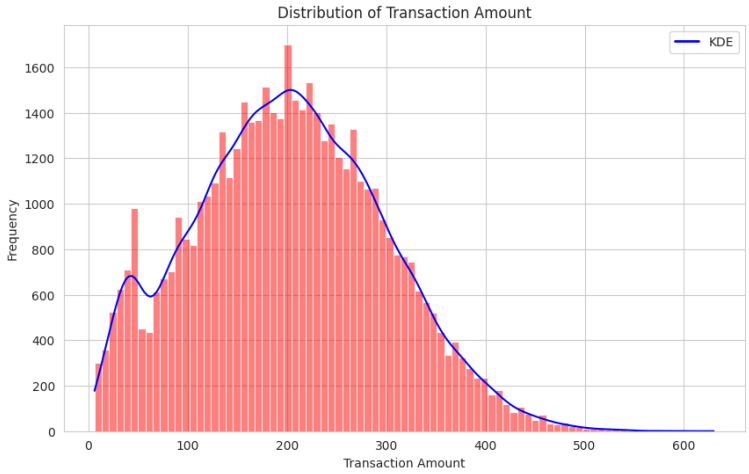


Figure 2 of Transaction Amount

Based on the provided summary statistics, it is evident that most transactions (50%) involved amounts that were less than or equal to 201.0. However, the dataset exhibits variability, as certain transactions recorded higher values, reaching a maximum of 630.0.

The histogram illustrates a distribution of transaction amounts that skews to the right. Most transactions involve smaller sums, creating a prominent peak on the left side. Conversely, a tail on the right indicates fewer transactions with larger sums, contributing to the extended tail of the distribution.

Moreover, based on the summary transactions (0) and 10,228 successful transactions (1). The distribution of the 'success' column is depicted in Figure above Similarly, for the '3D_secured' column, the summary statistics reveal 38,399 transactions with 3D security and 12,011 transactions without 3D security.

Summary statistics for 'Success' column:
success
0 40182

Summary statistics for '3D_secured' column:
3D_secured.
0 38399

1 10228
Name: count, dtype: int64

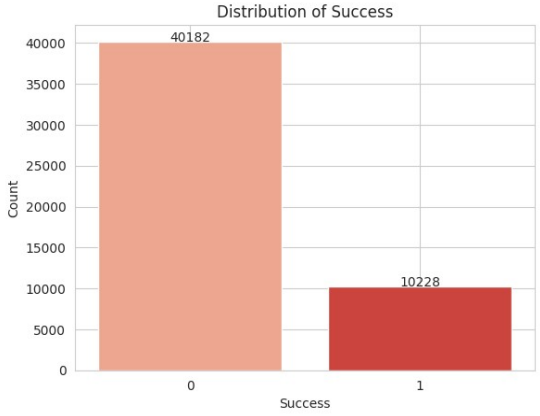


Figure 3: Distribution of Success

1 12011
Name: count, dtype: int64

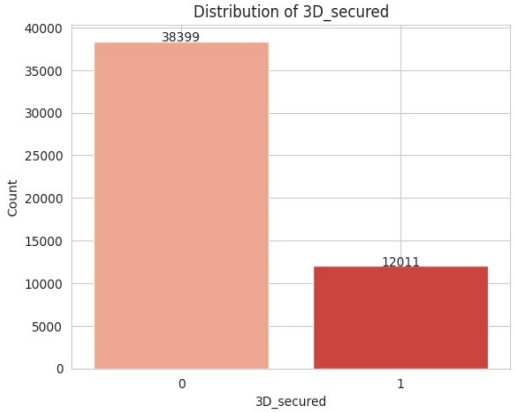


Figure 4: Distribution of 3D_secured

Additionally, the summary statistics pertaining to the 'country' column reveal that there are 30,233 entries for Germany, 10,338 entries for Switzerland, and 9,839 entries for Austria.

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

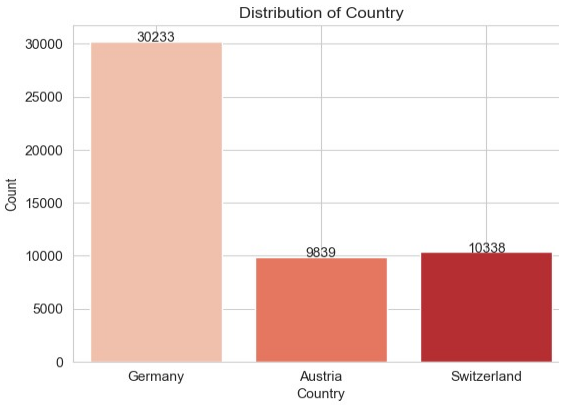


Figure 5: Distribution of Country

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

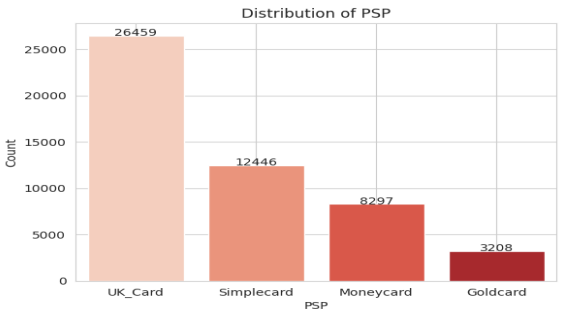


Figure 6: Distribution of PSP

As per the summary statistics of the 'PSP' column, there are 26,459 occurrences for UK_Card, 12,446 for Simplecard, 8,297 for Moneycard, and 3,208 for Goldcard.

The distribution of the 'card' column exhibits variability, where MasterCard emerges as the most frequently occurring option, with 29,002 instances. Following closely, Visa ranks as the second most prevalent card type, reported in 11,640 instances, while the Diners card is a close third with 9,768 instances.

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

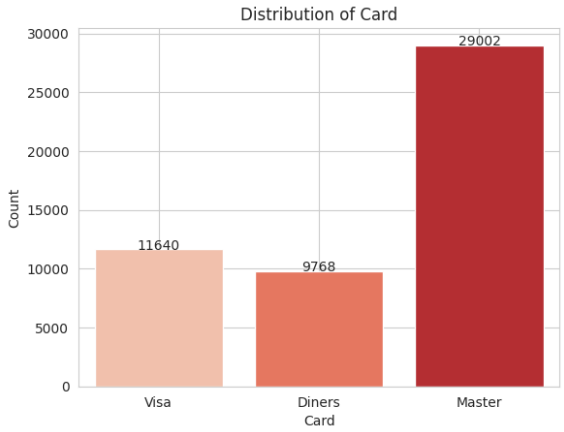


Figure 7: Distribution of card

This stage involves a thorough examination of the data to uncover intricate details and patterns. It includes a deeper analysis by exploring variable combinations, assessing variable correlations, and scrutinizing the behaviours of variables in relation to each other. Employing a boxplot to visualize the correlation between 'amount' and 'success' becomes crucial in comprehending how transaction amounts vary concerning the success or failure of transactions.

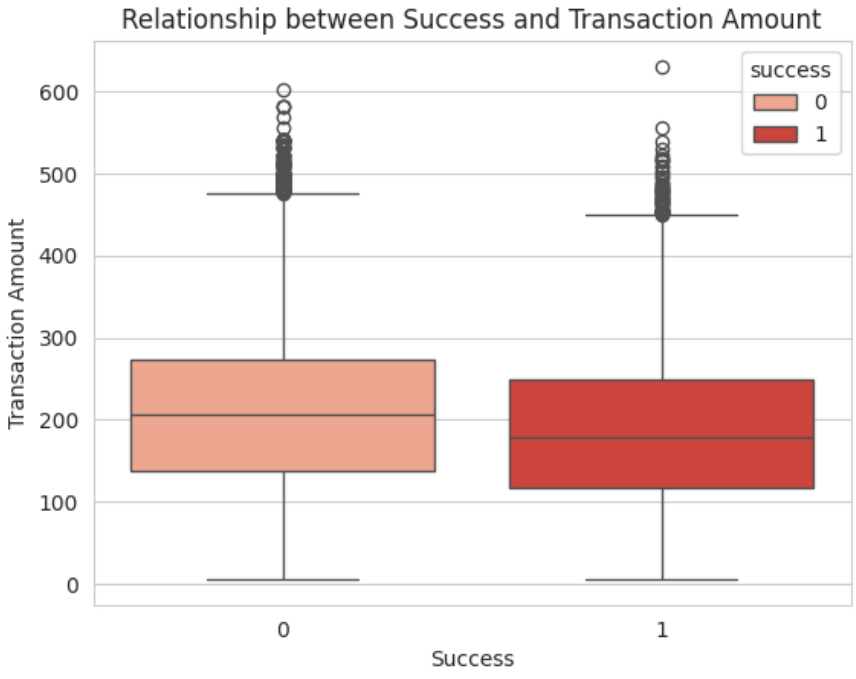


Figure 8: Boxplot to visualize the correlation between 'amount' and 'success'

The discrepancies in median values and the existence of data points beyond the whiskers in Figure 9 below indicate differences in transaction amounts between successful and unsuccessful transactions. Additionally, while the pair plot serves as a valuable tool, its effectiveness hinges on the dataset's structure and the objectives of our investigation. The pair plot is employed to visually represent provided data, offering insights into the relationships between variables, which can be either continuous or categorical (SL, 2020).

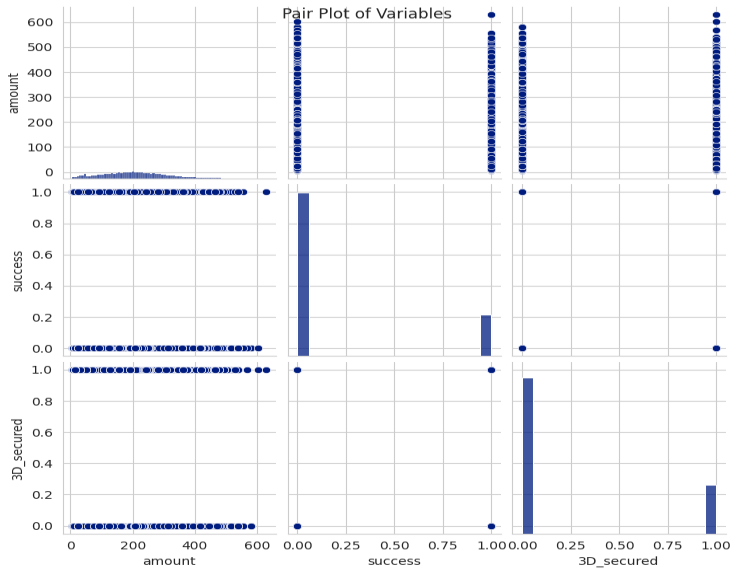


Figure 9: Pair Plot of Variables

The insights drawn from the depicted Figures 9 and 10, focusing on 'amount vs. success' and 'amount vs. 3D_secured,' suggest that certain transaction amount ranges may correlate with the outcomes of success/failure and the presence/absence of 3D security. This observation indicates discernible trends within the data.

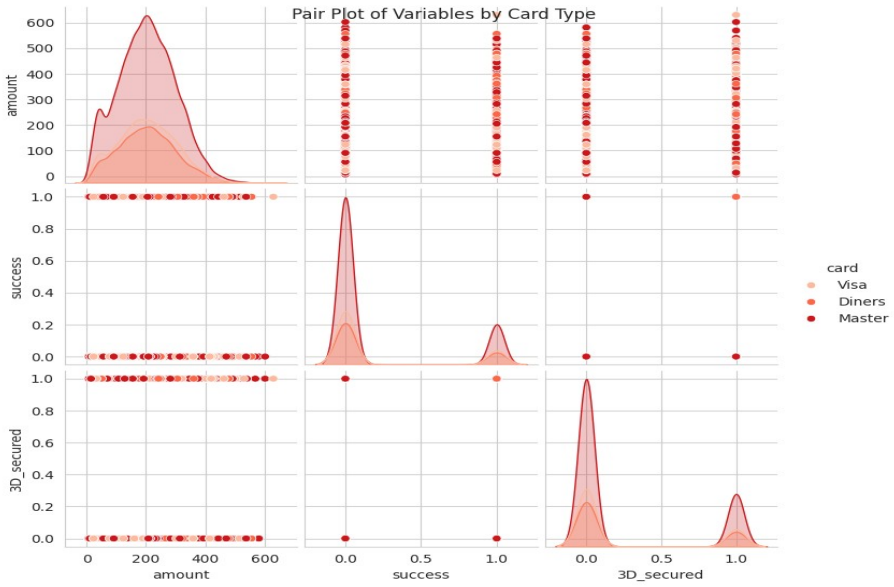


Figure 10: Pair Plot of Variables by Card

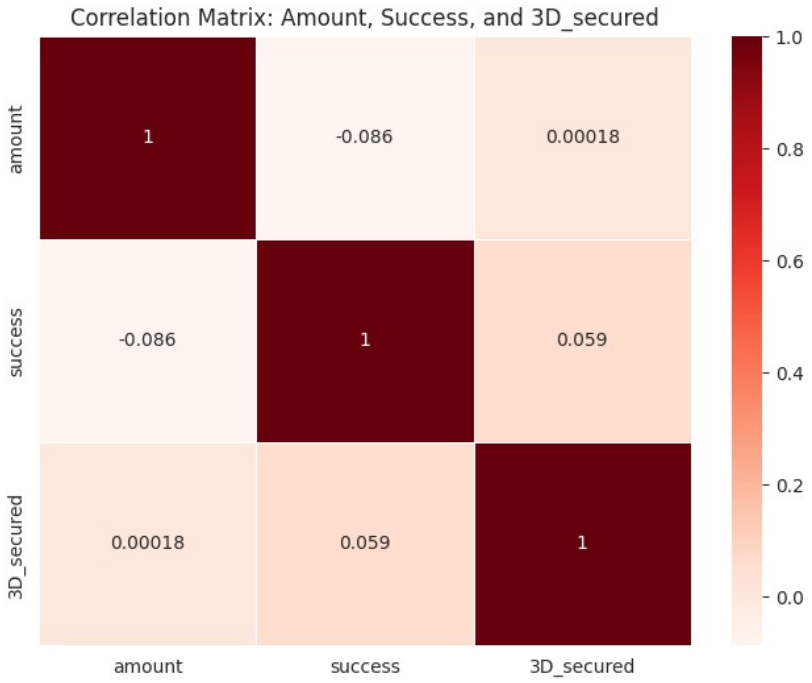


Figure 11: Correlation Matrix

The representation of 'Success vs. 3D_secured' with two distinct points emphasizes the categorical nature of both variables. They exhibit independent values for success (0 or 1) and 3D security (0 or 1). The modest correlations observed among these variables in Figure 11 suggest that there is not a significant linear relationship between transaction amount, transaction success, and the existence of 3D security. These results underscore the importance of considering additional factors and engaging in a more thorough analysis to grasp the intricate interactions within the data.

The bar plot illustrating success rates by card type stands as a crucial visualization, offering a lucid and succinct portrayal of success rates associated with various card categories. This provides valuable insights for informed decision-making, translating complex information into actionable knowledge.

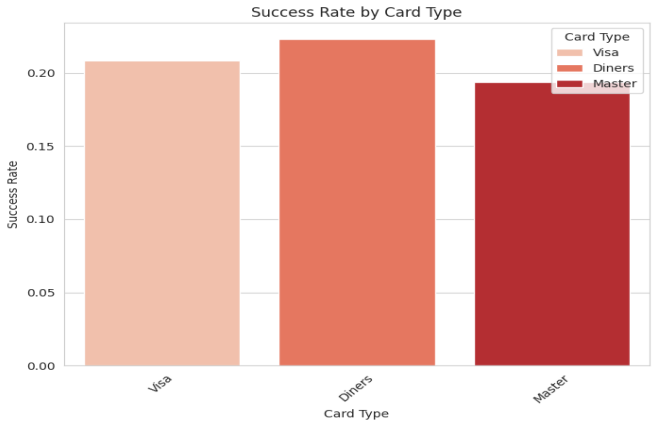


Figure 12: Success Rate by Card Type

The analysis of the bar graph in Figure 12 reveals that success rates exhibit variation across different card types, with "Dinner" showing the highest success rate, succeeded by "Visa" and then "Master." The observed hierarchy in success rates among these card types provides substantial insights, enabling informed decision-making, process optimization, and the potential enhancement of overall transaction success for specific card categories.

Examining transaction data through time series analysis proves to be a potent method for understanding the temporal dynamics of transactions. This approach facilitates the discovery of patterns and empowers decision-makers with valuable insights derived from past trends.

Figure 13: Daily Transaction Volume

The graph depicting daily transaction volume in Figure 13 exhibits a zig-zag pattern, suggesting the absence of a clear trend or seasonality in the daily transaction volume over time. Additionally, the boxplot illustrating transaction amounts by country serves as a crucial visualization, presenting the distribution of transaction amounts across different countries.

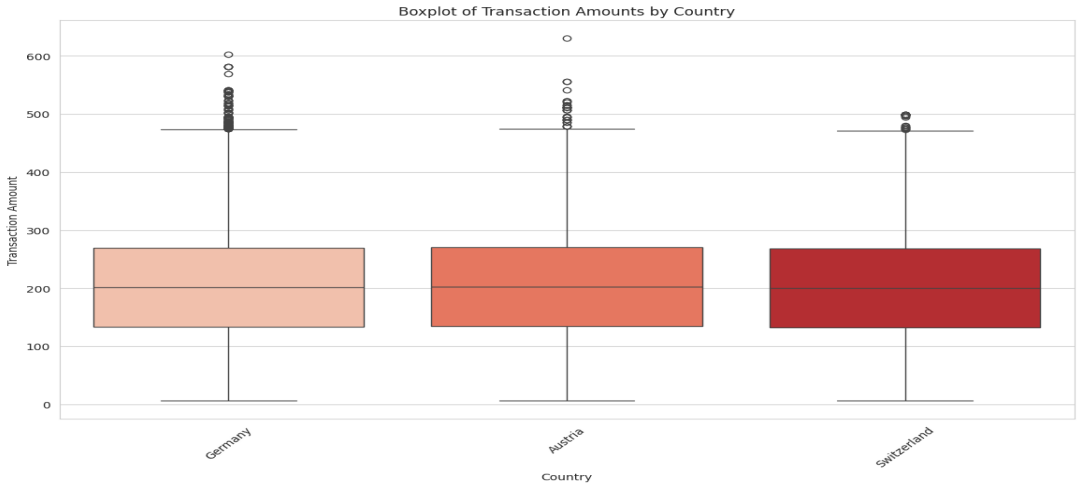


Figure 14: Boxplot of Transaction Amounts by country

The identical medians displayed in Figure 14 indicate similar transaction amounts in Germany, Austria, and Switzerland. However, distinctions in the upper whisker points suggest potential variability, with higher points in Germany indicating either larger transaction amounts or a broader range.

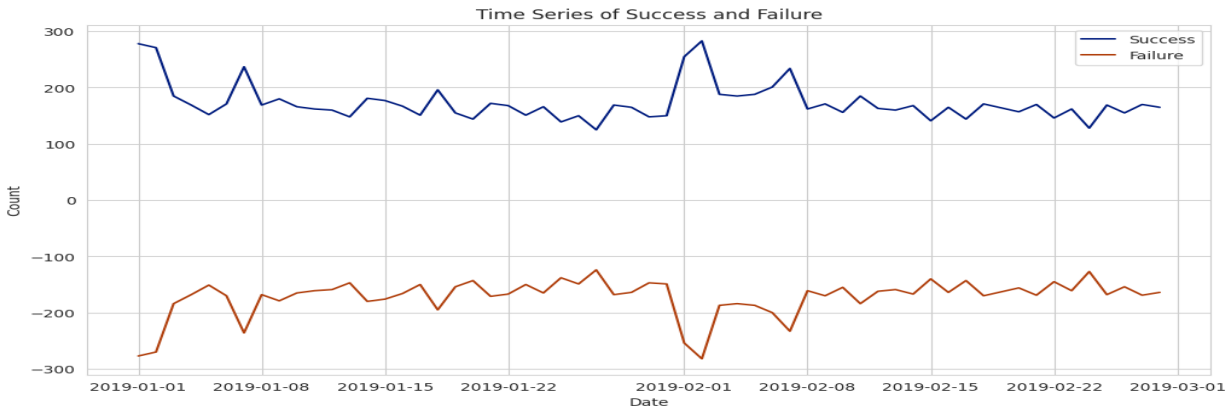


Figure 15*: Time Series of Success and Failure*

The time series plot depicting success and failure proves to be an effective instrument for assessing performance, identifying trends, and supporting decision-making processes. It provides a dynamic representation of the system's behaviour over time, contributing to the continuous refinement of processes and outcomes.

The time series plot depicting success and failure might suggest swift and frequent shifts between success and failure values. This tendency could be associated with the inherent nature of our data, where occurrences of success and failure are frequent and potentially brief. Additionally, comprehending the temporal patterns of success and failure in our dataset necessitates the use of a visualization illustrating the success rate over time. This serves as a valuable exploratory data analysis tool, offering actionable insights for decision-making and process enhancement. [Tukey,1977] Dasu and Johnson,2003].

The success rate exhibits regular fluctuations across the dates in the dataset. Each 'o' marker signifies the mean success rate for a specific day, and the lines connecting these markers portray the evolving trend over time.

In summary, an initial examination of the given dataset unveiled a transaction amount distribution skewed to the right, diverse success rates, and categorical variables denoting 'success' and '3D_secured.' A more detailed analysis, utilizing tools like boxplots, pair plots, and correlation matrices, brought to light intricate correlations that prompted a deeper investigation. Visual representations, such as success rates categorized by card type and time series analysis, yielded valuable insights into the variability of transactions and the dynamic patterns of success rates.

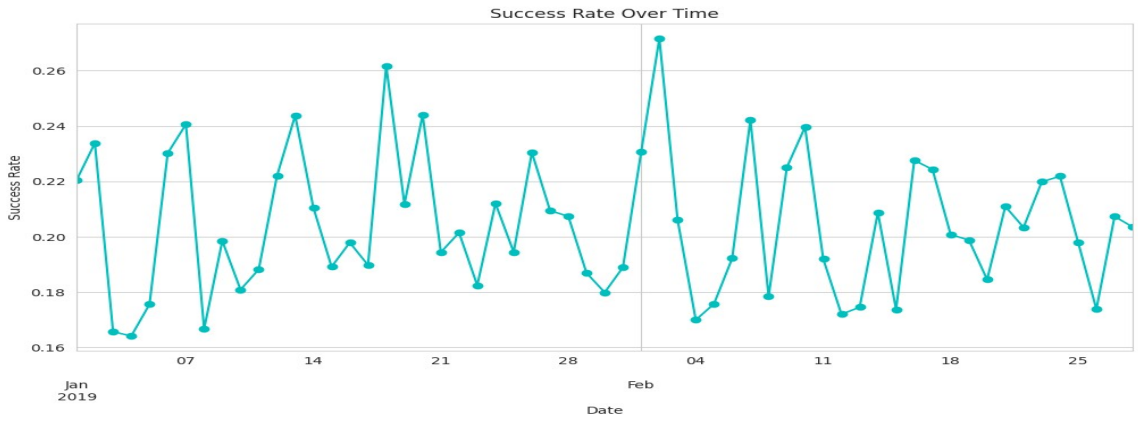


Figure 16: Success Rate Over Time

The phase of data preparation involves conducting operations that lead to the creation of the ultimate dataset, which will be fed into the modeling tool(s), from the initial raw data. These data preparation tasks are iterative and can be executed in a flexible sequence. Tasks such as selecting tables, records, and attributes, along with transforming and cleansing data for modeling tools, constitute integral components of this phase (CRISP-DM 1.0. Step-by-Step Data Mining Guide - PDF Free Download, n.d.).

5 Data Collection

In this scenario, it is imperative to ensure that the data acquired spans the required period and encompasses all essential variables. Beyond the provided data, there is a need to compile information on payment service providers, including their respective service fees.

Table 2: Transaction Fee

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

Associated the service fees of Payment Service Providers (PSPs) with the dataset under the column identifier "transaction_fee."

```
Associated the service fees of Payment Service Providers (PSPs) with the dataset under the
column identifier 'transaction_fee'

                 tmsp  country  amount  success         PSP  3D_secured  \
0 2019-01-01 00:01:11  Germany      89        0     UK_Card           0
1 2019-01-01 00:01:17  Germany      89        1     UK_Card           0
2 2019-01-01 00:02:49  Germany     238        0     UK_Card           1
3 2019-01-01 00:03:13  Germany     238        1     UK_Card           1
4 2019-01-01 00:04:33  Austria     124        0  Simplecard           0

     card  transaction_fee
0    Visa              1.0
1    Visa              3.0
2  Diners              1.0
3  Diners              3.0
4  Diners              0.5
```

Figure 17: PSP with the dataset under the column identifier 'transaction_fee'

5.1 Data Cleaning

Data cleaning, alternatively referred to as data preparation, constitutes a crucial phase in the data science pipeline. This process involves identifying and rectifying or eliminating errors, discrepancies, and inaccuracies in the data to enhance its quality and utility. [Huang et al., 1999] The significance of data cleaning arises from the fact that raw data is often characterized by noise, inaccuracy, and inconsistency, which can diminish the precision and dependability of insights derived from it ("ML | Overview of Data Cleaning," 2018).

Data earmarked for analysis using data mining techniques may display a range of imperfections, including incompleteness, noise, and inconsistencies. Incomplete data, lacking attribute values or essential attributes, often results from factors like unavailability or the perceived insignificance of information during input. In large, real-world databases, inconsistencies, such as discrepancies in department codes, are common. Reasons for incomplete data range from information unavailability to oversights or equipment malfunctions during recording. Noisy data, characterized by incorrect attribute values, may stem from faulty data collection instruments, human or computer errors during entry, or problems in data transmission. Cleaning routines are crucial to tackle these challenges, encompassing tasks like filling in missing values, smoothing noisy data, managing outliers, and resolving inconsistencies. Data cleaning is vital for enhancing the reliability of mining procedures and preventing confusion arising from unclean data. While many mining routines include procedures for dealing with incomplete or noisy data, it is recommended to incorporate dedicated pre-processing steps with data cleaning routines for optimal results. (Jiawei Han, Micheline Kamber, and Jian Pei, 3rd Edition, 2011)

Quantitative data, comprising integers or floating-point numbers measuring quantities of interest, may be presented as simple number sets or complex multidimensional arrays, occasionally recorded over time as time series. These data are typically associated with a unit of measure, necessitating uniformity across the dataset for meaningful analyses; however, challenges may arise, especially with unit conversion for volatile units like currencies. Data cleaning techniques in the realm of quantitative data often rely on statistical methods for outlier detection. These methods aim to pinpoint readings that deviate significantly from the expected values based on the rest of the dataset. In recent years, this domain has extended into the field of data mining, which has emerged, in part, to devise efficient statistical methods suitable for exceptionally large datasets. (Mann, 2005)

5.2 Data transformation

Data transformation involves altering data from one format to another to enhance its suitability for analysis. Various techniques, such as normalization, scaling, or encoding, may be employed in this process (source: "ML | Overview of Data Cleaning," 2018).
Specifically, within the dataset, a column requires conversion and separation into two independent columns: 'day_of_week' and 'minute_of_day.' The 'day_of_week' column will range from 0 to 6, denoting the day of the week for each corresponding timestamp in the 'tmsp' column, where Monday is denoted by 0, Tuesday by 1, and so forth until Sunday represented by 6. The 'minute_of_day' column, for each timestamp in the 'tmsp' column, will indicate the total number of minutes elapsed since midnight. Additionally, a new column named "payment_attempts" is introduced to the dataset to track the number of attempts, as detailed in the section on detecting and handling duplicate records.

Dataset Post Data Transformation

	tmsp	country	amount	success	PSP	3D_secured	card	transaction_fee	day_of_week	minute_of_day	payment_attempts
0	2019-01-10 03:49:12	Austria	6	0	Moneycard	0	Diners	2.0	3	229	1
1	2019-01-10 03:49:37	Austria	6	0	Simplecard	0	Diners	0.5	3	229	2
2	2019-02-08 05:02:33	Austria	6	0	UK_Card	0	Diners	1.0	4	302	1
3	2019-02-08 05:02:37	Austria	6	0	UK_Card	0	Diners	1.0	4	302	2
4	2019-02-08 05:02:39	Austria	6	0	Simplecard	0	Diners	0.5	4	302	3

Figure 18: Post Data Transformation of Dataset

In the preprocessing of data, one can employ feature selection methods to achieve efficient data reduction, a practice beneficial for identifying accurate data models (Jović et al., 2015). The purpose of this approach is to simplify the model and enhance its interpretability by eliminating unnecessary or redundant features.

The selected features for analysis in the provided dataset encompass 'country,' 'amount,' 'success,' 'PSP,' '3D_secured,' 'card,' 'transaction_fee,' 'day_of_week,' 'minute_of_day,' and 'payment_attempts.' Notably, the 'tmsp' column, representing timestamps, was deliberately excluded from the investigation as it was not utilized by the chosen features. This feature selection technique aims to emphasize essential features while excluding timestamp information, which has been modified and divided into two distinct columns: 'day_of_week' and 'minute_of_day.'

Updated dataset after feature selection

	country	amount	success	PSP	3D_secured	card	transaction_fee	day_of_week	minute_of_day	payment_attempts
0	Austria	6	0	Moneycard	0	Diners	2.0	3	229	1
1	Austria	6	0	Simplecard	0	Diners	0.5	3	229	2
2	Austria	6	0	UK_Card	0	Diners	1.0	4	302	1
3	Austria	6	0	UK_Card	0	Diners	1.0	4	302	2
4	Austria	6	0	Simplecard	0	Diners	0.5	4	302	3

Figure 19: Updated dataset after feature selection

5.3 Handling Imbalanced Data

In numerous real-world scenarios, datasets often exhibit imbalance, where certain classes have significantly more instances than others. This imbalance can significantly impact the performance of classifiers. Learning algorithms that do not consider class imbalance are susceptible to being dominated by the majority class, thereby neglecting the minority class (Liu et al., 2009). In our dataset context, addressing class imbalance involves employing 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, the Synthetic Minority Over-sampling Technique (SMOTE) is applied to generate synthetic samples for the minority class using imbalanced-learn. These combined strategies contribute to achieving a more balanced dataset, enhancing the

performance of machine learning models, particularly in situations with pronounced class imbalance.

5.4 Data Splitting

A crucial aspect of machine learning involves developing computational models with robust prediction and generalization capabilities. In supervised learning, a computational model is trained to predict the outputs of an unknown target function (Reitermanova, 2010). Adhering to the commonly used 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 unseen data, we split the dataset accordingly.

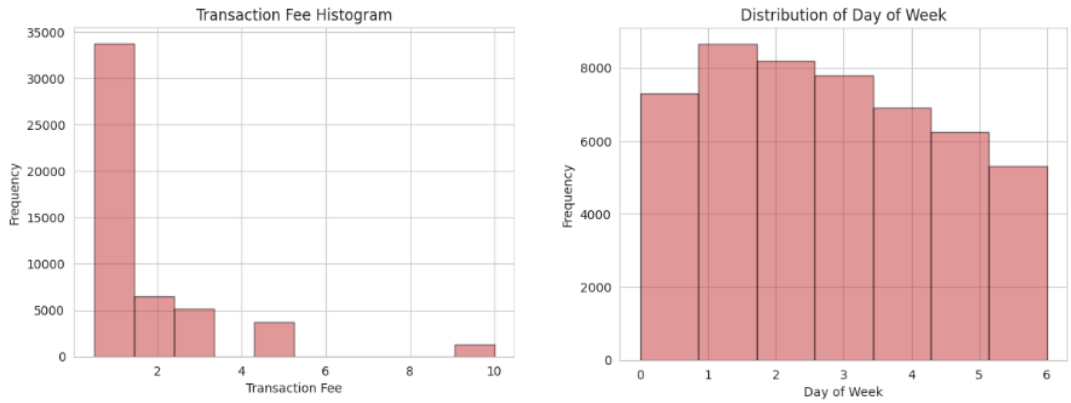
5.5 Comprehensive Evaluation of Data Quality

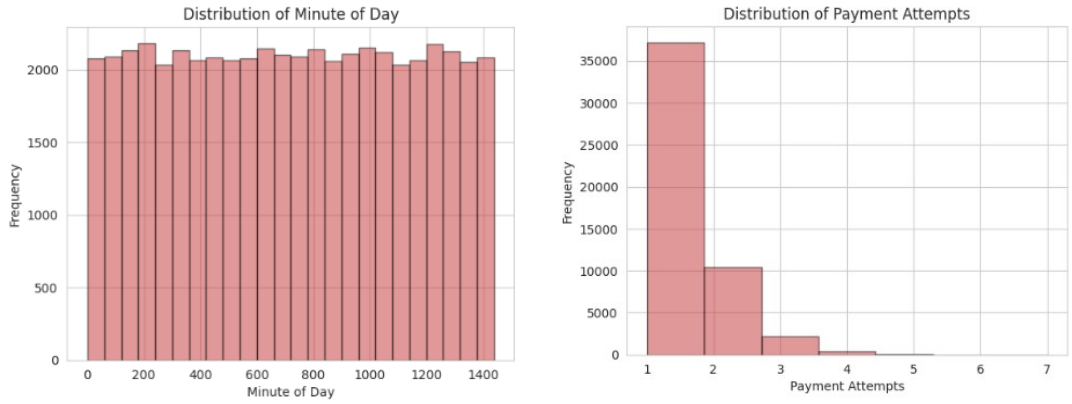
A thorough data quality assessment during the data preparation phase involves various procedures to identify inconsistencies, duplications, anomalies, and assess variable distributions before utilizing the refined data for model construction.

5.6 Summary of Recently Introduced Data Columns

The updated dataset, resulting from feature selection as outlined in Table 5, includes the columns 'transaction_fee,' 'day_of_week,' 'minute_of_day,' and 'payment_attempts.' Furthermore, when scrutinizing attributes like 'day_of_week,' 'minute_of_day,' and 'payment_attempts,' the emphasis shifts towards gaining insights from the distribution rather than relying solely on traditional statistical summaries. The utilization of histograms, as indicated by the code output and depicted, reveals a notable pattern:

Output 3: Histogram pattern





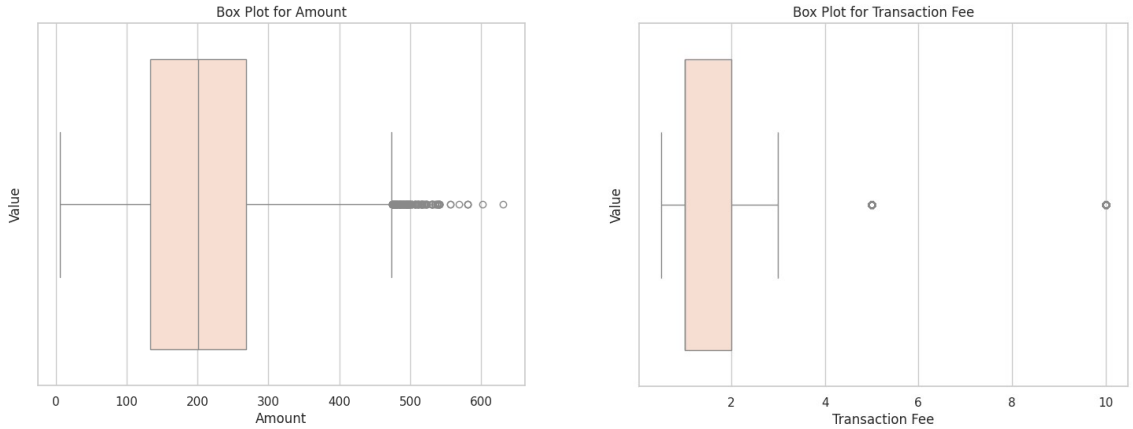
transactions peak on Tuesday, followed by Wednesday, Thursday, Monday, Friday, Saturday, and Sunday. Moreover, delving into the histogram illustrating the distribution of minutes throughout the day unveils valuable insights into transaction frequency across a 24-hour period. Notably, the dataset contains 37,227 entries with a single 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.

5.7 Verify for Inconsistencies

Reviewing the distinct values within categorical columns (country, PSP, card) to identify any unexpected or inconsistent entries. According to the output generated by the code, the unique values seem appropriate, and no unexpected or inconsistent values are evident in these specific columns.

Analysing the numeric variables, the box plot unveils distinct patterns. For "amount," a skew towards higher values is evident, portraying a prevalence of substantial transactions, as depicted by the extended upper whisker. Conversely, "transaction_fee" displays fewer instances of elevated values beyond the upper whisker, suggesting a tendency towards smaller transaction costs.

Output 4: Boxplot to display inconsistencies



Moving on to assess categorical variables, such as "country," "PSP," and "card," ensuring a balanced distribution of categories is crucial. Fortunately, the examination reveals no outliers

or uncommon categories, indicating a well-distributed dataset. This absence of irregularities is reassuring, indicating a reasonable spread of categories within our dataset.

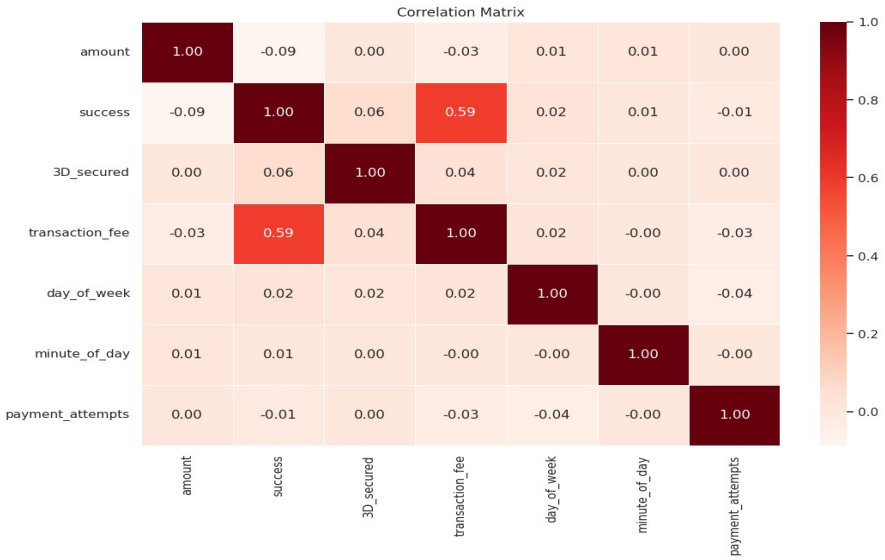


Figure 20: Correlation Matrix

6. Developing Model

Now, we embark on the exciting journey of model crafting, kicking off with the establishment of a foundational model. Our path then diverges into the realm of linear model techniques, embracing Logistic Regression and Decision Tree classification and Random Forest Classification alongside the captivating avenues of XGBoost classification. The overarching mission? To unearth the quintessential model.

Our litmus test in this odyssey lies in the realm of model evaluation, a realm where we partition our dataset into two distinct realms: one devoted to the nurturing of our model's intellect and the other, a sanctuary for the revelation of its prowess (in accordance with Phase 4 of the CRISP-DM Process Model). Herein lies the essence of our data division strategy, adhering fervently to the 80-10-10 principle: an allocation of 80% for training, 10% for the dance of validation and model scrutiny, and the remaining 10% for the unveiling of our model's finesse on uncharted terrain.

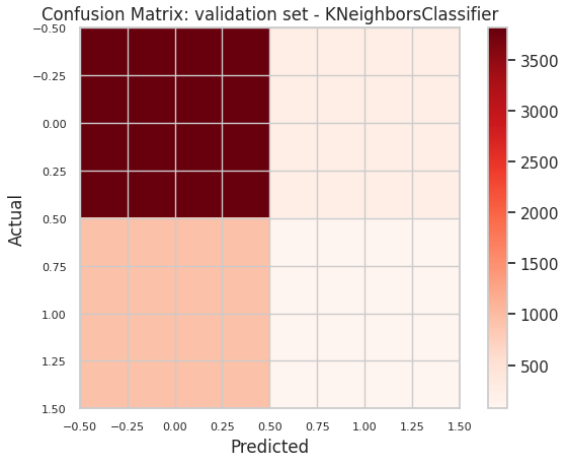
6.1 Baseline Model

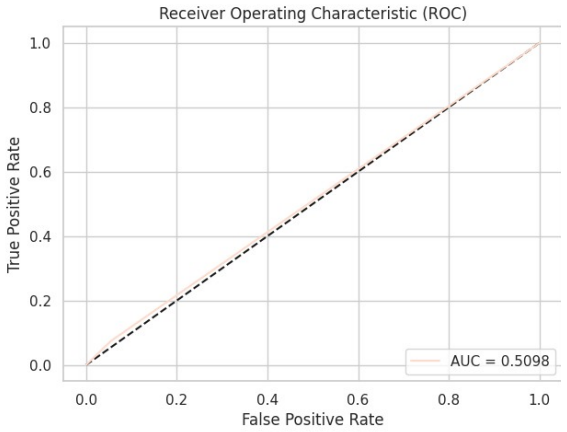
A baseline model serves as a humble yet crucial starting point, allowing us to gauge the potential impact and rationale behind venturing into more intricate methodologies (Conroy, 2023). Usually crafted before addressing any data imbalances, this initial model sets the stage for future explorations. In this regard, the K-Nearest Neighbors (KNN) technique emerges as a fitting candidate for such a foundational role. Renowned for its straightforwardness and accessibility, KNN stands as a stalwart choice, providing a solid platform from which to begin our analytical journey.

Output 5: Model Performance on both sets for Baseline model

Model Performance on Validation Set

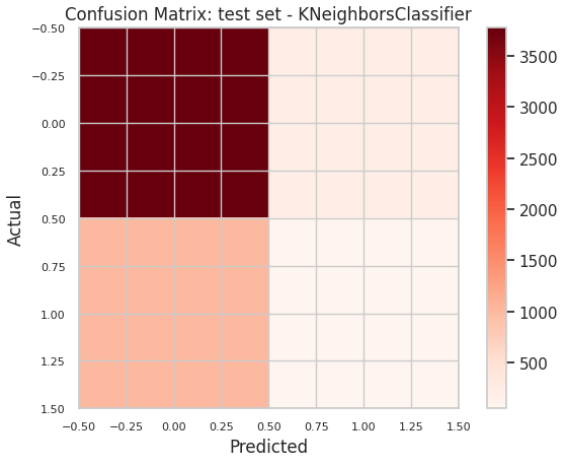
```
Accuracy: 0.7713
Precision: 0.2525
Recall: 0.0746
F1-Score: 0.1151
AUC-ROC: 0.5098
```

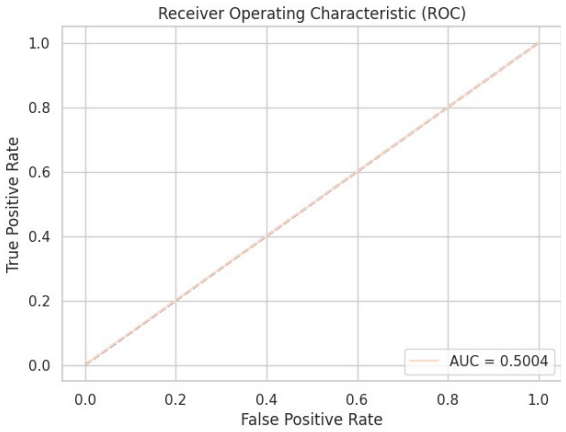




Model Performance on Test Set

```
Accuracy: 0.7590
Precision: 0.2137
Recall: 0.0526
F1-Score: 0.0844
AUC-ROC: 0.5004
```





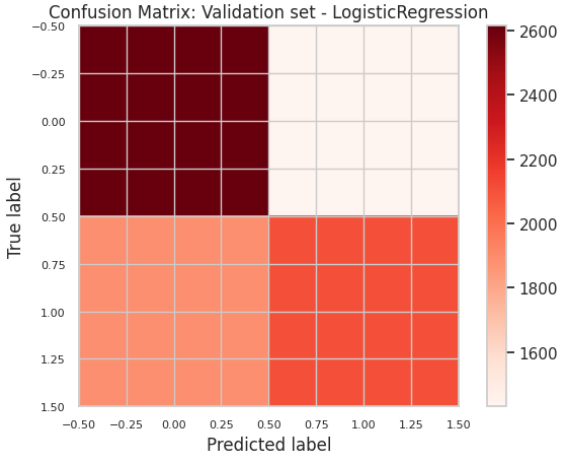
6.2 Model

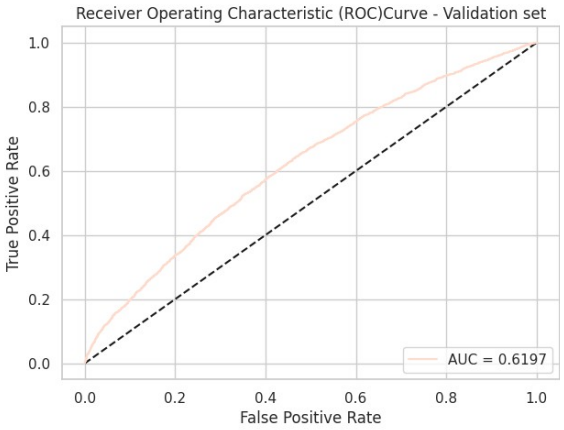
Model endeavours to forecast the likelihood of success, utilizing transaction-related attributes (excluding transaction_fee) as predictors, with success serving as the target variable. This model's primary aim is to compute success probabilities. To accomplish this, we employed a trio of modeling techniques: Logistic Regression for linear approaches, and Random Forest Classification for a non-linear method. Before crafting Model.

Output 6: Model Performance of both sets for different ML model

Model Performance on Validation Set

```
Model Performance on Validation set - LogisticRegression:


Classification Metrics:
Accuracy: 0.5870
Precision: 0.5951
Recall: 0.5269
F1-Score: 0.5589
AUC-ROC: 0.6197
```

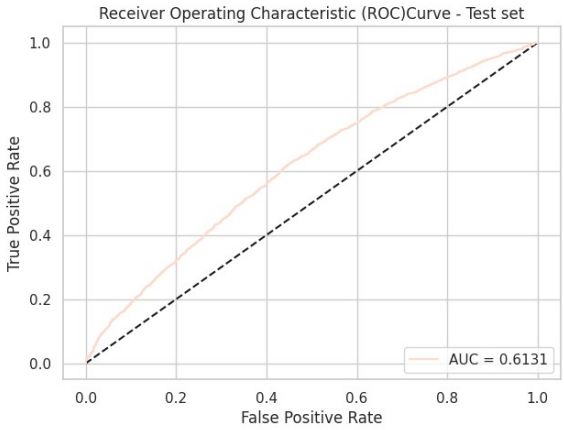




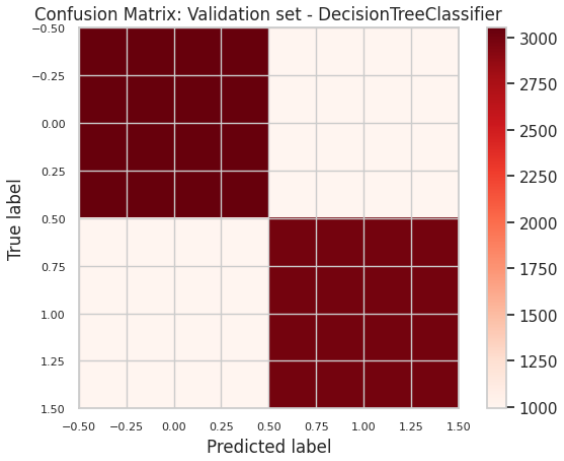

```
Model Performance on Validation set - DecisionTreeClassifier:


Classification Metrics:
Accuracy: 0.7522
Precision: 0.7510
Recall: 0.7497
F1-Score: 0.7503
AUC-ROC: 0.7522
```

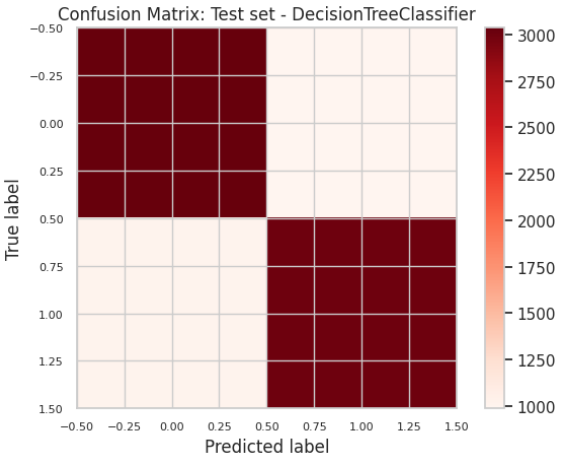
Model Performance on Test Set

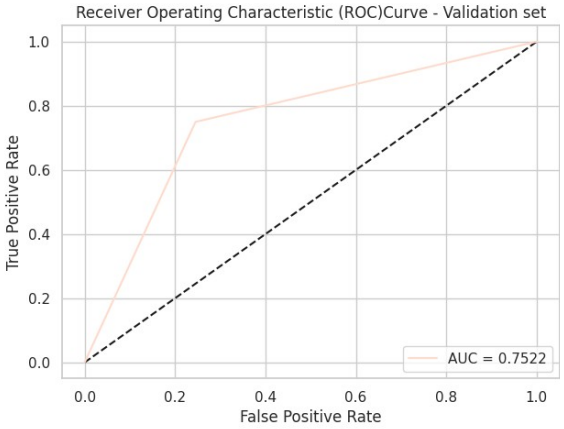
```
Model Performance on Test set - LogisticRegression:


Classification Metrics:
Accuracy: 0.5786
Precision: 0.5861
Recall: 0.5316
F1-Score: 0.5575
AUC-ROC: 0.6131
```

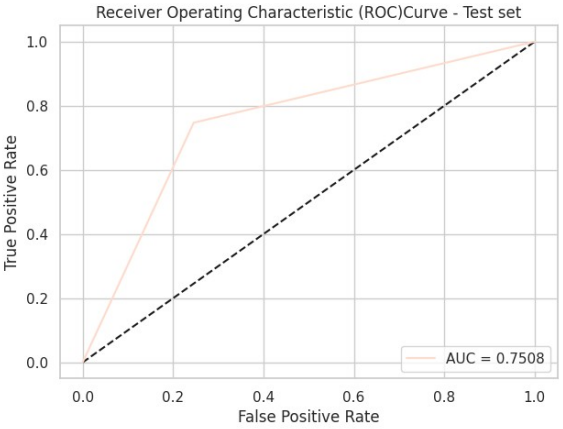



```
Model Performance on Test set - DecisionTreeClassifier:


Classification Metrics:
Accuracy: 0.7508
Precision: 0.7522
Recall: 0.7471
F1-Score: 0.7497
AUC-ROC: 0.7508
```



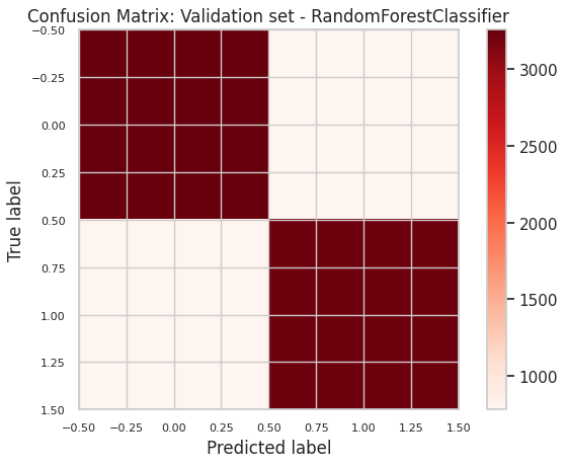


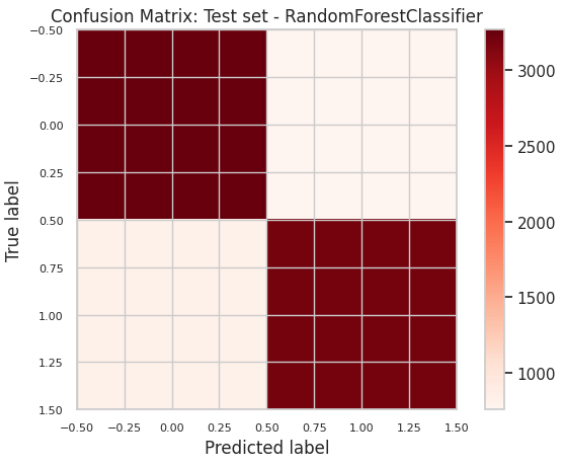


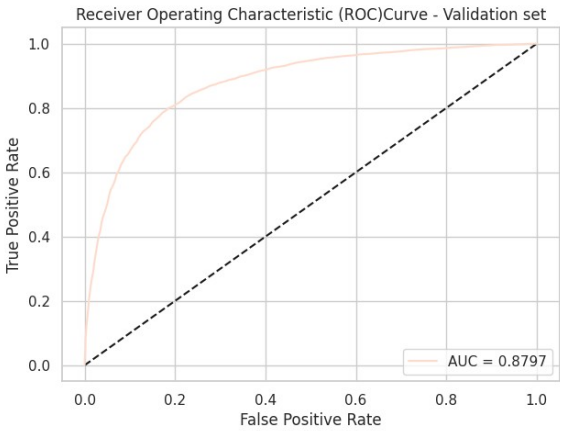


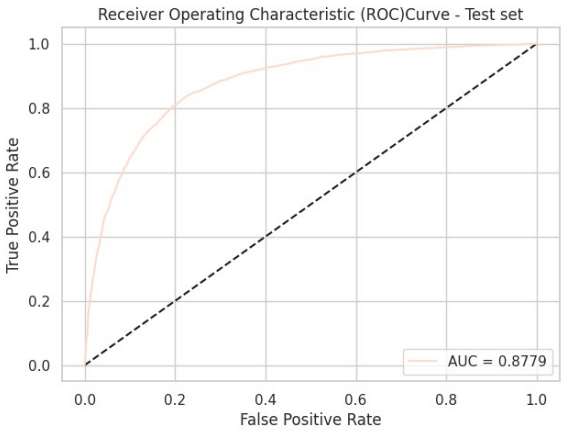

```
Model Performance on Validation set - RandomForestClassifier:


Classification Metrics:
Accuracy: 0.8053
Precision: 0.8034
Recall: 0.8048
F1-Score: 0.8041
AUC-ROC: 0.8797
```

```
Model Performance on Test set - RandomForestClassifier:


Classification Metrics:
Accuracy: 0.8034
Precision: 0.8083
Recall: 0.7950
F1-Score: 0.8016
AUC-ROC: 0.8779
```



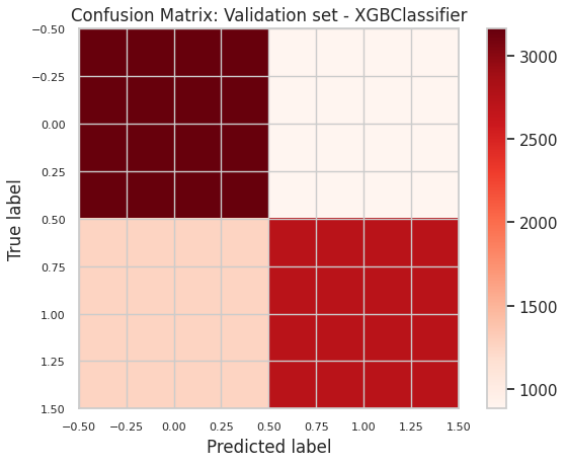


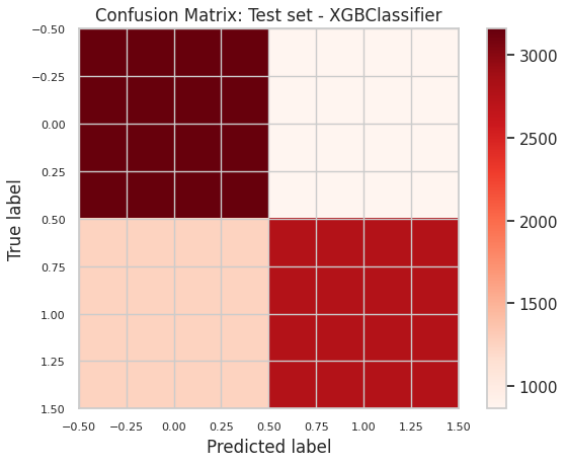


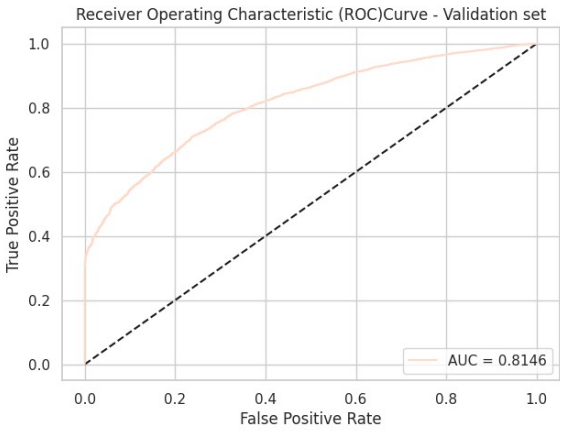


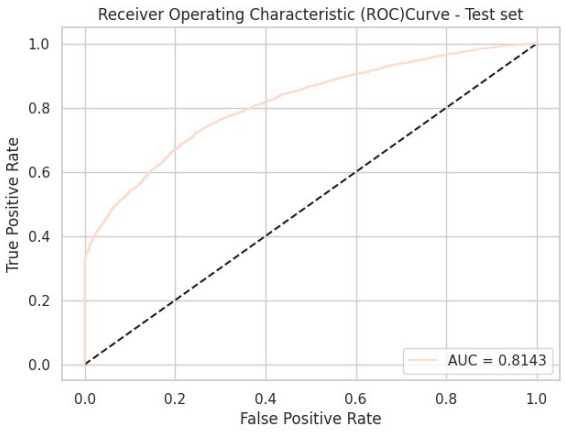

```
Model Performance on Validation set - XGBClassifier:


Classification Metrics:
Accuracy: 0.7331
Precision: 0.7558
Recall: 0.6833
F1-Score: 0.7177
AUC-ROC: 0.8146
```

```
Model Performance on Test set - XGBClassifier:


Classification Metrics:
Accuracy: 0.7365
Precision: 0.7616
Recall: 0.6876
F1-Score: 0.7227
AUC-ROC: 0.8143
```









6.3 Hyperparameter Tunning of Model

Based on the preceding findings, the Random Forest Classifier demonstrates superior performance compared to Logistic Regression, Decision Tree, XGBoost, and Linear

Regression across both validation and test sets. Notably, Decision Tree and XGBoost exhibit similar metrics. It is worth mentioning that the models exhibit slightly enhanced performance on the validation set, hinting at potential areas for further refinement. Addressing the concern of overfitting, we opt for the Random Forest Classifier due to its exceptional performance on both validation and test sets. To reinforce its reliability, we employ cross-validation and hyperparameter tuning. Cross-validation offers a more stable evaluation of model performance compared to a solitary train-test split, while hyperparameter tuning facilitates the identification of the most effective parameter settings for our model. By integrating these methodologies, we aim to cultivate a machine learning model that is not only more precise but also more dependable, enhancing its accuracy and trustworthiness (Singh, 2023).

Best Parameters found for Decision Tree set - DecisionTreeClassifier

```
Best Parameters found for Decision Tree set - DecisionTreeClassifier:
========================================
Best Parameters for Decision Tree set: {'criterion': 'gini', 'max_depth': 30,
'min_samples_leaf': 1, 'min_samples_split': 2}
Best Score for Decision Tree set: 0.7311287225501728
```

Figure 21: Best Parameters found for Decision Tree set

Best Parameters found for Random Forest set - RandomForestClassifier:

```
Best Parameters found for Random Forest set - RandomForestClassifier:
========================================
Best Parameters for Random Forest set: {'max_depth': 30, 'min_samples_leaf': 1,
'min_samples_split': 2, 'n_estimators': 200}
Best Score for Random Forest set: 0.7890373267439946
```

Figure 22: Best Parameters found for Random Forest set

Best Parameters found for XGBoost set - XGBClassifier:

```
Best Parameters found for XGBoost set - XGBClassifier:
========================================
Best Parameters for XGBoost set: {'colsample_bytree': 1.0, 'learning_rate': 0.1, 'max_depth':
5, 'n_estimators': 200, 'subsample': 0.8}
Best Score for XGBoost set: 0.7247515937091522
```

Figure 23: Best Parameters found for XGBoost set

Model Performance for various model:

Output 7: Model Performance of both sets for different ML model after Hyperparameter Tunning

Model Performance on Validation Set: Decision Tree

```
================== DecisionTreeClassifier ==================

Validation Set Evaluation:
Model Performance on Validation set - DecisionTreeClassifier:


Classification Metrics:
Accuracy: 0.7522
Precision: 0.7510
Recall: 0.7497
F1-Score: 0.7503
AUC-ROC: 0.7522
```

Model Performance on Test Set: Decision Tree

```
Test Set Evaluation:
Model Performance on Test set - DecisionTreeClassifier:


Classification Metrics:
Accuracy: 0.7508
Precision: 0.7522
Recall: 0.7471
F1-Score: 0.7497
AUC-ROC: 0.7508
```

Classification Report: Validation set - DecisionTreeClassifier

	precision	recall	f1-score	support
0	0.75	0.75	0.75	4045
1	0.75	0.75	0.75	3991
accuracy			0.75	8036
macro avg	0.75	0.75	0.75	8036
weighted avg	0.75	0.75	0.75	8036

Classification Report: Test set - DecisionTreeClassifier

	precision	recall	f1-score	support
0	0.75	0.75	0.75	4023
1	0.75	0.75	0.75	4014
accuracy			0.75	8037
macro avg	0.75	0.75	0.75	8037
weighted avg	0.75	0.75	0.75	8037

Model Performance on Validation Set: Random Forest

```
Validation Set Evaluation:
Model Performance on Validation set - RandomForestClassifier:


Classification Metrics:
Accuracy: 0.8053
Precision: 0.8034
Recall: 0.8048
F1-Score: 0.8041
AUC-ROC: 0.8797
```

Model Performance on Test Set: Random Forest

```
Test Set Evaluation:
Model Performance on Test set - RandomForestClassifier:


Classification Metrics:
Accuracy: 0.8034
Precision: 0.8083
Recall: 0.7950
F1-Score: 0.8016
AUC-ROC: 0.8779
```

Classification Report: Validation set - RandomForestClassifier

	precision	recall	f1-score	support
0	0.81	0.81	0.81	4045
1	0.80	0.80	0.80	3991
accuracy			0.81	8036
macro avg	0.81	0.81	0.81	8036
weighted avg	0.81	0.81	0.81	8036

Classification Report: Test set - RandomForestClassifier

	precision	recall	f1-score	support
0	0.80	0.81	0.81	4023
1	0.81	0.79	0.80	4014
accuracy			0.80	8037
macro avg	0.80	0.80	0.80	8037
weighted avg	0.80	0.80	0.80	8037

Model Performance on Validation Set: XGBoost

```
Validation Set Evaluation:
Model Performance on Validation set - XGBClassifier:


Classification Metrics:
Accuracy: 0.7331
Precision: 0.7558
Recall: 0.6833
F1-Score: 0.7177
AUC-ROC: 0.8146
```

Model Performance on Test Set: XGBoost

```
Test Set Evaluation:
Model Performance on Test set - XGBClassifier:


Classification Metrics:
Accuracy: 0.7365
Precision: 0.7616
Recall: 0.6876
F1-Score: 0.7227
AUC-ROC: 0.8143
```

Classification Report: Validation set - XGBClassifier

	precision	recall	f1-score	support
0	0.71	0.78	0.75	4045
1	0.76	0.68	0.72	3991
accuracy			0.73	8036
macro avg	0.74	0.73	0.73	8036
weighted avg	0.74	0.73	0.73	8036

Classification Report: Test set - XGBClassifier

	precision	recall	f1-score	support
0	0.72	0.79	0.75	4023
1	0.76	0.69	0.72	4014
accuracy			0.74	8037
macro avg	0.74	0.74	0.74	8037
weighted avg	0.74	0.74	0.74	8037

As depicted in Output 5, fine-tuning the hyperparameters had a profound impact on enhancing the metrics of the Random Forest Classifier, notably elevating the F1-Score, Recall, and AUC-ROC on both validation and test sets. This refined model exhibited consistent performance across sets, indicating robust generalization capabilities. Notably,

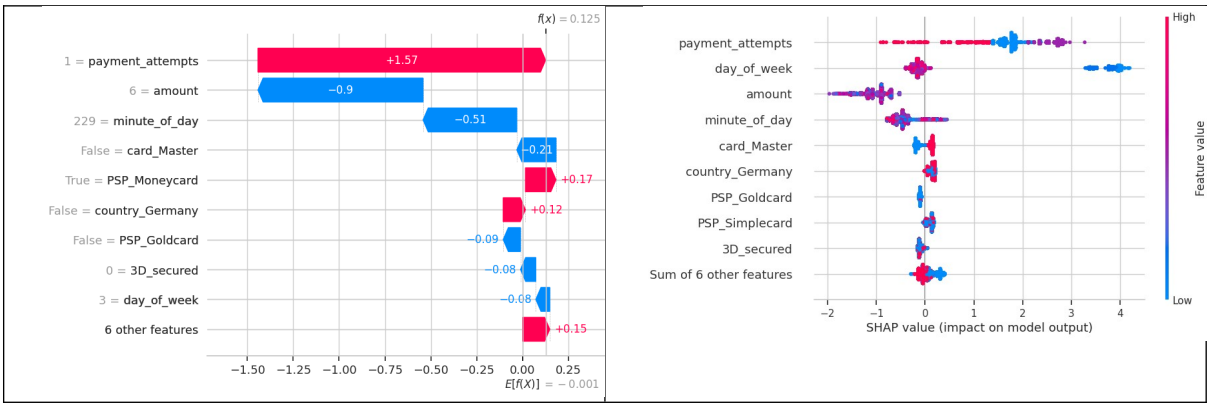
the revised confusion matrices highlighted a reduction in false positives and negatives, thereby enhancing overall predictive accuracy. The meticulous selection of optimal hyperparameters, including no maximum depth, one sample per leaf, two samples per split, and 200 trees, played a pivotal role in achieving the observed performance enhancements. After analysis all the result the best ML ops for our model is Random Forest algorithm.

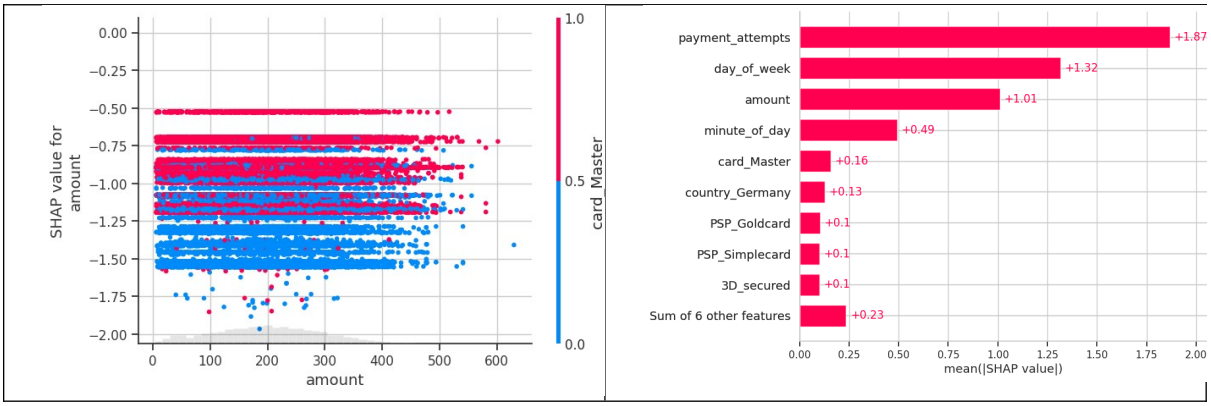
Moreover, following the meticulous process of hyperparameter tuning and cross-validation, we proceeded to utilize the best-trained Random Forest Classifier model (best_rf_model) to predict success probabilities. These probabilities were then appended to the original dataset, enriching it with the invaluable insight of success likelihood. Lastly, Table 7 presents the modified DataFrame, encompassing both the original features and the predicted success probabilities, culminating in a comprehensive depiction of the dataset's augmented composition.

6.4 Evaluation

Discussing the importance of individual features is crucial for aligning predictive models with business objectives. By elucidating the contribution of each feature, transparency is fostered, and stakeholder trust is bolstered. This approach bridges the gap between predictions and business goals, enhancing understanding of the model's functionality. In our scenario, comprehending why Model make specific predictions holds equal importance to prediction accuracy. However, achieving optimal accuracy for large-scale datasets often proves challenging due to the complexity of models that may elude experts' comprehension. In response, various strategies have emerged to aid users in interpreting predictions generated by complex models. However, the relationship between these strategies and their comparative efficacy remains unclear. To address this ambiguity, we propose SHAP (SHapley Additive exPlanations), which provides an importance score for each feature in a particular prediction. Notable features of SHAP include the introduction of a new class of additive feature significance measures and theoretical studies validating the existence of a single solution within this class, possessing desirable qualities (Lundberg & Lee, 2017).

Output 8: Model Performance and feature display after SHAP





The SHAP analysis underscores the importance of various variables in influencing Model 1 predictions. Notably, "day_of_week" emerges as a critical factor, with a SHAP value exceeding 0.07, indicating its significant impact. Additionally, variables such as "amount," "minute_of_day," and "3D_secured" also contribute significantly to the predictions. Payment-related features like "card_Visa," "card_Diners," and "PSP_Moneycard" underscore the relevance of transaction-specific attributes, while geographic factors such as "country_Germany," "country_Switzerland," and "country_Austria" highlight the regional significance in shaping Model 1 predictions. Overall, these findings underscore the pivotal role of temporal, transactional, and spatial factors in shaping the predictive outcomes.

Figure 24: Prediction's Explanation with a Force Plot

6.5 Error Analysis

Conducting a sophisticated error analysis involves delving deeper into model performance to uncover its strengths and weaknesses. This method is instrumental in identifying patterns and gaining insights into areas where the model can be improved or refined. Here are some steps for sophisticated error analysis that we can undertake (Karani, 2022).

6.5.1 Confusion Matrix Analysis

Model demonstrates superior performance in predicting success probability, achieving F1 scores of 0.805 (validation) and 0.804 (test). The accuracy and precision on both validation and test sets hover around 80.7% and 80.9%, respectively. Although the recall rate slightly dips on the test set to 79.9%, Model exhibits commendable performance in forecasting transaction success.

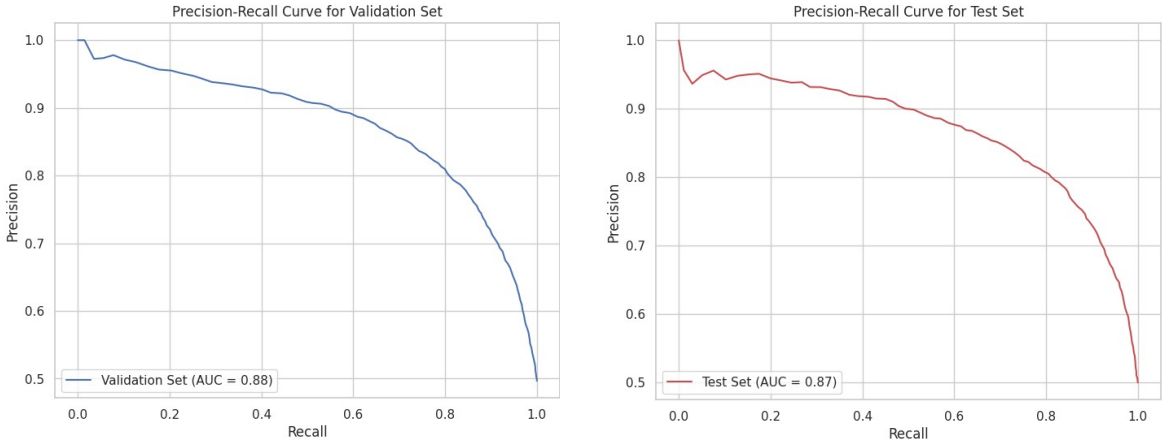
6.5.2 Precision-Recall Trade-off

The confusion matrices for Model on both validation and test sets reveal a notable number of True Positives, indicating proficient identification of instances in each class. Moreover, the model exhibits minimal occurrences of False Positives and False Negatives across most classes, signalling high efficacy in classifying instances. Overall, the results indicate that both models excel in accurately categorizing cases within the datasets.

The Precision-Recall curve analysis on both the validation and test sets highlighted an impressive AUC-PR score of 0.87 each, indicating a promising equilibrium between Precision and Recall. To fine-tune Model 1, adjustments were made to the threshold to attain the desired Precision and Recall values, thus enabling a suitable trade-off.

Subsequently, the Random Forest Model 1 underwent evaluation on the validation and test sets initially with the default threshold, followed by dynamically adjusting the decision threshold to achieve the desired balance between Precision and Recall. Utilizing the Precision-Recall curve facilitated the identification of the optimal threshold to strike the desired accuracy and recall equilibrium.

Output 9: Precision-Recall Curve

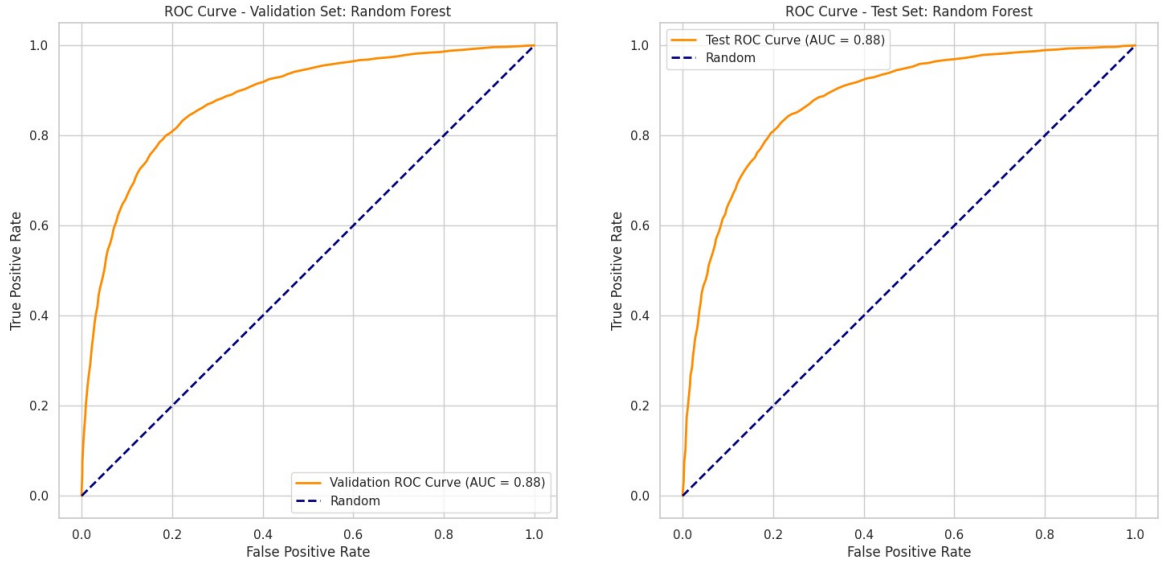



```
Desired Threshold: 0.4500

==================== RandomForestClassifier ==========

Validation Set Evaluation: RandomForestClassifier
Accuracy: 0.8053
Precision: 0.8053
Recall: 0.8053
F1 Score: 0.8053
ROC AUC: 0.8052
Validation Set Evaluation: RandomForestClassifier
```

```
==================== RandomForestClassifier ==========

Test Set Evaluation: RandomForestClassifier
Accuracy: 0.8034
Precision: 0.8035
Recall: 0.8034
F1 Score: 0.8034
ROC AUC: 0.8034
Test Set Evaluation: RandomForestClassifier
```

Adjusting the threshold for ROC Curve analysis resulted in an AUC of 0.88. Surprisingly, the values for both validation and test sets remained consistent even after the threshold adjustment.

Output 10: ROC Curve



This suggests that the model's ability to differentiate between positive and negative instances, as indicated by the AUC, was unaffected by the threshold modification. The AUC serves as a metric for the model's overall discriminatory power and is less influenced by changes in the decision threshold.

6.6 Cross Validation

Cross-validation (CV) is a method used to assess and validate the performance of machine learning models. It is commonly utilized in practical machine learning scenarios to aid in the comparison and selection of the most suitable model for a specific predictive task (Lyashenko, 2022). Thus, cross-validation was utilized to assess the consistency of Models 1 and 2 across multiple folds. Significant variation observed during this process may suggest that Models 1 and 2 are sensitive to variations in the training data.

Table 11: Cross-validation results

Cross Validation results for XGBoost	Cross Validation results for Random Forest
Results for Fold 1: Mean Test Score: 0.735437290329559 Standard Deviation Test Score: 0.003425995624542122 Params: {'learning_rate': 0.1, 'max_depth': 10, 'min_child_weight': 2, 'n_estimators': 50} Results for Fold 2: Mean Test Score: 0.7458120405231317 Standard Deviation Test Score: 0.0044130245866453545 Params: {'learning_rate': 0.1, 'max_depth': 10, 'min_child_weight': 2, 'n_estimators': 100} Results for Fold 3: Mean Test Score: 0.7524381444793538 Standard Deviation Test Score: 0.0024227086410330547 Params: {'learning_rate': 0.1, 'max_depth': 10, 'min_child_weight': 2, 'n_estimators': 200} Results for Fold 4: Mean Test Score: 0.7339907128431806 Standard Deviation Test Score: 0.004564923155756975 Params: {'learning_rate': 0.1, 'max_depth': 10, 'min_child_weight': 5, 'n_estimators': 50} Results for Fold 5: Mean Test Score: 0.7420478421864358 Standard Deviation Test Score: 0.004134851181198634 Params: {'learning_rate': 0.1, 'max_depth': 10, 'min_child_weight': 5, 'n_estimators': 100}	Results for Fold 1: Mean Test Score: 0.6801263536401752 Standard Deviation Test Score: 0.004165829300033321 Params: {'max_depth': 10, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 50} Results for Fold 2: Mean Test Score: 0.6816350826717402 Standard Deviation Test Score: 0.0025292434157193947 Params: {'max_depth': 10, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100 Results for Fold 3: Mean Test Score: 0.6827394486529785 Standard Deviation Test Score: 0.0024241055046142087 Params: {'max_depth': 10, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200 Results for Fold 4: Mean Test Score: 0.6814328279644963 Standard Deviation Test Score: 0.0033573024504930343 Params: {'max_depth': 10, 'min_samples_leaf': 1, 'min_samples_split': 5, 'n_estimators': 50} Results for Fold 5: Mean Test Score: 0.682070584999771 Standard Deviation Test Score: 0.0024579291548511356 Params: {'max_depth': 10, 'min_samples_leaf': 1, 'min_samples_split': 5, 'n_estimators': 100

Initially, after thorough testing, the model's optimal hyperparameters are determined, highlighting consistent performance across both validation and test sets. The use of GridSearchCV to evaluate the model indicates minimal fluctuations in mean test scores and

small standard deviations, underscoring its stability and ability to generalize well. This resilience to variations in training data subsets suggests the model's robustness. Overall, the model demonstrates cross-validation stability, hinting at its potential to perform well on new data.

Furthermore, assessing the sub-model's stability involves scrutinizing individual cross-validation scores and calculating standard deviations. With specific hyperparameters in place, the sub-model demonstrates stable and high-performance attributes across various cross-validation folds. The average F1 weighted score maintains its steadiness, with consistent scores across individual folds, highlighting its reliability and attractiveness as a choice.

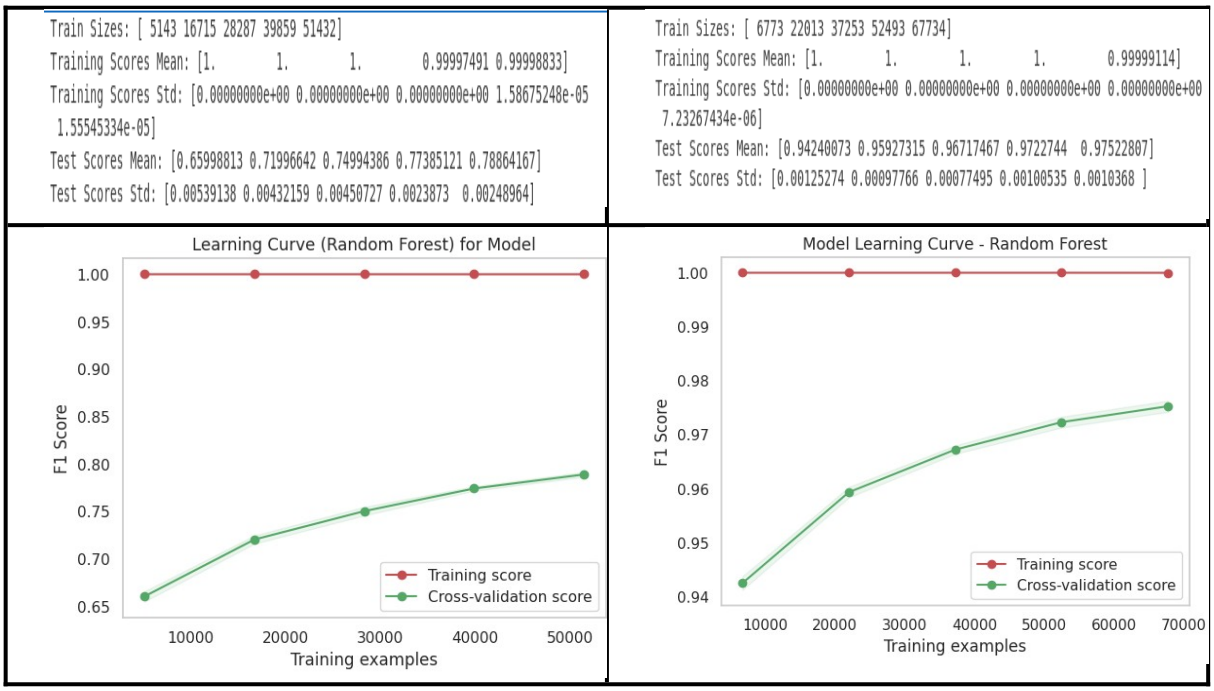
```
Cross-Validation Scores with Best Model of Random Forest Classification:
[0.97689045 0.97493756 0.97557847 0.97333734 0.97516384]
Average F1 Weighted Score: 0.9752
Standard Deviation of F1 Weighted Scores: 0.0011
Fold 1: 0.9769
Fold 2: 0.9749
Fold 3: 0.9756
Fold 4: 0.9733
Fold 5: 0.9752
```

Figure 25: Cross-validation score with best model of Random Forest

6.7 Overfitting

In supervised machine learning, overfitting is a common challenge that prevents models from effectively generalizing to both observed training data and unseen testing data. To assess the possibility of overfitting, techniques such as learning curves, as suggested by Ying (2019), are often employed.

Table 12: Overfitting Results



In our context, the learning curve illustrates the F1 score for both the training and cross-validation sets. As depicted in Figure, the training score for Model 1 indicates the model's ability to accurately predict labels within the training data, while the cross-validation score curve demonstrates how well the model generalizes to unknown data. The upward trend of the curve suggests that the model is progressively learning from the data.

7. Business Understanding

At this stage, it is crucial to ensure that the evaluation criteria align with the initial understanding and objectives of the business. In the context of a retail company's credit card routing project, a two-model approach is adopted to automate credit card routing, aiming to enhance success rates while minimizing transaction expenses. The project plan outlines requirements, assumptions, and risk management strategies, emphasizing communication and collaboration with online payment and IT teams. Evaluation criteria are closely tied to business objectives, ensuring that the models address relevant concerns. SHAP analysis offers deeper insights into the significance of specific features in Models. Prioritizing interpretability, results cater to both technical and non-technical stakeholders. Error analysis, including confusion matrices, precision-recall trade-offs, ROC curves, and inspection of misclassified samples, is conducted for Model. Cross-validation stability analysis confirms the models' robust generalization. These techniques underscore the connection to broader business goals, highlighting the strategic value of the project to the retail organization.

In the end, our model analysis the count of predicted and best PSP available for the customer. This also help in future analysis of card use by customers.

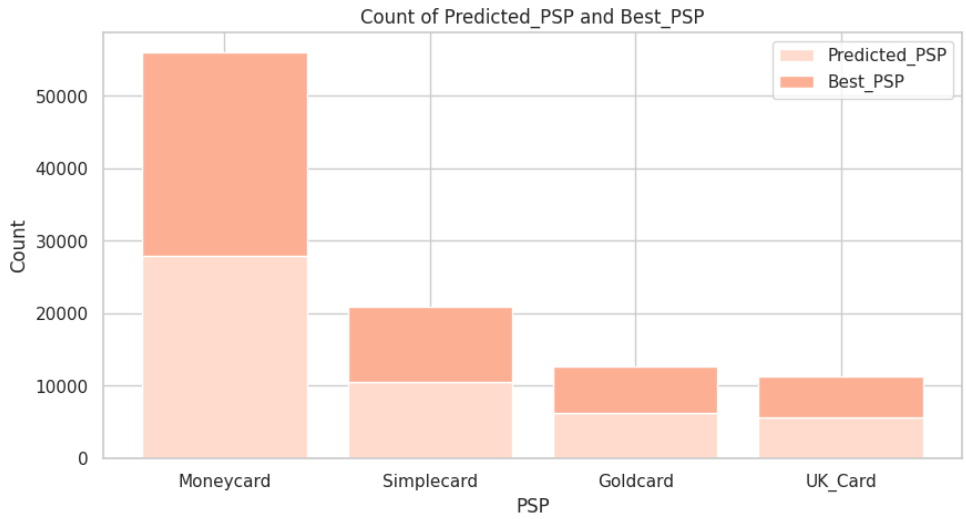


Figure 25: Count of predicted PSP and Best PSP

7.1 Deployment Phase

The deployment phase in the CRISP-DM methodology entails implementing the solutions generated during the modeling phase into practical use. This final stage of the data mining process involves integrating these solutions into a production environment to generate value for businesses. In our case, since Model 2 was developed to predict optimal PSP, Output 13

displays the predicted PSP for each existing transaction under the column "Predicted_PSP," targeting the lowest transaction cost and highest success probabilities. Additionally, the code is structured to handle any ties in predicted PSPs by selecting the most cost-effective PSP based on transaction costs, identified under the column "Best_PSP." However, it is worth noting that no instances of ties were encountered in our specific scenario.

Moreover, the deployment phase within the CRISP-DM framework holds paramount significance in ensuring the success of the entire data mining process. All the efforts invested in understanding the business, comprehending the data, preparing it, modeling, and evaluating will be futile without an efficient deployment strategy. The solutions derived from the modeling phase hold value only when they are put into action, which underscores the importance of the deployment step. To address this, a deployment proposal tailored for the business production environment is outlined (Heymann et al., 2022).

7.1.1 Technical Setup

Commence by establishing the necessary technological infrastructure to host the predictive model, ensuring a smooth deployment process. This involves configuring hardware and software settings, as well as verifying compatibility with machine learning libraries. Serialize the models (Models 1 and 2) for easy loading and document dependencies to maintain consistency between development and production environments.

7.1.2 Data Pipeline Integration

Leverage the machine learning pipeline to automate the process of building prediction models, thereby increasing reusability and improving optimization. Integrate predictive models into the data processing pipeline, establish connections, implement real-time data handling, and incorporate appropriate pre-processing procedures for enhanced efficiency. This modular approach ensures that models are versatile and efficient to create ("Data Pipeline Machine Learning Architecture," 2023).

7.1.3 GUI Development

Craft a user-friendly dashboard using Gradio to facilitate easy transaction input and visualize routing suggestions in machine learning. Ensure secure application access through user authentication, enhancing 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.).

7.1.4 Key Feature Integration

Integrate real-time decision-making by incorporating predictions from Model 1 and Model 2, dynamically selecting payment service providers based on success probabilities and transaction fees. Implement alerts or notifications to address crucial occurrences such as model failures promptly, guaranteeing proactive system monitoring and maintenance.

Implement continuous model monitoring, involving tracking, analyzing, and assessing machine learning models in real-world scenarios. This ensures long-term accuracy and dependability by monitoring various data and model metrics over time ("Model Monitoring for

ML in Production," n.d.). Develop a monitoring dashboard to measure essential performance parameters such as success rates and fees. Establish an automatic method for updating models in real-time when new data becomes available, enabling adaptability to changing transaction patterns through regular retraining. This permits efficient monitoring and seamless modifications to Models 1 and 2 for optimal performance.

8 Conclusion

The successful completion of the predictive modeling project for credit card routing signifies a remarkable accomplishment, highlighting a thorough and methodical approach from project inception to implementation. The proposed structure of the Git repository serves as a well-structured framework facilitating improved project management, code organization, and collaboration. It ensures transparency, consistency, and accessibility for users through effective directory organization, a "requirements.txt" file, and comprehensive documentation in the form of a "README.md" file. The investigative approach employed in Chapter 3 follows the CRISP-DM model, indicating a thorough data mining process. The initial phase of business knowledge establishes clear project objectives using a strategic two-model approach for predicting success and selecting PSPs. Subsequent phases delve into meticulous data collection, quality assessment, and feature exploration, enhancing project insights through visualization and analytics.

The thorough data assessment encompasses detailed examination of transaction specifics, success metrics, security attributes, card variations, and temporal patterns, enriching the understanding of the credit card routing project. This comprehension is further refined by identifying needs, assumptions, limitations, risks, and contingencies, forming a robust foundation for project planning and risk management aligned with broader organizational objectives. The data preparation phase is critical, involving rigorous tasks such as data collection, cleaning, transformation, and feature selection. After augmenting the dataset with information on payment service providers and their fees, meticulous cleaning and transformation ensure data integrity. Feature selection prioritizes crucial variables, while addressing data imbalances with advanced techniques like SMOTE, laying the groundwork for subsequent modeling steps.

Sophisticated error analysis techniques confirm the robust performance of both models in accurately categorizing occurrences, indicating their effectiveness in forecasting success probability, and recommending best PSPs. The deployment strategy outlines a comprehensive plan to ensure smooth integration into the business environment, aiming for enhanced efficiency in credit card routing, increased success rates, and reduced transaction costs.

In summary, the project's success is attributed to its meticulous approach, effective project management, and seamless integration of models into the retail organization's operations. The models not only fulfil business aims but also prove interpretability and reliable performance, significantly enhancing credit card routing processes and aligning with overarching goals of financial advancement and customer satisfaction.

9 Future Work

Prospective endeavours may involve exploring innovative methodologies to address data imbalances within Model 1, delving into alternative algorithms for Model 2 to enhance classification accuracy, conducting thorough examinations of misclassified cases, and instituting ongoing enhancements to the models through real-time monitoring and feedback mechanisms. Additionally, investigating how external variables influence model efficacy and enhancing the system's scalability and adaptability to evolving business needs would contribute to the continuous enhancement of the credit card routing system.

Reference

- Basic Git Concepts—Version Control with Git, 2nd Edition [Book]
- 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. CRISP-DM The New Blueprint for Data Mining Colin Shearer (Fall 2000)
- Build a GUI for Your Machine Learning Models | HackerNoon
- Conroy, M. (2023, May 2). CRISP-DM, Phase 4: Model Building. Data Lab Notes.
- Correlation Matrix—An overview | ScienceDirect Topics
- CRISP DM Step 1—Understanding About the Business | PGBS
- CRISP-DM 1.0. Step-by-step data mining guide—PDF Free Download
- Data Pipeline Machine Learning Architecture | StreamSets
- 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.
- 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.
- Karani, D. (2022, July 21). Deep Dive into Error Analysis and Model Debugging in Machine Learning (and Deep Learning) | Neptune.Ai
- 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, fromhttps://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, fromhttps://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-tun-ing-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 Inter-national 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 De-tailed 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.
- Python version: Python 3.11.6
- Operating System: Ubuntu 23.10
- Tested with Windows 11 with Python 3.12

```
# Importing pandas for data manipulation and analysis
import pandas as pd

# Importing numpy for numerical computing
import numpy as np

# Importing io for input/output operations
import io

# Importing classification models for building predictive models
from sklearn.tree import DecisionTreeClassifier # Decision Tree
Classifier
from sklearn.ensemble import RandomForestClassifier # Random Forest
Classifier
from sklearn.linear_model import LogisticRegression # Logistic
Regression Classifier
from sklearn.neighbors import KNeighborsClassifier # K-Nearest
Neighbors Classifier
from xgboost import XGBClassifier # XGBoost Classifier
import xgboost as xgb

# Importing metrics for model evaluation
from sklearn.metrics import (accuracy_score, precision_score,
recall_score,
f1_score, roc_auc_score, confusion_matrix,
classification_report, r2_score, mean_absolute_error,
mean_squared_error, roc_curve, auc, precision_recall_curve)

# Importing model selection and hyperparameter tuning utilities
from sklearn.model_selection import (GridSearchCV, train_test_split,
learning_curve,
StratifiedKFold, cross_val_score)

# Importing preprocessing utilities for feature scaling
from sklearn.preprocessing import StandardScaler, LabelBinarizer,
OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.calibration import label_binarize
# Importing visualization libraries
```

```
import matplotlib.pyplot as plt # Matplotlib for basic plotting
import seaborn as sns # Seaborn for enhanced visualization
import matplotlib.lines as mlines # For drawing lines in plots
import matplotlib.patches as patches # For drawing shapes in plots

# Importing library for handling imbalanced datasets
from imblearn.over_sampling import SMOTE # Synthetic Minority Over-
sampling Technique

# Importing SHAP library for model interpretability
import shap # SHapley Additive exPlanations


# Read the Excel file into a DataFrame
df = pd.read_excel(r"/home/samyak/Python_Projects/Case _Study_
Model_Engineering/Datasets/PSP_Jan_Feb_2019.xlsx")

# Display the DataFrame
print(df.head())
print(df.columns)
missing_values= df.isnull().sum()
missing_data =pd.DataFrame({
                            'Missing Values': missing_values,
                            #'Percentage Missing Value':
                            Percentage_missing_value
                            })
print(f"The dataframe has {len(missing_values)} rows with missing
values.")
data_type=df.dtypes
print(f"Data Type Information:\n{data_type}")
#Print the number of rows in the DataFrame
print("Number of rows in ideal data set : ",len(df))

for column in df.columns:
      unique_values= df[column].unique
      print(f"Column: {column}\nUnique Values: {unique_values}\n")
      # Perform mathematical operations on a specific column of data.
      if np.issubdtype(df[column].dtype,np.number): # Check if the
      datatype is numeric.
            mean = df[column].mean() # Calculate the mean of the
            numerical column.
            std = df[column].std() # Calculate the standard deviation
            of the numerical column.
            std = df[column].std() # Calculate the standard deviation
            of the numerical column.
            std_dev = df[column].std() # Calculate the standard
            deviation of the numerical column.

#Print the number of column in the Data Frame
```

```
print("Number of column in ideal data set : ",len(column))
# Displaying the information about the DataFrame, including data type
and memory usage
# of each column.
print("Data Frame Information")
print("-----------------------")
print(f"Number of rows: {len(df)}")
print(f"Number of columns: {len(df.columns)}")
for col in df.columns:
      print(f"\nColumn Name: {col}")
      print(f"Data Type: {type(df[col].values[0])}")

print(df.head())
#Dropint hr "Unnamed: 0" column form the Data Frame
df =df.drop('Unnamed: 0', axis=1)
# Print the remaining columns after dropping 'Unnamed: 0'
print(df.head())


print(df['amount'].describe().to_frame().T)
print(df['success'].describe().to_frame().T)
print(df['PSP'].describe().to_frame().T)
print(df['3D_secured'].describe().to_frame().T)
print(df['card'].describe().to_frame().T)
print(df['country'].describe().to_frame().T)

plt.figure(figsize=(10,6))
sns.set_style("whitegrid")
sns.histplot(df['amount'], kde=True, color='red') # Change the color of
the histogram
kde_line = mlines.Line2D([], [], color='blue', linewidth=2,
label='KDE')
plt.legend(handles=[kde_line])
plt.title('Distribution of Transaction Amount')
plt.xlabel('Transaction Amount')
plt.ylabel('Frequency')
plt.gca().get_lines()[-1].set_color('blue') # Change the color of the
KDE line
plt.show()

def plot_distribution(column, title, xlabel, ylabel):
      sns.set_style("whitegrid")

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

```
        # Create a countplot for the column with hue and legend
        parameters
        ax = sns.countplot(x=column, hue=column, data=df, palette='Reds',
        legend=False)
        # Create a grid
        plt.grid(True)
        # Set plot title and axis labels
        plt.title(title)
        plt.xlabel(xlabel)
        plt.ylabel(ylabel)
        # Add count labels to the bars
        for p in ax.patches:
        height = p.get_height()
        ax.text(p.get_x() + p.get_width() / 2.,
        height + 3,
        '{:1.0f}'.format(height),
        ha="center")
        # Show the plot distribution
        plt.show()
# Call the function for different columns
plot_distribution('success', 'Distribution of Success', 'Success',
'Count')
plot_distribution('3D_secured', 'Distribution of 3D_secured',
'3D_secured', 'Count')
plot_distribution('country', 'Distribution of Country', 'Country',
'Count')
plot_distribution('PSP', 'Distribution of PSP', 'PSP', 'Count')
plot_distribution('card', 'Distribution of Card', 'Card', 'Count')

# Create a boxplot with 'success' on the x-axis and 'amount' on the y-
axis
sns.boxplot(x='success', y='amount', data=df, palette='Reds',
hue='success')

# 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()

# Set the default color palette for seaborn to 'Dark'
sns.set_palette('dark')

# Create a pair plot for the selected variables
pair_plot = sns.pairplot(df[['amount', 'success', '3D_secured']])

# Set the overall title for the pair plot
```

```
pair_plot.fig.suptitle('Pair Plot of Variables')

# Show the plot
plt.show()
# Create a pair plot for the 'amount', 'success', and '3D_secured'
variables
# with hue based on the 'card' variable
sns.pairplot(df, hue='card', vars=['amount', 'success', '3D_secured'],
palette='Reds')

# 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='Reds', 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
fig, ax = plt.subplots(figsize=(15, 6))

# Use a bar plot with 'card' on the x-axis, 'success' on the y-axis,
and no confidence interval (errorbar=None)
barplot = sns.barplot(x='card', y='success', hue='card', data=df,
dodge=True, palette='Reds', ax=ax, legend=False)
# Set plot title and axis labels
plt.title('Success Rate by Card Type')
plt.xlabel('Card Type')
plt.ylabel('Success Rate')
# Rotate x-axis labels to prevent overlapping
plt.xticks(rotation=45)
# Add labels to the bars
for p in ax.patches:
      height = p.get_height()
      ax.text(p.get_x() + p.get_width() / 2.,
      height + 0.01, f'{height:.2f}', # Format the label as needed
      ha="center")

# 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), cmap='PuOr')

# 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, palette='Reds',
hue='country')
# 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='-', color='color'[0])
# 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()

def process_transaction_fees(df):
      # 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("Associated the service fees of Payment Service Providers
      (PSPs) with the dataset under the column identifier
      'transaction_fee' \n")
      print(df.head())
# Call the function with your DataFrame
process_transaction_fees(df)
# 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("Dataset Post Data Transformation \n")
print(df.head())

# Drop the 'tmsp' column
df.drop('tmsp', axis=1, inplace=True)
print("Updated dataset after feature selection")
print(df.head())


def visualize_data(df):
      # 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("\n", transposed_summary, "\n")

      # Create a histogram to visualize the distribution of transaction
      fees
      plt.hist(df['transaction_fee'], bins=10, edgecolor='black',
      color=sns.color_palette("coolwarm_r", 20)[0],
      alpha=0.5)

      # 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',
color=sns.color_palette("coolwarm_r", 20)[0], alpha=0.5)

# 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',
color=sns.color_palette("coolwarm_r", 20)[0],
alpha=0.5)

# 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',
color=sns.color_palette("coolwarm_r", 20)[0],
alpha=0.5)

# 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()

# Call the function with your DataFrame
visualize_data(df)

# 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)

sns.set(style="whitegrid")
# Set the palette to 'Reds'
palette='Reds'
sns.set_palette(palette)

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

# Identify non-numeric columns
non_numeric_columns = df.select_dtypes(exclude='number').columns

# Drop non-numeric columns before calculating correlation
correlation_matrix = df.drop(columns=non_numeric_columns).corr()

# Visualize the correlation matrix using a heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='Reds', fmt=".2f",
linewidths=.5)
plt.title('Correlation Matrix')
plt.show()


#Developing Model
# Create a copy of the DataFrame
df_1 = df.copy()
# Drop the 'transaction_fee' column from the copied DataFrame
#df_1 = df_1.drop(['PSP'], axis=1)
df_1 = df_1.drop('transaction_fee', axis=1)
# Print the modified DataFrame without the 'transaction_fee' column
print(df_1)
df_encoded_1 = pd.get_dummies(df_1, columns=['country', 'card'])
"""
Baseline Model
The baseline model is a simple linear regression model. It assumes that
the relationship between the dependent variable and independent
variables can be described by a
The baseline model is a simple linear regression model with no
interaction terms.
"""
def evaluate_model(model, X_scaled, y, set_name):
      """
      Evaluate the performance of a model on a given dataset.
      Parameters:
      model (object): A trained model object.
      X_scaled (array): The scaled features of the dataset.
      y (array): The target variable of the dataset.
      set_name (str): The name of the dataset (e.g., 'Training',
      'Testing').
      Returns:
      None
      """
      # Make predictions using the model
      y_pred = model.predict(X_scaled)
      # Get the predicted probabilities for each class
      y_pred_proba = model.predict_proba(X_scaled)[:, 1]
```

```
# Calculate the precision, recall, F1-score, and AUC-ROC
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)
# Calculate the accura
accuracy = accuracy_score(y, y_pred)
# Get the confusion matrix
conf_matrix = confusion_matrix(y, y_pred)
# Print the model performance metrics
print(f'\n{"="*20} {type(model).__name__} {"="*20}\n')
print(f'Model Performance on {set_name} set -
{type(model).__name__}:')
print('\n')
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(f'\n{"="*20}{"="*20}\n')
print(classification_report(y, y_pred))
# Calculate and print confusion matrix
cm = confusion_matrix(y, y_pred)
cm_df1 = pd.DataFrame(cm, index=model.classes_,
columns=model.classes_)
print(f'\n{"="*20}{"="*20}\n')
print("Confusion Matrix:")
print(cm_df1)
plt.figure()
plt.imshow(cm_df1, cmap='Reds')
plt.title(f'Confusion Matrix: {set_name} -
{type(model).__name__}')
plt.colorbar()
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.tick_params(axis='both', which='major', labelsize=8)
plt.tight_layout()
plt.show()
# Plot the ROC curve
fpr, tpr, _ = roc_curve(y, y_pred)
plt.figure(figsize=(7, 5))
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr, tpr, label='AUC = %0.4f' % roc_auc)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC)')
plt.legend(loc="lower right")
plt.show()
```

```
# 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)
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 A
# 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, random_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, random_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)

def calculate_classification_metrics(y_true, y_pred, y_pred_proba):
      """
      Calculate various classification metrics and display results.

      Parameters:
      y_true (array): True labels.
      y_pred (array): Predicted labels.
      y_pred_proba (array): Predicted probabilities for positive class.

      Returns:
      dict: Dictionary containing calculated metrics.
      """
      precision = precision_score(y_true, y_pred)
      recall = recall_score(y_true, y_pred)
      f1 = f1_score(y_true, y_pred)
      roc_auc = roc_auc_score(y_true, y_pred_proba)
      accuracy = accuracy_score(y_true, y_pred)

      metrics = {
      'Accuracy': accuracy,
      'Precision': precision,
      'Recall': recall,
      'F1-Score': f1,
      'AUC-ROC': roc_auc
      }

      print("Classification Metrics:")
      for metric, value in metrics.items():
      print(f"{metric}: {value:.4f}")
```

```
        return metrics

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

        Parameters:
        model (object): A trained model object.
        X_scaled (array): The scaled features of the dataset.
        y (array): The target variable of the dataset.
        set_name (str): The name of the dataset (e.g., 'Training',
        'Testing').

        Returns:
        None
        """
        print(f'Model Performance on {set_name} set -
        {type(model).__name__}:')
        print("\n")
        y_pred = model.predict(X_scaled)
        y_pred_proba = model.predict_proba(X_scaled)[:, 1]
        calculate_classification_metrics(y, y_pred, y_pred_proba)
        conf_matrix = confusion_matrix(y, y_pred)
        cm_df = pd.DataFrame(conf_matrix, index=model.classes_,
        columns=model.classes_)

        print(f'\n{"="*20}{"="*20}\n')
        print(f'Classification Report: {set_name} set -
        {type(model).__name__} \n')
        print(classification_report(y, y_pred))
        print(f'\n{"="*20}{"="*20}\n')
        print(f"Confusion Matrix: {set_name} set - {type(model).__name__}
        ")
        print(cm_df)
        print(f'\n{"="*20}{"="*20}\n')
        plt.figure()
        plt.imshow(cm_df, cmap='Reds')
        plt.title(f'Confusion Matrix: {set_name} set -
        {type(model).__name__}')
        plt.colorbar()
        plt.xlabel('Predicted label')
        plt.ylabel('True label')
        plt.tick_params(axis='both', which='major', labelsize=8)
        plt.tight_layout()
        plt.show()

        fpr, tpr, _ = roc_curve(y, y_pred_proba)
        plt.figure(figsize=(7, 5))
        plt.plot([0, 1], [0, 1], 'k--')
```

```
        plt.plot(fpr, tpr, label=f'AUC = {roc_auc_score(y,
        y_pred_proba):.4f}')
        plt.xlabel('False Positive Rate')
        plt.ylabel('True Positive Rate')
        plt.title(f'Receiver Operating Characteristic (ROC)Curve -
        {set_name} set')
        plt.legend(loc="lower right")
        plt.show()

# Model Training and Evaluation
def train_and_evaluate_model(model, X_train_scaled, y_train,
X_valid_scaled, y_valid, X_test_scaled, y_test):
        model.fit(X_train_scaled, y_train)

        print(f'\n{"="*20} {type(model).__name__} {"="*20}\n')

        print("Validation Set Evaluation:")
        evaluate_model(model, X_valid_scaled, y_valid, "Validation")

        print("Test Set Evaluation:")
        evaluate_model(model, X_test_scaled, y_test, "Test")

# Logistic Regression
log_reg_model = LogisticRegression(random_state=42)
train_and_evaluate_model(log_reg_model, X_train_scaled_1, y_train_1,
X_valid_scaled_1, y_valid_1, X_test_scaled_1, y_test_1)

# Decision Tree
dt_model = DecisionTreeClassifier(random_state=42)
train_and_evaluate_model(dt_model, X_train_scaled_1, y_train_1,
X_valid_scaled_1, y_valid_1, X_test_scaled_1, y_test_1)

# Random Forest
rf_model = RandomForestClassifier(random_state=42)
train_and_evaluate_model(rf_model, X_train_scaled_1, y_train_1,
X_valid_scaled_1, y_valid_1, X_test_scaled_1, y_test_1)

# XGBoost
xgb_model = xgb.XGBClassifier(random_state=42,
objective='multi:softmax', num_class=9)
train_and_evaluate_model(xgb_model, X_train_scaled_1, y_train_1,
X_valid_scaled_1, y_valid_1, X_test_scaled_1, y_test_1)

def optimize_and_evaluate_model(model, param_grid, X_train, y_train,
X_valid, y_valid, X_test, y_test, set_name):
        # Initialize GridSearchCV with the given model, hyperparameters,
        scoring metric, and cross-validation
        grid_search = GridSearchCV(estimator=model,
        param_grid=param_grid, scoring='accuracy', cv=5, n_jobs=-1)
```

```
        # Fit GridSearchCV to the training data
        grid_search.fit(X_train, y_train)

        # Print the best parameters and best score found by GridSearchCV
        print(f'Best Parameters found for {set_name} set -
        {type(model).__name__}:')
        print(f'{"="*20}{"="*20}')
        print(f"Best Parameters for {set_name} set:
        {grid_search.best_params_}")
        print(f"Best Score for {set_name} set:
        {grid_search.best_score_}")
        # Get the best model (i.e., the model with the best
        hyperparameters)
        best_model = grid_search.best_estimator_

# Perform grid search for Decision Tree model
# Define the grid of hyperparameters for the Decision Tree model
param_grid_dt = {
                 'criterion': ['gini', 'entropy'],
                 'max_depth': [10, 20, 30],
                 'min_samples_split': [2, 5, 10],
                 'min_samples_leaf': [1, 2, 4]
                 }
# Instantiate the Decision Tree model with a random state for
reproducibility
dt_model_A = DecisionTreeClassifier(random_state=42)
# Optimize and evaluate the Decision Tree model using the defined grid
of hyperparameters
optimize_and_evaluate_model(dt_model_A, param_grid_dt,
X_train_scaled_1, y_train_1, X_valid_scaled_1, y_valid_1,
X_test_scaled_1, y_test_1, set_name="Decision Tree")
train_and_evaluate_model(dt_model_A, X_train_scaled_1, y_train_1,
X_valid_scaled_1, y_valid_1, X_test_scaled_1, y_test_1)

# Perform grid search for Random Forest model
# Define the grid of hyperparameters for the Random Forest model
param_grid_rf = {
                 'n_estimators': [50, 100, 200],
                 'max_depth': [10, 20, 30],
                 'min_samples_split': [2, 5, 10],
                 'min_samples_leaf': [1, 2, 4]
                 }

# Instantiate the Random Forest model with a random state for
reproducibility
best_rf_model = RandomForestClassifier(random_state=42)
# Optimize and evaluate the Random Forest model using the defined grid
of hyperparameters
```

```
optimize_and_evaluate_model(best_rf_model, param_grid_rf,
X_train_scaled_1, y_train_1, X_valid_scaled_1, y_valid_1,
X_test_scaled_1, y_test_1, set_name="Random Forest")
train_and_evaluate_model(best_rf_model, X_train_scaled_1, y_train_1,
X_valid_scaled_1, y_valid_1, X_test_scaled_1, y_test_1)

# Define parameter grid for XGBoost model
param_grid_xgb = {
                  'max_depth': [3, 4, 5],
                  'learning_rate': [0.1, 0.01, 0.001],
                  'n_estimators': [50, 100, 200],
                  'subsample': [0.8, 0.9, 1.0],
                  'colsample_bytree': [0.8, 0.9, 1.0]
                  }

# Initialize XGBoost model
xgb_model = xgb.XGBClassifier(random_state=42,
objective='multi:softmax', num_class=9)

# Perform grid search and evaluate model
optimize_and_evaluate_model(xgb_model, param_grid_xgb,
X_train_scaled_1, y_train_1, X_valid_scaled_1, y_valid_1,
X_test_scaled_1, y_test_1, set_name="XGBoost")
train_and_evaluate_model(xgb_model, X_train_scaled_1, y_train_1,
X_valid_scaled_1, y_valid_1, X_test_scaled_1, y_test_1)

# Initialize the JS visualization for SHAP plots
shap.initjs()
# Train an XGBoost Regressor model using the validation set
model = xgb.XGBClassifier().fit(X_valid_scaled_1, y_valid_1)
# Create a SHAP explainer for the XGBoost model
explainer = shap.Explainer(model)
# Generate SHAP values for the entire dataset
shap_values = explainer(X)
# visualize the first prediction's explanation
shap.plots.waterfall(shap_values[0])
# visualize the first prediction's explanation with a force plot
shap.plots.force(shap_values[0])
# visualize the Secondary prediction's explanation with a force plot
shap.plots.force(shap_values[1])
# visualize all the training set predictions
shap.plots.force(shap_values[:500])
# summarize the effects of all the features
shap.plots.beeswarm(shap_values)
# create a dependence scatter plot to show the effect of a single
feature across the whole dataset
shap.plots.scatter(shap_values[:, "amount"], color=shap_values)
shap.plots.bar(shap_values)
```

```
# 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()

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 is 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)
    conf_matrix = confusion_matrix(y, y_pred)

    print(f'\n{"="*20} {type(model).__name__} {"="*20}\n')
    print(f'{set_name} Set Evaluation: {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"ROC AUC: {roc_auc:.4f}")
    cm_df = pd.DataFrame(conf_matrix, index=model.classes_,
    columns=model.classes_)

    print(f'{set_name} Set Evaluation: {type(model).__name__}')
    print(f'\n{"="*20}{"="*20}\n')
    print(f'Classification Report: {set_name} set -
    {type(model).__name__} \n')
    print (classification_report(y,y_pred))
    print(f'\n{"="*20}{"="*20}\n')
    print(f'Confusion Matrix: {set_name} set - {type(model).__name__}
    \n')
    print("Confusion Matrix:")
    print(cm_df)
    print(f'\n{"="*20}{"="*20}\n')
    plt.figure()
    plt.imshow(cm_df, cmap='Reds')
    plt.title(f'Confusion Matrix {set_name} Set Evaluation:
    {type(model).__name__}')
    plt.colorbar()
    plt.xlabel('Predicted label')
    plt.ylabel('True label')
    plt.tick_params(axis='both', which='major', labelsize=8)
    plt.tight_layout()
    plt.show()

def plot_roc_curve(fpr, tpr, auc_score, set_name="", model_name=""):
```

```
        """
        Plot ROC Curve for a given set.
        Parameters:
        - fpr: False Positive Rate.
        - tpr: True Positive Rate.
        - auc_score: Area Under the ROC Curve.
        - set_name: The name of the dataset (e.g., "Validation", "Test").
        - model_name: The name of the classification model.
        """
        plt.figure(figsize=(8, 8))
        plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'{set_name}
        ROC Curve (AUC = {auc_score:.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(f'ROC Curve - {set_name} Set: {model_name}')
        plt.legend()
        plt.show()


# Fit the Random Forest model
best_rf_model.fit(X_train_scaled_1, y_train_1)

# 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 - desired_recall))

# Get the corresponding threshold
desired_threshold = thresholds[closest_point_index]

adjusted_predictions_valid = (y_valid_pred_proba >=
desired_threshold).astype(int)
# 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 =
(best_rf_model.predict_proba(X_test_scaled_1)[:, 1] >=
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,
best_rf_model.predict_proba(X_test_scaled_1)[:, 1])
roc_auc_test = auc(fpr_test, tpr_test)

# Print the desired threshold
print(f"Desired Threshold: {desired_threshold:.4f}")
# Adjust the decision threshold for the validation set
# Evaluate the model on the validation set
evaluate_model(best_rf_model, X_valid_scaled_1, y_valid_1,
set_name="Validation")
plot_roc_curve(fpr_valid, tpr_valid, roc_auc_valid,
set_name="Validation", model_name="Random Forest")
# Evaluate the model on the test set
evaluate_model(best_rf_model, X_test_scaled_1, y_test_1,
set_name="Test")
plot_roc_curve(fpr_test, tpr_test, roc_auc_test, set_name="Test",
model_name="Random Forest")

def plot_confusion_matrix(confusion_matrix, title):
      # Calculate total correctly classified and misclassified
      instances
      correctly_classified = np.trace(confusion_matrix)
      misclassified = np.sum(confusion_matrix) - correctly_classified

      # Create a new figure and a set of subplots
      fig, ax = plt.subplots()

      # Create a pie chart
      ax.pie([correctly_classified, misclassified], labels=['Correctly
      Classified', 'Misclassified'], colors=['#ff9999', '#66b3ff'],
      autopct='%1.1f%%', shadow=True, startangle=140)

      # Display the pie chart
      plt.title(title)
      plt.show()

      # Print the results
      print("Total Correctly Classified Instances:",
      correctly_classified)
```

```
      print("Total Misclassified Instances:", misclassified)

# Confusion matrix of Model A on Validation Set
confusion_matrix_a_validation = np.array([[3259, 786],
                                          [779, 3212]])

plot_confusion_matrix(confusion_matrix_a_validation, 'Correctly vs
Misclassified Instances on Validation Set of Model')

# Confusion matrix of Model A on Test Set
confusion_matrix_a_test = np.array([[3266, 757],
                                    [ 823, 3191]])

plot_confusion_matrix(confusion_matrix_a_test, 'Correctly vs
Misclassified Instances on Test Set of Model')



#Evaluate Cross-Validation Stability of Model
# Define the parameter grid for hyperparameter tuning
param_grid = {
            'n_estimators': [50, 100, 200],
            'max_depth': [10, 20, 30],
            'min_child_weight': [2, 5, 10],
            'learning_rate': [0.1, 0.01, 0.001]
            }

# Create a scaler object
scaler = StandardScaler()

# Fit the scaler on the training data
scaler.fit(X_train_1)

# Scale the training and testing data
X_train_scaled_1 = scaler.transform(X_train_1)
X_test_scaled_1 = scaler.transform(X_test_1)

# Create GridSearchCV
grid_search = GridSearchCV(estimator=XGBClassifier(),
param_grid=param_grid, cv=5, scoring='accuracy', 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_xgb = 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_xgb[f'mean_test_score']
        [i]}")
        print(f"Standard Deviation Test Score:
        {cv_results_xgb[f'std_test_score'][i]}")
        print(f"Params: {cv_results_xgb['params'][i]}")

# Define the parameter grid for hyperparameter tuning
param_grid = {
            'n_estimators': [50, 100, 200],
            'max_depth': [10, 20, 30],
            'min_samples_split': [2, 5, 10],
            'min_samples_leaf': [1, 2, 4]
            }
# Create GridSearchCV
grid_search = GridSearchCV(estimator=best_rf_model,
param_grid=param_grid, cv=5, scoring='accuracy', 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
# 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")

        # Print calculations
        print(f"Train Sizes: {train_sizes}")
        print(f"Training Scores Mean: {train_scores_mean}")
        print(f"Training Scores Std: {train_scores_std}")
        print(f"Test Scores Mean: {test_scores_mean}")
        print(f"Test Scores Std: {test_scores_std}")
        return plt

# Plot learning curve
title = "Learning Curve (Random Forest) for Model"
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
success_probabilities = xgb_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()
# Drop the 'transaction_fee' column from the copied DataFrame
# 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)
# Split the dataset into features (X) and target variable (y)
X = df_encoded_2.drop(['PSP', 'success'], axis=1)
y = df_encoded_2['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, random_state=42)
# Create and train a KNN model
knn_model = KNeighborsClassifier(n_neighbors=5)
knn_model.fit(X_train, y_train)
# 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, random_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_model2(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'\n{"="*20} {type(model).__name__} {"="*20}\n')
      print(f'{set_name} Set Evaluation: {type(model).__name__}')
      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}")
        cm = confusion_matrix(y, y_pred)
        cm_df = pd.DataFrame(cm, index=model.classes_,
        columns=model.classes_)
        print(f'\n{"="*20}{"="*20}\n')
        print(f'Classification Report: {set_name} set -
        {type(model).__name__} \n')
        print(classification_report(y_values, y_pred))
        print (classification_report(y,y_pred))
        print(f'\n{"="*20}{"="*20}\n')
        print(f'Confusion Matrix: {set_name} set - {type(model).__name__}
        \n')
        print("Confusion Matrix:")
        print(cm_df)
        plt.figure()
        plt.imshow(cm_df, cmap='Reds')
        plt.title(f'Confusion Matrix {set_name} Set Evaluation:
        {type(model).__name__}')
        plt.colorbar()
        plt.xlabel('Predicted label')
        plt.ylabel('True label')
        plt.tick_params(axis='both', which='major', labelsize=8)
        plt.tight_layout()
        plt.show()

# Create and train Random Forest model with specified parameters
rf_model_B = 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_B, X_train_scaled_2, y_train_2,
scoring='f1_weighted', cv=StratifiedKFold(n_splits=5))
# Print cross-validation scores
print("Cross-Validation Scores:\n", cv_scores_rf)
print(f"Average F1 Weighted Score: {cv_scores_rf.mean():.4f}")

# Fit the Logistic Regression model on the training set
rf_model_B.fit(X_train_scaled_2, y_train_2)
# Print model performance on the validation set
print("Model Performance on Validation set - Random Forest::")
evaluate_model2(rf_model_B, X_valid_scaled_2, y_valid_2,
set_name="Validation")
# Print model performance on the test set
print("\nModel Performance on Test set - Random Forest::")
evaluate_model2(rf_model_B, X_test_scaled_2, y_test_2, set_name="Test")
# Random Forest Model  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': [10, 20, 30]
      }
# Use GridSearchCV for hyperparameter tuning
rf_grid = GridSearchCV(rf_model, param_grid, scoring='f1_weighted',
cv=StratifiedKFold(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, scoring='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_model2(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_model2(rf_best_model, X_test_scaled_2, y_test_2,
set_name="Test")

# Precision-Recall curve for validation and test set of Model B
# Binarize the labels for precision-recall curve
from sklearn.preprocessing import StandardScaler, LabelBinarizer

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_recall_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_recall_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
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")

# Cross-Validation stability of SubModel
# Cross-validation scores with the best model
cv_scores_rf = cross_val_score(rf_best_model, X_train_scaled_2,
y_train_2, scoring='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
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")

        # Print calculations
        print(f"Train Sizes: {train_sizes}")
        print(f"Training Scores Mean: {train_scores_mean}")
        print(f"Training Scores Std: {train_scores_std}")
        print(f"Test Scores Mean: {test_scores_mean}")
        print(f"Test Scores Std: {test_scores_std}")
        return plt
        # Plot learning curves
        plot_learning_curve(rf_best_model, "Model 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)

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
                              })

# Plot the count
plt.figure(figsize=(10, 5))
```

```
plt.bar(matching_counts.index, matching_counts['Predicted_PSP_Count'],
label='Predicted_PSP')
plt.bar(matching_counts.index, matching_counts['Best_PSP_Count'],
label='Best_PSP', bottom=matching_counts['Predicted_PSP_Count'])
plt.xlabel('PSP')
plt.ylabel('Count')
plt.title('Count of Predicted_PSP and Best_PSP')
plt.legend()
plt.show()
```