Predicting Customer Churn in the Banking Sector: A Machine Learning Approach

Higher Diploma in Science in Data Analytics

Final Report

Word Count: 3953

Raju Khan
20033471@mydbs.ie

Supervisor: DR. Shazia A. Afzal

January 2025

Abstract

Customer churn is a major challenge for the banking industry, directly affecting profitability and long-term growth. This study examines predictive modelling and customer segmentation to efficiently address churn using a dataset including customer demographics, financial behaviour and churn status. Logistic regression has been used as a starting model, with random forest and K-nearest neighbours (KNN) is incorporated to capture nonlinear relationships and increases the accuracy prediction. Advanced methods such as SMOTE were used to deal with class imbalances, while K-means refined clustering divided into three distinct classes for usable insights and evaluate model performance with accuracy, precision and statistical efficiency By integrating predictive modelling, this work provides a comprehensive framework for pre-empting customer churn and tailoring retention strategies The findings contribute both theoretical insights and practical applications of customer relationship management for the banking industry.

Acknowledgments

I would like to express my sincere gratitude to Dublin Business School (DBS) for providing the resources and support necessary to complete this project. I am deeply thankful to my project supervisor, Dr. Shazia A. Afzal, for her invaluable guidance, encouragement, and expertise throughout the research process. Her insights have been instrumental in shaping the direction and outcomes of this study. I am also thankful to my fellow classmates for their advice and support.

Table of Contents

Abstract 2

Acknowledgments 3

Chapter 1: Introduction 5

Chapter 2: Background/Literature Review 5

- Project Background: 5
- Associated Theory: 6
  - CRISP-DM: 6
  - Customer Segmentation: 7
  - Imbalanced Classification: 7
  - Logistic Regression: 7
  - K-Nearest Neighbours (KNN): 8
  - Random Forest: 8
- Anticipated Benefits of System: 8
- Typical Users of the Project Product: 8
- Relevant/Similar Existing Solutions: 9

Chapter 3: Requirements Specification and Design 9

- Business/Project Need: 9
- Information Requirements: 9
- Tools and Equipment: 10
- Design: 10

Chapter 4: Implementation 11

- Business Understanding: 11
- Data Understanding: 11
- Data Preparation: 12
- Modelling: 12
- Evaluation: 13
- Deployment: 13

Chapter 5: Testing and Results 13

- Results: 14
- Shortcomings: 15

Chapter 6: Conclusion and Future Work 16

- Future Work: 16

References and Bibliography 17

Appendices 18

- Tables: 18
- Code Snippets: 18

Chapter 1: Introduction

The Banking Churn Project aims to analyse and predict customer churn within the banking industry, leveraging advanced machine learning techniques. Since keeping existing clients is frequently more cost-effective than finding new ones, churn prediction is an essential component for banks. This initiative aims to assist decision-makers in putting targeted retention strategies into practice by identifying the factors that contribute to customer attrition.

This study uses publicly available data from Kaggle (Badole, S., 2024), including customer demographics, investment behaviour and churn status. The scope of this project encompasses two key areas: predictive modelling and customer segmentation. Predictive modelling focuses on customers at risk of churn by using logistic regression, random forest, and K-nearest neighbours (KNN) models. Customer classification, implemented through K-means clustering, aims to group customers into three different categories to reveal patterns and propose tailored retention strategies.

The **approach** taken in this project involves a multi-step process:

1. Data Preprocessing: Cleaning, transforming, addressing class imbalance, and normalizing the dataset to ensure it is ready for analysis.
2. Exploratory Data Analysis (EDA): Understanding the data distribution, identifying trends, and detecting relationships between variables.
3. Creating Customer Segments using K-means clustering to group customers based on Age and Balance.
4. Model Development: Implementing a baseline logistic regression model, followed by more complex models like Random Forest and K-Nearest Neighbours (KNN).
5. Evaluation: Comparing model performances using metrics such as accuracy, precision, recall, and F1-score.
6. Deploying best performing model for real-world applications.

By focusing on customer segmentation and leveraging advanced imbalance-handling techniques, this project not only aims to build predictive models but also provides meaningful insights for strategic interventions to reduce churn.

Chapter 2: Background/Literature Review

Project Background:

In recent years, the banking industry has become increasingly competitive, with customers having more options than ever before. One of the biggest challenges for banks is to retain their customers and prevent them from leaving their bank. Customer churn, or the loss of valuable customers can significantly impact a bank's revenue and growth, hence, it's crucial to make customer churn prediction as it can forestall potential churning while also improve bank services, along with the gains of the institute (Briker, V., Farrow, R., Trevino, W., & Allen, B., 2019). The cost of customer acquisition is high, and losing a customer can have a significant impact on the bank's revenue and profitability. Therefore, it is important for banks to identify the factors that contribute to customer churn and to develop strategies to retain their customers (Mr. Shadakshari, Srijan S., Prashant K., Aniket S., Shivam C., 2024). The project leverages datadriven analytics to address this challenge by predicting churn and enabling banks to proactively engage with at-risk customers.

Zhu H. et al. (2023) study examined the comparative performance of logistic regression (LR) and random forest (RF) models in predicting customer churn. Using a Kaggle data set of 120,000 records, the study used K-fold cross-validation to ensure robust analysis and addressed data imbalance through feature engineering. The results showed that RF outperformed LR, with an average cross-validation of 86.2% compared to 79.3% for LR. Among the predictors, age emerged as the most important factor, followed by account balances and credit scores. The findings highlight the effectiveness of RF and other clustering techniques in processing complex data and identifying important churn predictors.

Tran, H., Le, N., & Nguyen, V. et al. (2023) investigated the role of customer segmentation in enhancing churn prediction accuracy while evaluating multiple ML models, including LR, KNN, decision trees (DT), random forest (RF), and support vector machines (SVM). Their methodology included the use of K-means clustering for segmentation and SMOTE for addressing class imbalance. The study revealed that RF achieved the highest accuracy (97.25%), closely followed by SVM (96.63%). However, the impact of customer segmentation on prediction precision varied, depending on dataset characteristics and model configurations. This research highlights the potential for combining segmentation with advanced ML techniques to improve prediction performance, although it emphasizes that its effectiveness is context dependent.

Shadakshari et al. (2024) conducted a comparative analysis of various ML models, including LR, DT, RF, and SVM, for churn prediction. Their study utilized a dataset enriched with demographic and transactional information, addressing data imbalance through random oversampling. Among the evaluated models, SVM achieved the highest accuracy (92%), outperforming RF (85%) and LR (82%). The study identified account age, transaction frequency, and account balance as the most critical predictors of churn. These findings suggest that while RF remains a robust choice, SVM can offer competitive performance, particularly when supported by balanced datasets and effective feature selection.

Associated Theory:

CRISP-DM:

In the world of data science, a structured approach is crucial for guiding projects from inception to completion. Enter the CRISP-DM (Cross-Industry Standard Process for Data Mining) program—a complex, robust framework for data mining projects. Its versatility has made it an industry standard for small and medium-sized businesses across a variety of industries (Chumbar, S., 2023).

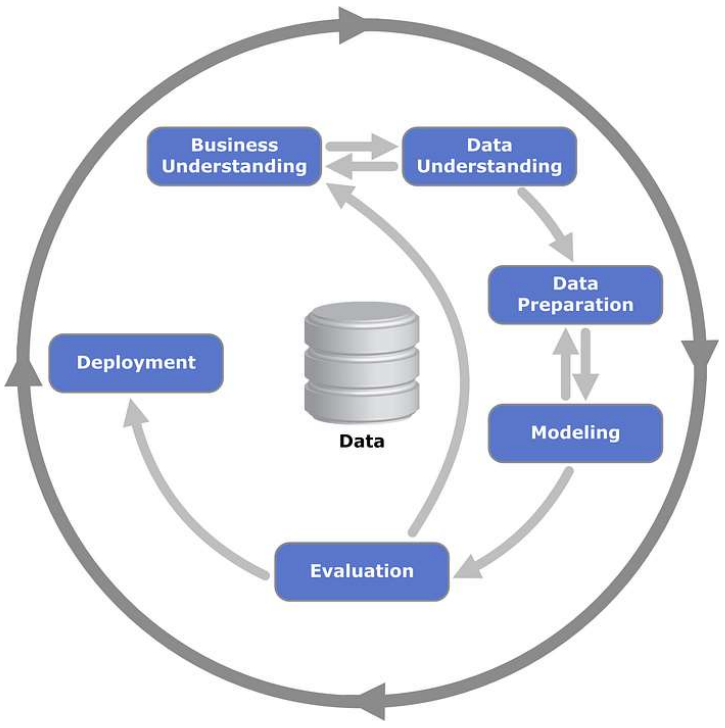


Figure 1: Diagram of the CRISP-DM process. *(Shearer, C., 2000).*

CRISP-DM is a data mining process model that is independent of industry. From business understanding to deployment, it goes through six iterative phases (see Table 1 in Appendices). The primary concept, tasks, and results of these phases are briefly described in Table 1 (Ncr and Clinton, J., 2000).

Customer Segmentation:

Is the practice of dividing a customer base into groups of individuals that have similar characteristics relevant to marketing, such as age, gender, interests and spending habits (Barney, N. and Gillis, A.S.,2024).

Imbalanced Classification:

SMOTE is an oversampling technique where the synthetic samples are generated for the minority class. The overfitting issue caused by random oversampling is mitigated by this algorithm. With the aid of interpolation between the positive instances that lie together, it concentrates on the feature space to produce new instances (Satpathy, S., 2020).

ADASYN is a machine learning technique used to address imbalanced datasets by generating synthetic samples for underrepresented classes. It works by generating synthetic samples for minority classes based on the feature space of the original dataset. Its greatest advantages are not copying the same minority data and generating more data for “harder to learn” examples (Nian, R., 2019).

Logistic Regression:

Logistic regression in machine learning is a statistical method that is used for building machine learning models where the dependent variable is dichotomous: i.e. binary. Logistic regression is used to describe data and the relationship between one dependent variable and one or more independent variables. The independent variables can be nominal, ordinal, or of interval type (Banoula, M., 2024).

K-Nearest Neighbours (KNN):

The k-nearest neighbours (KNN) algorithm is a non-parametric, supervised learning classifier, which uses proximity to make classifications or predictions about the grouping of an individual data point. It is one of the popular and simplest classification and regression classifiers used in machine learning today (IBM 2021).

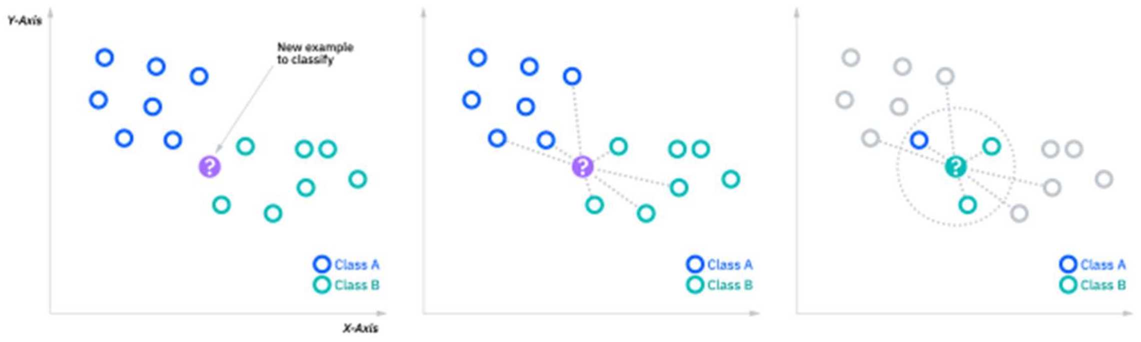


Figure 2: KNN Diagram.

A majority vote determines the class label for classification problems; that is, the label most commonly associated with a particular data point is used.

Random Forest:

A Random Forest is an ensemble machine learning model that combines multiple decision trees. Each tree in the forest is trained on a random sample of the data (bootstrap sampling) and considers only a random subset of features when making splits (Baladram S., 2024).

The forest averages the predictions for regression tasks and makes predictions for classification tasks based on majority voting among trees. The "wisdom of crowds" approach of the model is its strongest point; although individual trees may make mistakes, the group's decision-making process tends to average out these errors and produce predictions that are more trustworthy.

Anticipated Benefits of System:

The primary benefit of the churn prediction system is its ability to improve customer retention by identifying individuals at risk of leaving. By providing actionable insights, the system helps banks:

- Optimize resource allocation for retention efforts.
- Increase customer lifetime value.
- Enhance customer satisfaction by addressing pain points proactively.

Secondary benefits include supporting customer segmentation for personalized marketing and reducing costs associated with high customer turnover.

Typical Users of the Project Product:

The end-users of this predictive system include:

- Customer Relationship Managers: To identify and engage with at-risk customers.

- Marketing Teams: To design targeted campaigns based on churn likelihood and customer profiles.
- Bank Executives: To make strategic decisions on customer retention policies and resource allocation.
- Data Analysts: To monitor, refine, and validate the performance of the churn prediction models.

Relevant/Similar Existing Solutions:

Several banking and fintech companies currently utilize churn prediction systems, often integrated with Customer Relationship Management (CRM) software. Examples include:

- **Salesforce Einstein AI:** Provides churn prediction within its CRM platform.
- **Zetaris:** A churn analytics tool offering real-time insights using machine learning.
- **Python-based Open-Source Models:** Various machine learning libraries (e.g., scikit-learn, TensorFlow) enable the development of customizable churn prediction systems.

While existing solutions are effective, they often rely on generalized models or require significant resources for implementation. The Banking Churn Project focuses on building a tailored, cost effective, and interpretable model suited to the specific needs of the bank under study.

Chapter 3: Requirements Specification and Design

Business/Project Need:

The banking industry faces high customer turnover rate, which leads to significant financial losses in the banking sector. Finding at-risk clients and addressing the underlying reasons for churn are the main objectives of this initiative. This involves building a predictive model to answer critical business questions:

- Which customers are likely to churn?
- What are the key factors contributing to churn?
- How can customer retention strategies be tailored based on predictive insights?

Addressing these questions will enable banks to reduce churn, optimize marketing and retention strategies, and improve overall customer satisfaction.

Information Requirements:

To answer the business questions effectively, the following data is necessary which is available in the dataset acquired from Kaggle as mentioned above:

- Demographics: Age, gender, geography.
- Financial Behaviour: Credit score, balance, number of products held.
- Engagement Metrics: Tenure, activity status, credit card ownership.
- Churn Status: Whether the customer has exited (target variable).

Tools and Equipment:

The following technologies/Libraries shall be used in the course of the project:

- Google Collaboratory Notebook employing Python coding software.
- Data Preprocessing tools: Pandas and Numpy for cleaning the dataset and feature engineering.
- Scikit-learn library to apply machine learning models such Logistic Regression, KNN and Random Forest models.
- Visualisation tools: Seaborn and Plotly for understanding data trends and model performance visually.
- SMOTE and balancing techniques: Libraries like imblearn to handle imbalanced target variable (churn).
- Evaluation Metrics: Accuracy, Precision and Recall to evaluate model performance and compare each model to figure out which is best model for deployment.

Design:

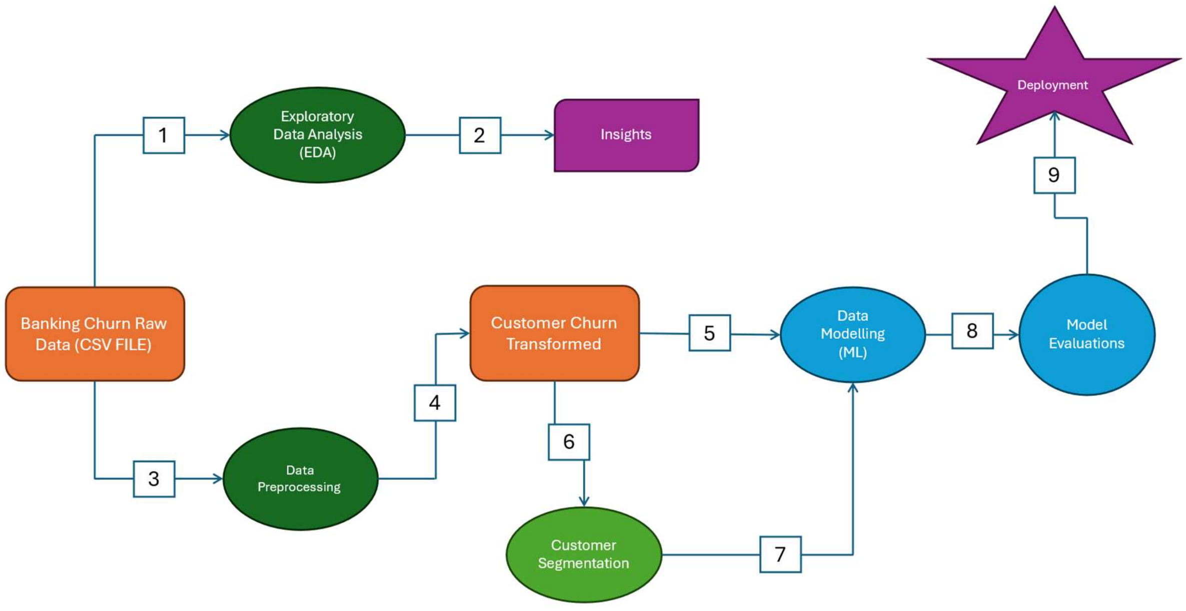


Figure 3: Customer Churn High Level Design.

1. Exploratory Data Analysis in Google Collab Notebook using python libraries Seaborn and Plotly. E.g. exploring missing values and patterns.
2. Derive some insights from initial exploration. E.g. imbalance of customers that churn and customers that do not churn.
3. Data Preprocessing in Google Collab Notebook.
4. Customer Churn Data Transformed: resulting after cleaning of initial raw data and now prepared for modelling.
5. Machine learning models employed. E.g. Logistic Regression, KNN Regression and Random Forest.
6. Customer Segments created using K-means clustering.
7. ML models applied to each segment.

8. Each model evaluated using performance metrics and compared. E.g. Accuracy, Precision and Recall.
9. Best performing model is deployed.

Chapter 4: Implementation

This Data Science Project was implemented using the CRISP-DM methodology which was outlined in Chapter two of this report. As thus, this methodology follows:

Business Understanding:

Customer churn is one of the most important challenges in the banking industry, directly affecting profitability and sustainability. Retaining existing customers is significantly more cost-effective than acquiring new ones. Thus, understanding the factors that contribute to churn and proactively identifying customers who are likely to leave are essential to developing effective retention strategies.

The aim of this project is to develop predictive models to identify consumers at risk of being churned based on demographic, economic and behavioural data. Using advanced machine learning techniques, the goal is to generate actionable insights that can identify targeted retention campaigns, optimize resource distribution, and to provide increased overall satisfaction to consumers.

Data Understanding:

As mentioned previously, the dataset sourced for this data science project is publicly available in Kaggle. In order, to see what this dataset included I performed some Exploratory Data Analysis using Python libraries Pandas, Plotly and Seaborn.

Firstly, we checked the following details about the dataset and their corresponding Code Snippets are available in the Appendices section:

- The dimensions of the dataset.
- The data type of each column in the dataset.
- Checked if there were any missing values in the dataset.
- Summary Statistics. E.g. mean, median, count etc.

Next, univariate analysis, which consists of analysing a single variable. This was conducted using Plotly and Seaborn, so that this analysis can be visualized. The following visuals were created:

- Churn Distribution. E.g. count of customers who exited and stayed.
- Histograms for Numeric features. E.g. Age, Balance, CreditScore etc.
- Barplots for Categorical variables. E.g. Gender, Geography etc.

Next, bivariate analysis, which consists of analysing two variables. Each of the features in the dataset was analysed against churn. Also, a correlation matrix was created to analyse the relationship between the numeric features.

Lastly, multivariate analysis was conducted. This consisted of creating a scatter plot Age versus Balance coloured by churn.

Data Preparation:

There were no values missing as we saw in the EDA, but we did have a lot of outliers especially in the CreditScore and Age features. Box-plot technique was used to remove these outliers from these variables. Also, unnecessary variables such as, Rownumber, CustomerId and Surname were removed. These columns were removed because they were insignificant in the modelling stage. Table 2 in appendices section consists of the features used in the bank customer churn analysis.

Afterwards, I had to encode the categorical variables into numeric. Machine learning models can only have numerical inputs. Therefore, I had to use binary encoding for the Gender variable. Where, male was equal to 0 and female was equal to 1. For Geography variable, I had to do one-hot encoding using Pandas get dummies function.

Next step was to separate the inputs and target variable. I assigned the Target variable (Exited) to Y and all other features in the dataframe to X. Then I used the StandardScaler to scale the inputs (X). This ensures that the values are centered around the mean with a unit standard deviation.

The next pre-processing step before a regression model is run is to split the dataset into training and testing sets. Splitting the dataset into training and testing sets is essential for evaluating the performance and generalization of a machine learning model. It helps avoid overfitting by ensuring that the model is not biased toward the training data. It provides a realistic measure of how the model will perform on new, unseen data, which is crucial for model selection and hyperparameter tuning. The dataset is split using the train_test_split function from scikit library. 20% of the data was used for testing, and the remaining 80% was used for training.

Lastly, I used SMOTE and ADASYN for handling class imbalance. Both of these techniques are oversampling methods. SMOTE works by creating synthetic samples for the minority class. It does this by selecting a minority class instance and its k nearest neighbours. It then generates synthetic examples by interpolating between the selected instance and its neighbours. ADASYN takes a more adaptive approach. It focuses on the minority instances that are difficult to classify correctly, rather than oversampling all minority instances uniformly. It assigns a different weight to each minority instance based on its level of difficulty in classification.

Modelling:

Three different Machine Learning Models were constructed in this data science project, and they were:

- Logistic Regression Model.
- K-Nearest Neighbour (KNN) Model.
- Random Forest Model.

Two separate models were created for each of the ML models listed above. One employing SMOTE and the other ADASYN to handle class imbalance for the Target variable. Both imbalance handling techniques were compared and better performing one was identified.

Afterwards, I conducted some Hyperparameter Tuning for each ML model. I used the GridSearchCV function to determine the best set of parameters for each model.

Next, Customer segments were created. Segmenting customers allows us to group them based on similar characteristics, which can help improve model accuracy and interpretability. For example, older customers with high balances may behave differently than younger customers with low balances, so treating them separately may yield better predictions. The following steps were taken:

- Optimal number of 3 clusters determined using Elbow method.
- K-means clustering was applied.
- Scatter plot created to visualize segments.
- ML models applied to each segment.

Evaluation:

The evaluation phase focuses on the effectiveness of machine learning models developed to predict customer churn. The main evaluation metrics used are precision, accuracy, recall, and F1-score, to ensure a balanced assessment of model performance given the imbalanced classes in the dataset. Models tested include logistic regression, K-nearest neighbours (KNN), and random forest. The best performing model is determined using the classification report of each model and compared.

Customer segmentation was also reviewed to ensure meaningful grouping based on demographic and behavioural characteristics.

Deployment:

This stage involves integrating the developed best performing model into the bank's operational system to predict customer churn in real time. The predictive model will be integrated into the bank's CRM system to flag high-risk customers, enabling the retention team to take targeted action promptly.

Chapter 5: Testing and Results

The testing phase of the Banking Churn project was conducted following the CRISP-DM methodology, emphasizing iterative evaluation and refinement of the models. The main objectives were to validate the models' performance and ensure that customer segmentation and churn predictions were aligned with project objectives. The testing process involved splitting the data into training and testing sets, applying imbalancing techniques such as SMOTE/ADASYN, and evaluating three machine learning models: Logistic Regression, K-Nearest Neighbours (KNN), and Random Forest.

Results:

From the churn distribution that was conducted in the EDA section we saw that there was a major class imbalance between Class 0 and Class 1 as seen below in figure 4.

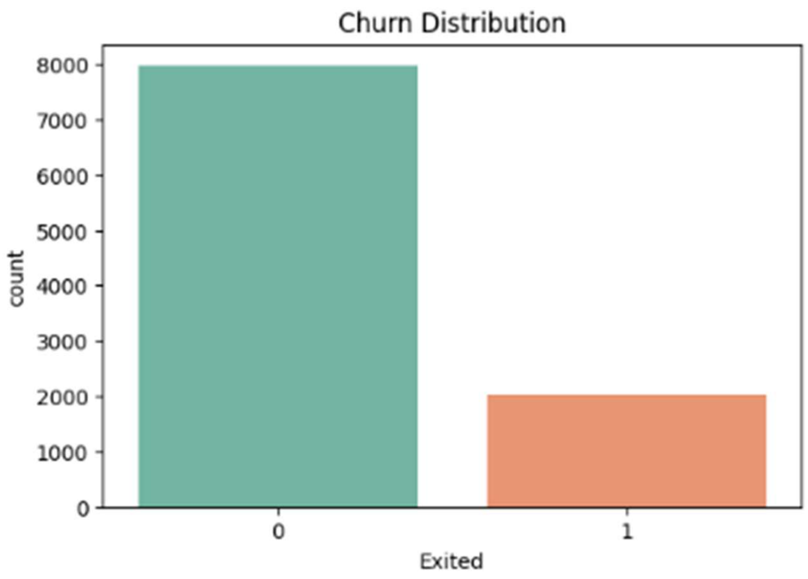
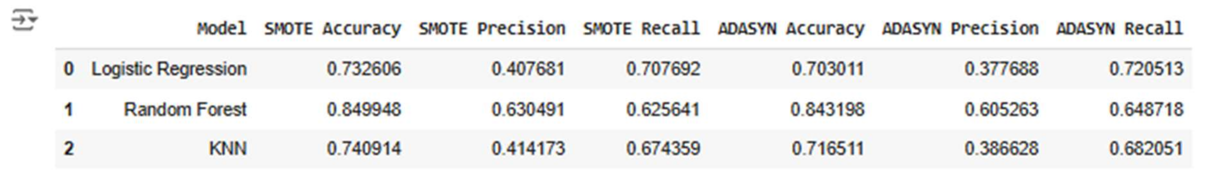


Figure 4: Bar plot of Churn Distribution from EDA.

So therefore, we employed imbalance handling techniques, which were SMOTE and ADASYN. I ran Logistic Regression, KNN and Random Forest models using both the imbalance handling techniques and SMOTE models performed better than ADASYN models as can be seen in figure 5.



	Model	SMOTE Accuracy	SMOTE Precision	SMOTE Recall	ADASYN Accuracy	ADASYN Precision	ADASYN Recall
0	Logistic Regression	0.732606	0.407681	0.707692	0.703011	0.377688	0.720513
1	Random Forest	0.849948	0.630491	0.625641	0.843198	0.605263	0.648718
2	KNN	0.740914	0.414173	0.674359	0.716511	0.386628	0.682051

Figure 5: Comparison of SMOTE and ADASYN.

Next, I compared each of the ML models with their corresponding Hyperparameter Tuning models, which was conducted using GridSearch to determine the best parameters to use. The GridSearch was conducted using cross-validation of 5 folds. Both the normal and tuned Logistic Regression model was the same. Both had an accuracy of 73% and their precision and recall were the same as well for both classes. The KNN model differed slightly. The tuned KNN showed better accuracy of 78% and precision for class 1. But recall for class 0 and class 1 increases and decreases respectively. According to f1-score, the tuned KNN model is better. Accuracy for both the normal Random Forest and Tuned model were the same of 85%. Precision for class 1 increases in the Tuned model but everything else decreases slightly. Therefore, I determined that the normal Random Forest model performed better.

Using the K-Means algorithm, the dataset was divided into three customer segments to capture finer details of customer behaviour.

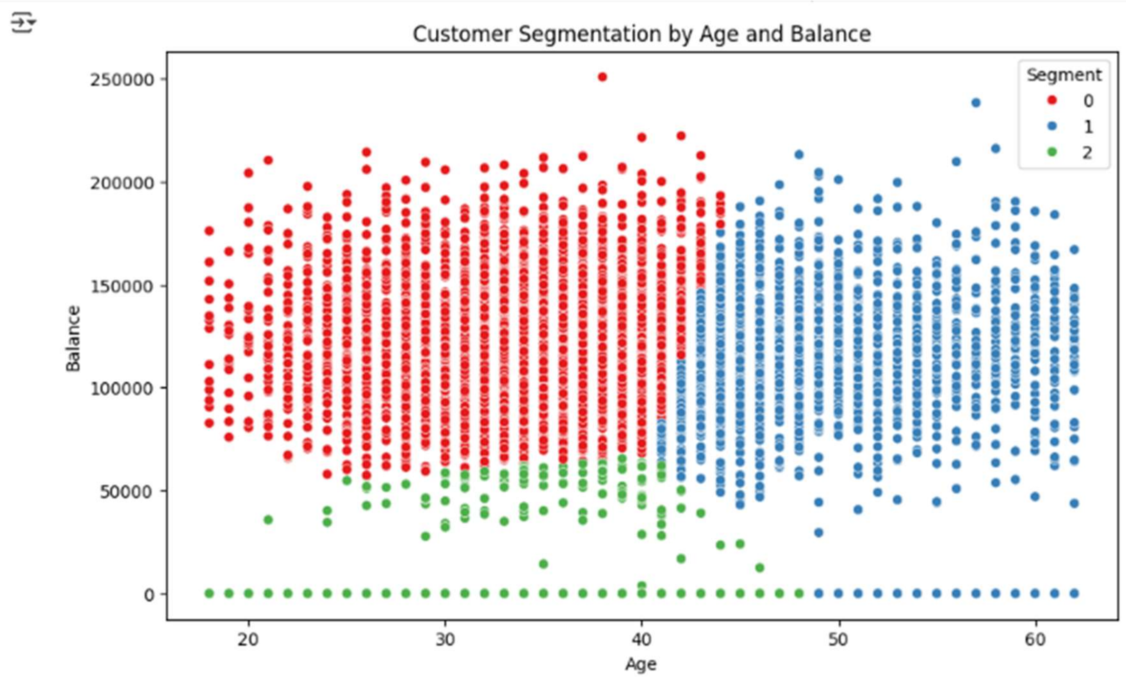


Figure 6: Three Customer Segments Created.

Red (Segment 0): Likely represents customers with moderate to high balances and ages in a middle range (~30-50 years). This segment might include financially stable, working professionals or families. Blue (Segment 1): Likely represents older customers (~50-70 years) with varying balances. This segment could include retirees or customers nearing retirement, with established financial profiles. Green (Segment 2): Likely represents younger customers (~20-30 years) with lower balances. This group might consist of students, early-career individuals, or customers with lower financial activity.

These segments provided actionable insights into customer demographics, product preferences, and churn likelihood. Segment analysis revealed distinct patterns, such as higher churn rates among younger customers with low account balances.

Shortcomings:

Random Forest, while the best-performing model, had significantly higher computational requirements compared to Logistic Regression and KNN. The high performance of Random Forest could indicate potential overfitting. Further testing on unseen data is needed to confirm its generalizability. The dataset lacked certain potentially influential features such as customer feedback or transaction history, which could enhance predictive power. While effective, the segmentation approach relied on basic K-Means clustering, which may not fully capture the complexities of customer behaviour.

Chapter 6: Conclusion and Future Work

The Banking Churn project focused on predicting customer churn and offering actionable insights through customer segmentation and advanced methods for addressing class imbalance. We used Logistic Regression as a baseline model and compared its performance with more complex models like KNN and Random Forest, effectively showcasing the strengths and weaknesses of each approach.

Key findings of the project highlight that Random Forest outperformed other methods in predictive accuracy and precision, though it requires more computational resources. On the other hand, Logistic Regression, being less complex, offered a solid baseline for both interpretability and efficiency. Additionally, employing SMOTE to tackle class imbalance was found to be more effective than ADASYN, as it maintained balanced training data while preserving the integrity of minority class patterns.

Customer segmentation was re-evaluated into three distinct clusters, revealing valuable insights into the diversity of customers in terms of age, balance, and activity levels. This framework for segmentation can be effectively used to create targeted retention strategies, tailor personalized offers, and optimize resource allocation.

Using the CRISP-DM methodology provided a clear and organized framework for the project, guiding us from data comprehension to the implementation of actionable insights. This approach was key to the project's success and helped in effectively communicating the results.

Future Work:

Future studies in this area are required. From exploring more advanced predictive models to using more advanced techniques and algorithms. ML models such as XGBoost and LightGBM may further enhance predictive performance. Class imbalance can be experimented using hybrid imbalancing techniques, such as SMOTE-Tomek or ensemble SMOTE, to further improve data balance. Alternative clustering algorithms can be used such as DBSCAN or hierarchical clustering, to create customer segments. These algorithms may uncover potentially more meaningful segments. By using the insights gained from this project and tackling the recommended areas for future efforts, the bank can improve its grasp of customer churn, create effective retention strategies, and ultimately increase customer satisfaction and loyalty.

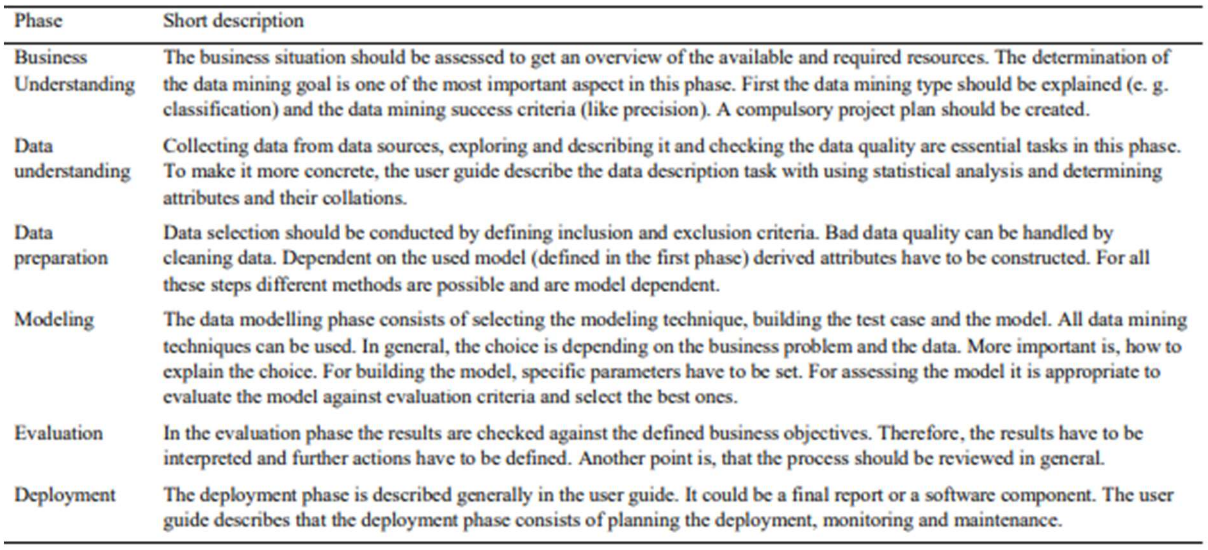
References and Bibliography

1. Badole, S. (2024). *Banking Customer Churn Prediction Dataset*. [online] Kaggle.com. Available at: https://www.kaggle.com/datasets/saurabhbadole/bank-customer-churn-prediction-dataset.
2. Baladram, S. (2024). Random Forest | Towards Data Science. [online] Medium. Available at: https://towardsdatascience.com/random-forest-explained-a-visual-guide-with-code-examples-9f736a6e1b3c.
3. Banoula, M. (2024). An Introduction to Logistic Regression in Python. [online] Simplilearn.com. Available at: https://www.simplilearn.com/tutorials/machine-learning-tutorial/logistic-regression-in-python.
4. Barney, N. and Gillis, A.S. (2024) What is customer segmentation? https://www.techtarget.com/searchcustomerexperience/definition/customer segmentation#:~:text=Customer%20segmentation%20is%20the%20practice,gender%2C%20interest s%20and%20spending%20habits.
5. Briker, V., Farrow, R., Trevino, W., & Allen, B. (2019). SMU Data Science Review, 2(3).
6. Chumbar, S. (2023). The CRISP-DM Process: A Comprehensive Guide. [online] Medium. Available at: https://medium.com/@shawn.chumbar/the-crisp-dm-process-a-comprehensive-guide-4d893aecb151.
7. Cote, C. (2024) A Beginner's Guide to Data & Analytics [e book]. Harvard Business School.
8. Ncr, Clinton, J. (2000). CRISP-DM 1.0 Step-by-step data mining guide. [online] DaimlerChrysler. Available at: https://www.kde.cs.uni-kassel.de/wp-content/uploads/lehre/ws2012-13/kdd/files/CRISPWP-0800.pdf.
9. Nian, R. (2019). An Introduction to ADASYN (with code!). [online] Medium. Available at: https://medium.com/@ruinian/an-introduction-to-adasyn-with-code-1383a5ece7aa.
10. SATPATHY, S. (2020). SMOTE - A Common Technique to Overcome Class Imbalance Problem. [online] Analytics Vidhya. Available at: https://www.analyticsvidhya.com/blog/2020/10/overcoming-class-imbalance-using-smote-techniques/.
11. Shadakshari, Shashwat, S., Himanshu, P., Singh, A., Chaudhary, S. (2024). A Comparative Study of Machine Learning Algorithms for Bank Customer Churn Prediction. East African Scholars J Eng Comput Sci, 7(1), 1-6.
12. Shearer, C., 2000. The CRISP-DM model: the new blueprint for data mining. *Journal of data warehousing*, *5*(4), pp.13-22.
13. Tran, H., Le, N., & Nguyen, V.-H. (2023). Customer churn prediction in the banking sector using machine learning-based classification models. Interdisciplinary Journal of Information, Knowledge, and Management, 18, 87-105. Available at: https://doi.org/10.28945/5086.
14. Zhu, H. (2024). Bank Customer Churn Prediction with Machine Learning Methods. Advances in Economics, Management and Political Sciences. 69. 23-29. 10.54254/2754 1169/69/20230773.

Appendices

Tables:

1. Table 1: CRISP-DM process model descriptions (Ncr and Clinton, J., 2000).



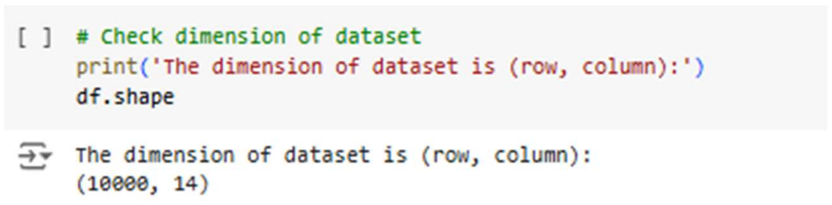
Phase	Short description
Business Understanding	The business situation should be assessed to get an overview of the available and required resources. The determination of the data mining goal is one of the most important aspect in this phase. First the data mining type should be explained (e. g. classification) and the data mining success criteria (like precision). A compulsory project plan should be created.
Data understanding	Collecting data from data sources, exploring and describing it and checking the data quality are essential tasks in this phase. To make it more concrete, the user guide describe the data description task with using statistical analysis and determining attributes and their collations.
Data preparation	Data selection should be conducted by defining inclusion and exclusion criteria. Bad data quality can be handled by cleaning data. Dependent on the used model (defined in the first phase) derived attributes have to be constructed. For all these steps different methods are possible and are model dependent.
Modeling	The data modelling phase consists of selecting the modeling technique, building the test case and the model. All data mining techniques can be used. In general, the choice is depending on the business problem and the data. More important is, how to explain the choice. For building the model, specific parameters have to be set. For assessing the model it is appropriate to evaluate the model against evaluation criteria and select the best ones.
Evaluation	In the evaluation phase the results are checked against the defined business objectives. Therefore, the results have to be interpreted and further actions have to be defined. Another point is, that the process should be reviewed in general.
Deployment	The deployment phase is described generally in the user guide. It could be a final report or a software component. The user guide describes that the deployment phase consists of planning the deployment, monitoring and maintenance.

2. Table 2: The variables and attribution used in the bank customer churn prediction analysis

Variable	Data Type	Description
CreditScore	Numerical	The credit score of the customer
Gender	Binary (Male or Female)	Sex of the customer
Age	Numerical	The age of the customer
Tenure	Numerical	For how many years he/she is having bank account
Balance	Numerical	The account balance of the customer
NumOfProducts	Numerical	Number of products from bank
HasCrCard	Binary (Yes or No)	Whether if the customer owns a credit card at the bank
IsActiveMember	Binary (Yes or No)	Is the customer an active member of the bank
EstimatedSalary	Numerical	Estimated salary of the customer
Geography	Categorical	Country of residence
Exited	Binary (Yes or No)	Whether the customer has stayed or left the bank. (i.e. Account closed or not)

Code Snippets:

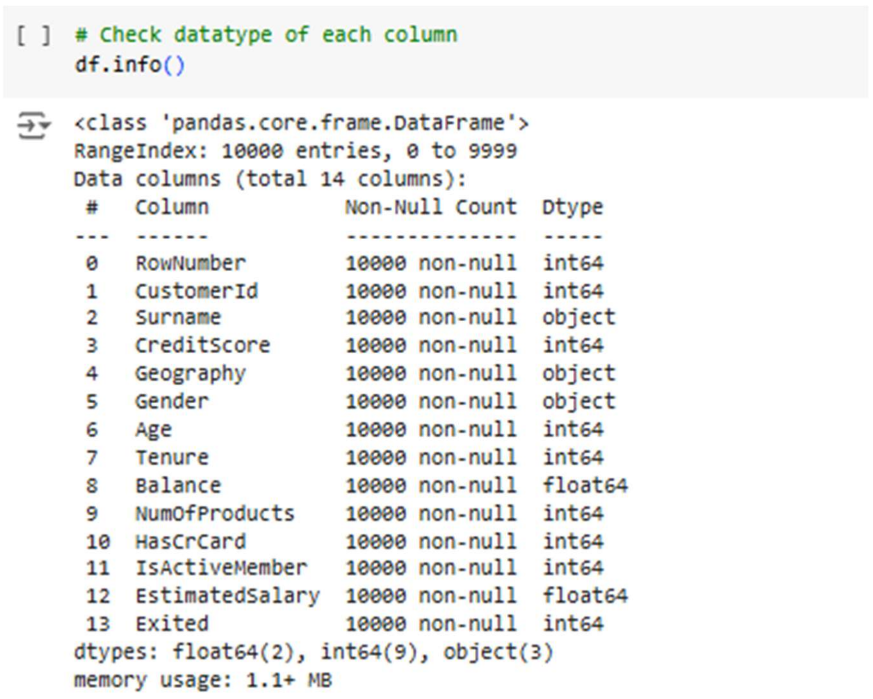
1. Dimension of Dataset.



```
[ ] # Check dimension of dataset
    print('The dimension of dataset is (row, column):')
    df.shape
```

```
The dimension of dataset is (row, column):
(10000, 14)
```

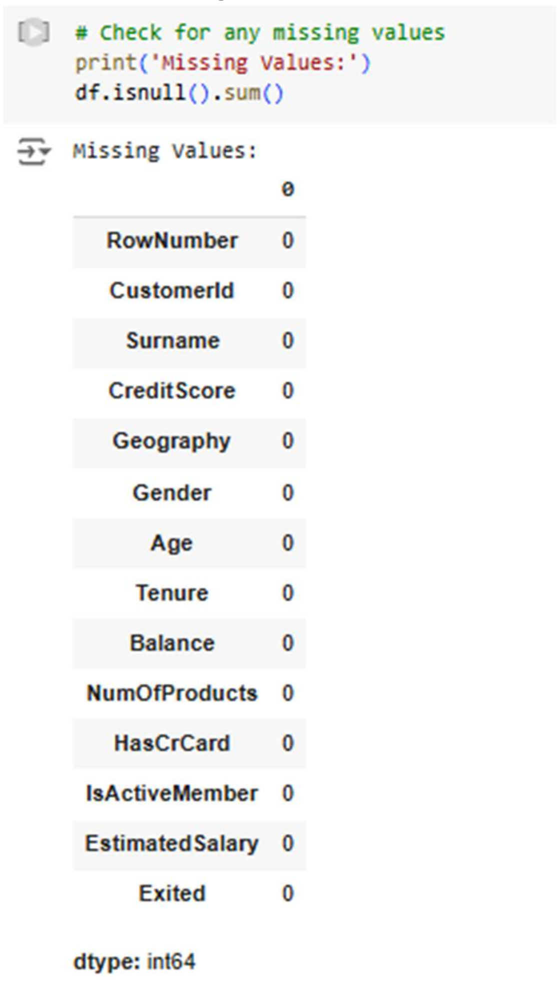
2. Data Type of each Column.



```
[ ] # Check datatype of each column
    df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   RowNumber        10000 non-null  int64  
 1   CustomerId       10000 non-null  int64  
 2   Surname          10000 non-null  object 
 3   CreditScore      10000 non-null  int64  
 4   Geography        10000 non-null  object 
 5   Gender           10000 non-null  object 
 6   Age              10000 non-null  int64  
 7   Tenure           10000 non-null  int64  
 8   Balance          10000 non-null  float64
 9   NumOfProducts    10000 non-null  int64  
 10  HasCrCard        10000 non-null  int64  
 11  IsActiveMember   10000 non-null  int64  
 12  EstimatedSalary  10000 non-null  float64
 13  Exited           10000 non-null  int64  
dtypes: float64(2), int64(9), object(3)
memory usage: 1.1+ MB
```

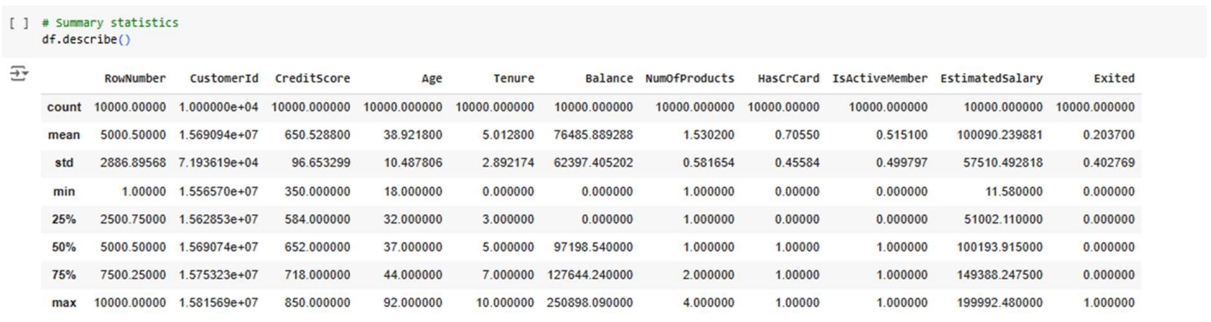
3. Check for Missing Values.



```
# Check for any missing values
print('Missing Values:')
df.isnull().sum()
```

```
Missing Values:
                 0
RowNumber        0
CustomerId       0
Surname          0
CreditScore      0
Geography        0
Gender           0
Age              0
Tenure           0
Balance          0
NumOfProducts    0
HasCrCard        0
IsActiveMember   0
EstimatedSalary  0
Exited           0

dtype: int64
```

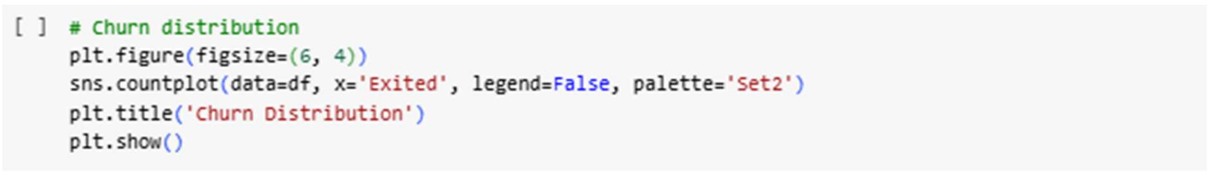
4. Summary Statistics.



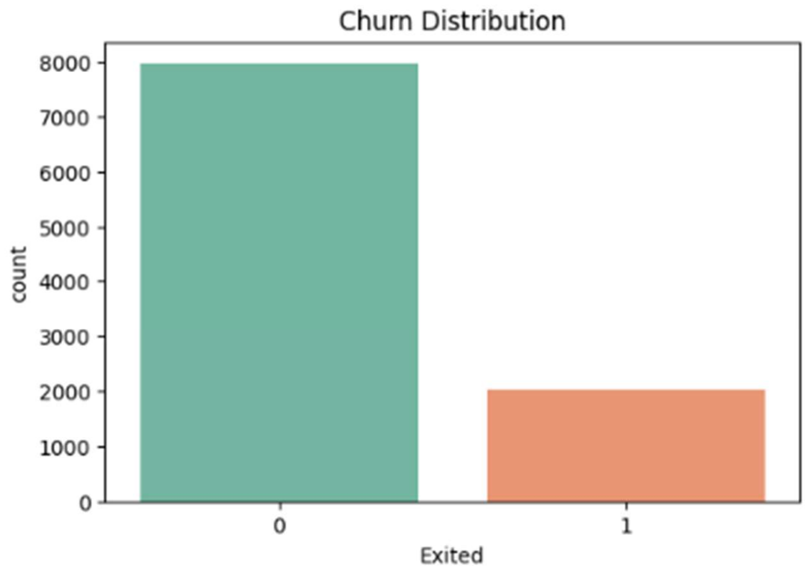
```
[ ] # Summary statistics
    df.describe()
```

```
       RowNumber    CustomerId   CreditScore           Age        Tenure        Balance  NumOfProducts    HasCrCard  IsActiveMember  EstimatedSalary        Exited
count  10000.00000  1.000000e+04  10000.000000  10000.000000  10000.000000   10000.000000   10000.000000  10000.00000    10000.000000     10000.000000  10000.000000
mean    5000.50000  1.569094e+07    650.528800     38.921800      5.012800   76485.889288       1.530200      0.70550        0.515100    100090.239881      0.203700
std     2886.89568  7.193619e+04     96.653299     10.487806      2.892174   62397.405202       0.581654      0.45584        0.499797     57510.492818      0.402769
min        1.00000  1.556570e+07    350.000000     18.000000      0.000000       0.000000       1.000000      0.00000        0.000000        11.580000      0.000000
25%     2500.75000  1.562853e+07    584.000000     32.000000      3.000000       0.000000       1.000000      0.00000        0.000000     51002.110000      0.000000
50%     5000.50000  1.569074e+07    652.000000     37.000000      5.000000   97198.540000       1.000000      1.00000        1.000000    100193.915000      0.000000
75%     7500.25000  1.575323e+07    718.000000     44.000000      7.000000  127644.240000       2.000000      1.00000        1.000000    149388.247500      0.000000
max    10000.00000  1.581569e+07    850.000000     92.000000     10.000000  250898.090000       4.000000      1.00000        1.000000    199992.480000      1.000000
```

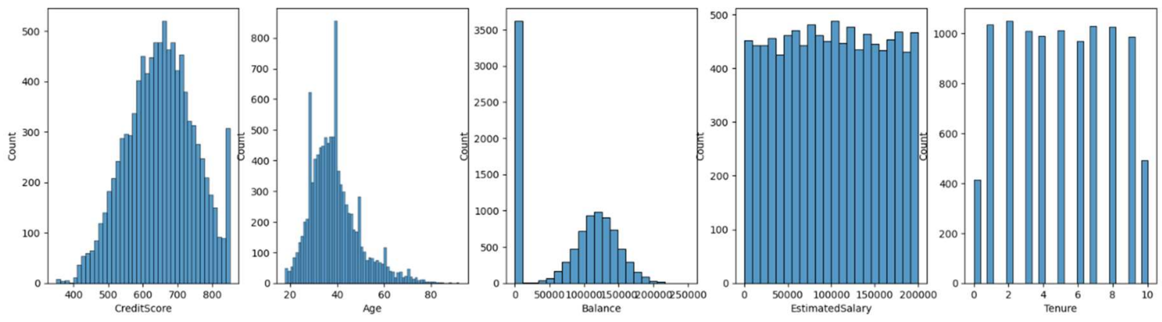
5. Univariate Analysis.



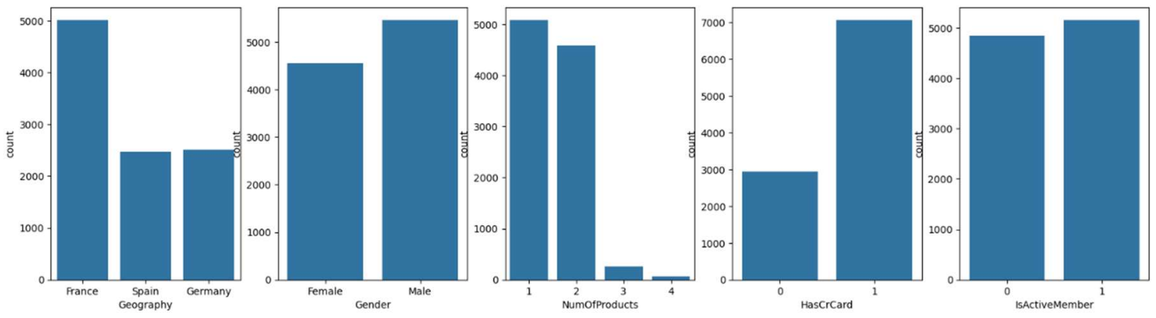
```
[ ] # Churn distribution
    plt.figure(figsize=(6, 4))
    sns.countplot(data=df, x='Exited', legend=False, palette='Set2')
    plt.title('Churn Distribution')
    plt.show()
```



```python
# Histograms for numeric features
numeric_features = ['CreditScore', 'Age', 'Balance', 'EstimatedSalary', 'Tenure']
fig, ax = plt.subplots(1,len(numeric_features),figsize=(20,5))
for i, feature in enumerate(numeric_features):
  sns.histplot(data=df, x=feature, ax=ax[i])
```

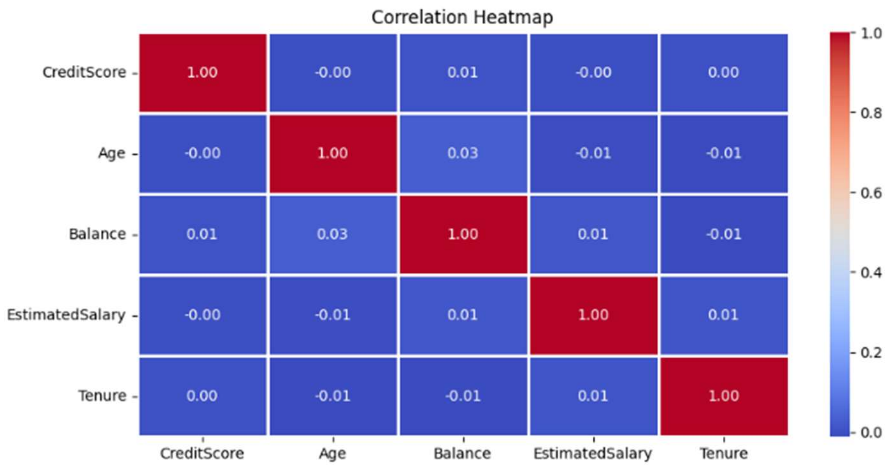


```python
# Bar plots for categorical features
categorical_features = ['Geography', 'Gender', 'NumOfProducts', 'HasCrCard', 'IsActiveMember']
fig, ax = plt.subplots(1,len(numeric_features),figsize=(20,5))
for i, feature in enumerate(categorical_features):
  sns.countplot(data=df, x=feature, ax=ax[i])
```

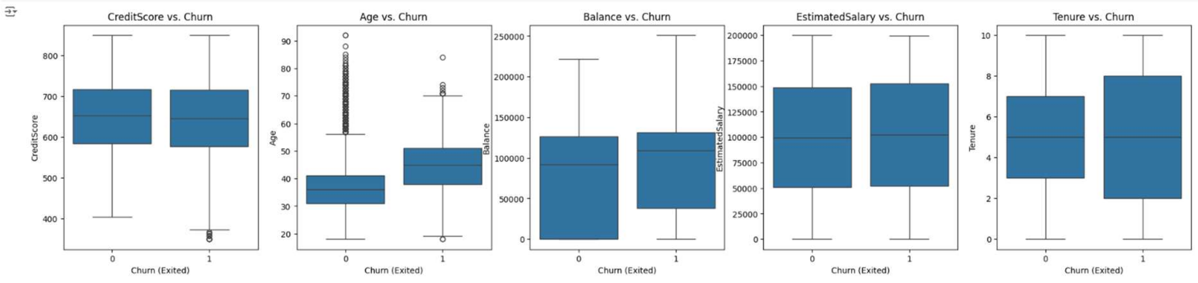


6. Bivariate Analysis.

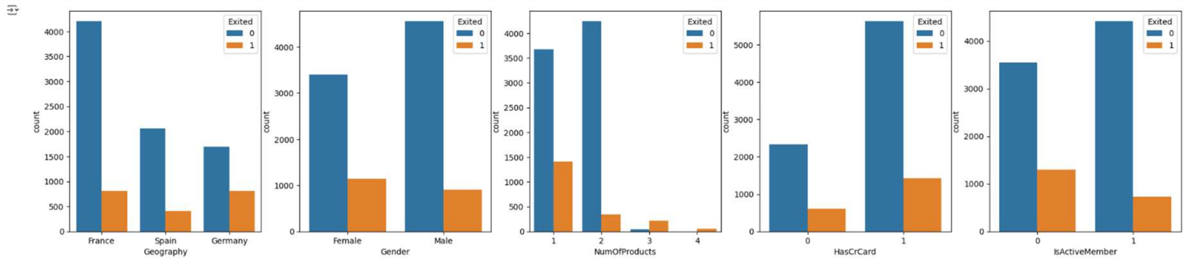
```python
# Correlation Matrix
plt.figure(figsize=(10, 5))
sns.heatmap(df[['CreditScore', 'Age', 'Balance', 'EstimatedSalary', 'Tenure']].corr(), annot=True, fmt='.2f', cmap='coolwarm', linewidths=2)
plt.title('Correlation Heatmap')
plt.show()
```



```
[ ] # Boxplot of numeric features vs. Churn
    fig, ax = plt.subplots(1,len(numeric_features),figsize=(25,5))
    for i, feature in enumerate(numeric_features):
        sns.boxplot(data=df, x='Exited', y=feature, ax=ax[i])
        ax[i].set_title(f'{feature} vs. Churn')
        ax[i].set_xlabel('Churn (Exited)')
        ax[i].set_ylabel(feature)
```

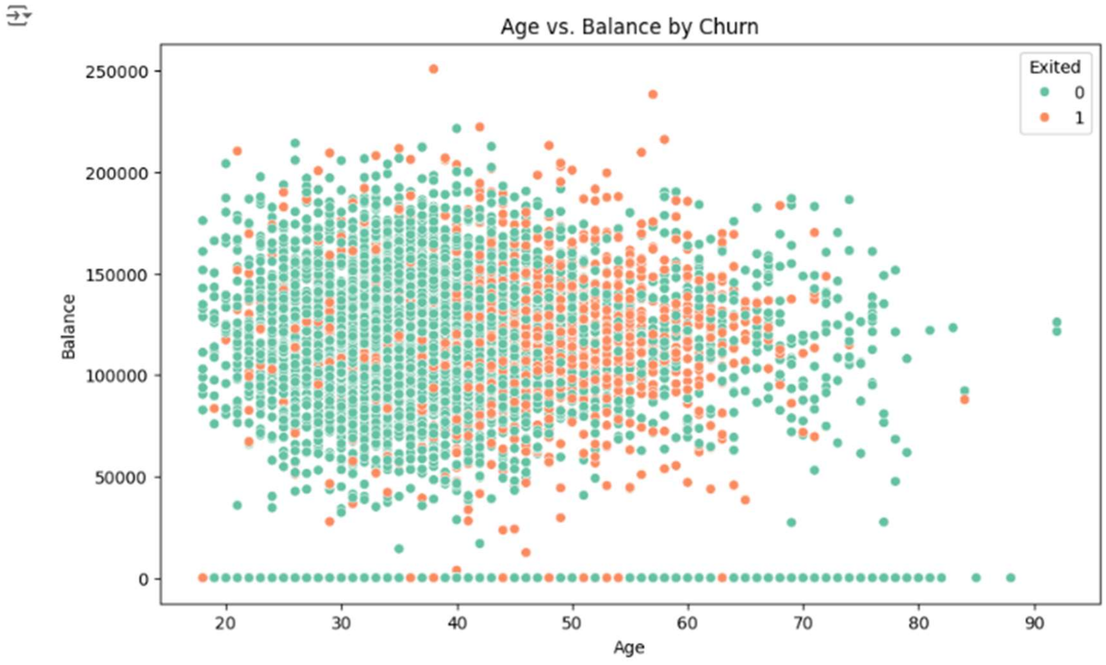


```
[ ] # Churn rate by categorical features
    fig, ax = plt.subplots(1,len(categorical_features),figsize=(25,5))
    for i, feature in enumerate(categorical_features):
        sns.countplot(data=df, x=feature, hue='Exited', ax=ax[i])
        ax[i].set_xlabel(feature)
```



7. Multivariate Analysis.

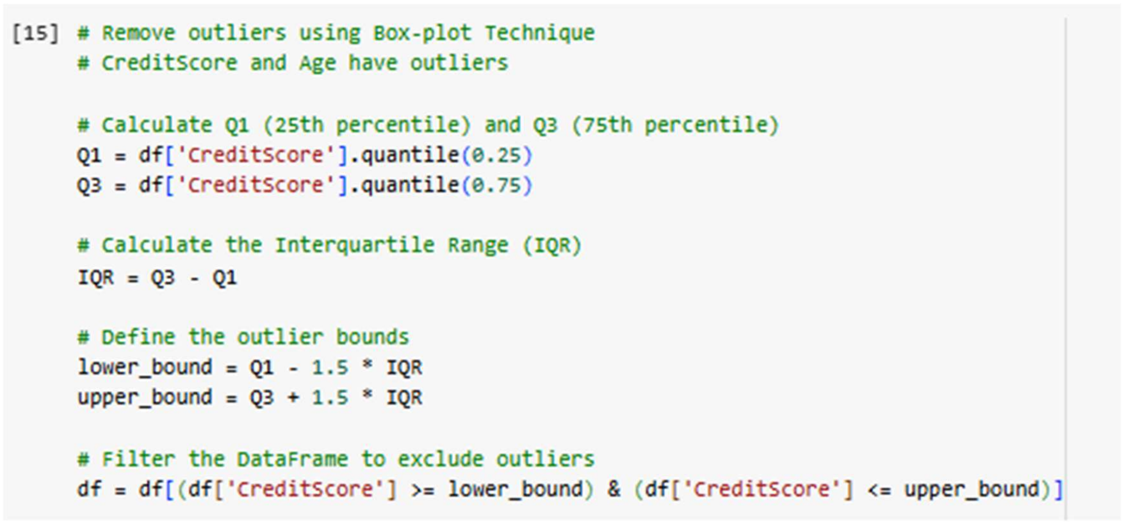
```
[ ] # Scatter plot of Age vs. Balance colored by Churn
    plt.figure(figsize=(10, 6))
    sns.scatterplot(data=df, x='Age', y='Balance', hue='Exited', palette='Set2')
    plt.title('Age vs. Balance by Churn')
    plt.show()
```



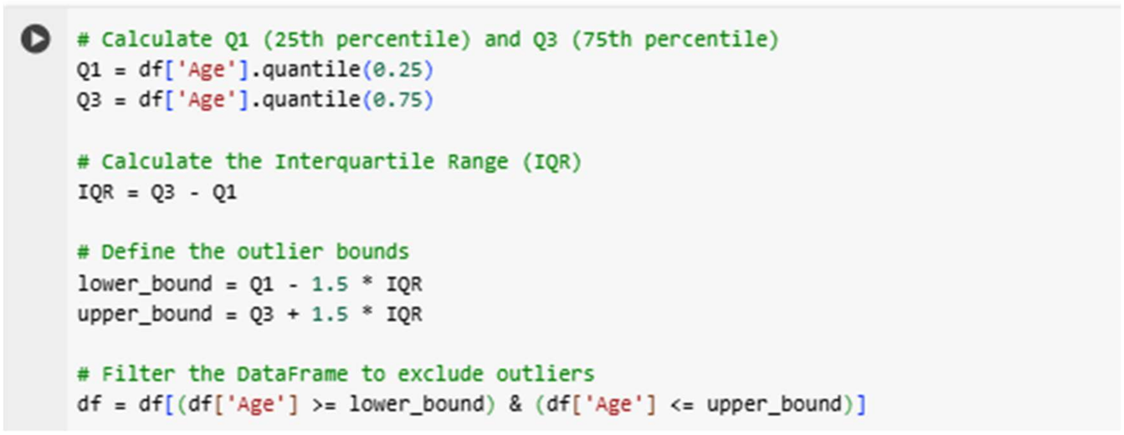
8. Removing Unnecessary columns.

```
[14] # Drop unnecessary columns
     df.drop(columns=['RowNumber', 'CustomerId', 'Surname'], inplace=True)
```

9. Removing outliers.

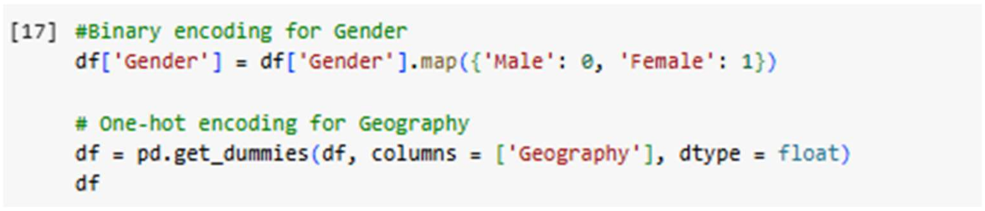


```
[15] # Remove outliers using Box-plot Technique
     # CreditScore and Age have outliers

     # Calculate Q1 (25th percentile) and Q3 (75th percentile)
     Q1 = df['CreditScore'].quantile(0.25)
     Q3 = df['CreditScore'].quantile(0.75)

     # Calculate the Interquartile Range (IQR)
     IQR = Q3 - Q1

     # Define the outlier bounds
     lower_bound = Q1 - 1.5 * IQR
     upper_bound = Q3 + 1.5 * IQR

     # Filter the DataFrame to exclude outliers
     df = df[(df['CreditScore'] >= lower_bound) & (df['CreditScore'] <= upper_bound)]
```



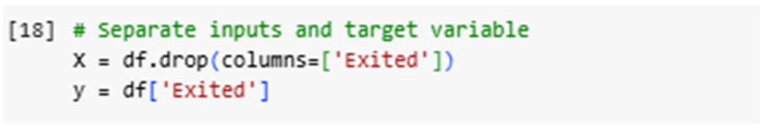
```
# Calculate Q1 (25th percentile) and Q3 (75th percentile)
Q1 = df['Age'].quantile(0.25)
Q3 = df['Age'].quantile(0.75)

# Calculate the Interquartile Range (IQR)
IQR = Q3 - Q1

# Define the outlier bounds
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Filter the DataFrame to exclude outliers
df = df[(df['Age'] >= lower_bound) & (df['Age'] <= upper_bound)]
```

10. Encoding Categorical variables.



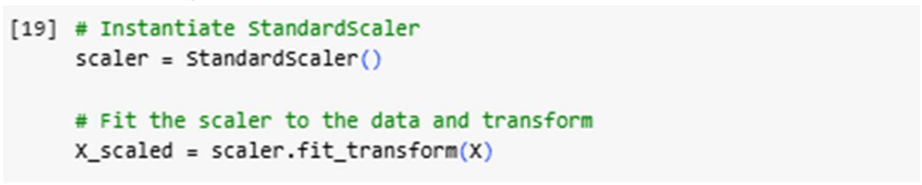
```
[17] #Binary encoding for Gender
     df['Gender'] = df['Gender'].map({'Male': 0, 'Female': 1})

     # One-hot encoding for Geography
     df = pd.get_dummies(df, columns = ['Geography'], dtype = float)
     df
```

11. Declaring Inputs and Target variable.



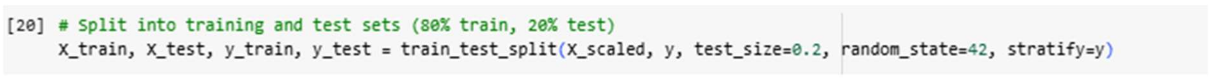
```
[18] # Separate inputs and target variable
     X = df.drop(columns=['Exited'])
     y = df['Exited']
```

12. Feature Scaling.



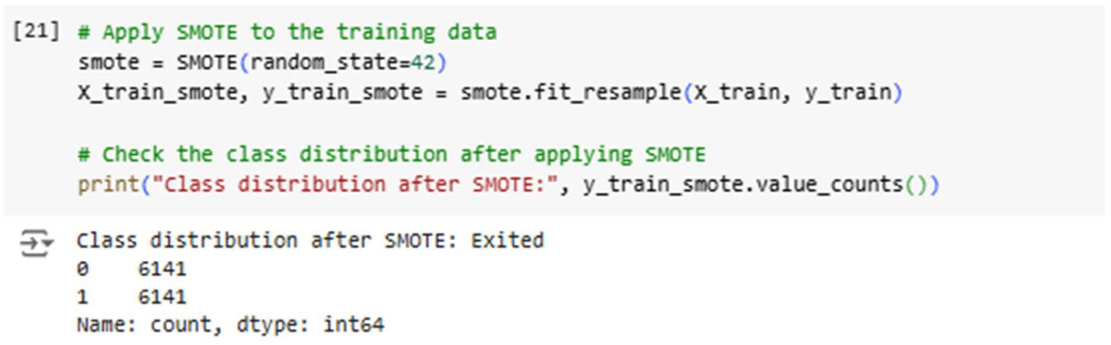
```
[19] # Instantiate StandardScaler
     scaler = StandardScaler()

     # Fit the scaler to the data and transform
     X_scaled = scaler.fit_transform(X)
```

13. Training and Testing Split.



```
[20] # Split into training and test sets (80% train, 20% test)
     X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42, stratify=y)
```

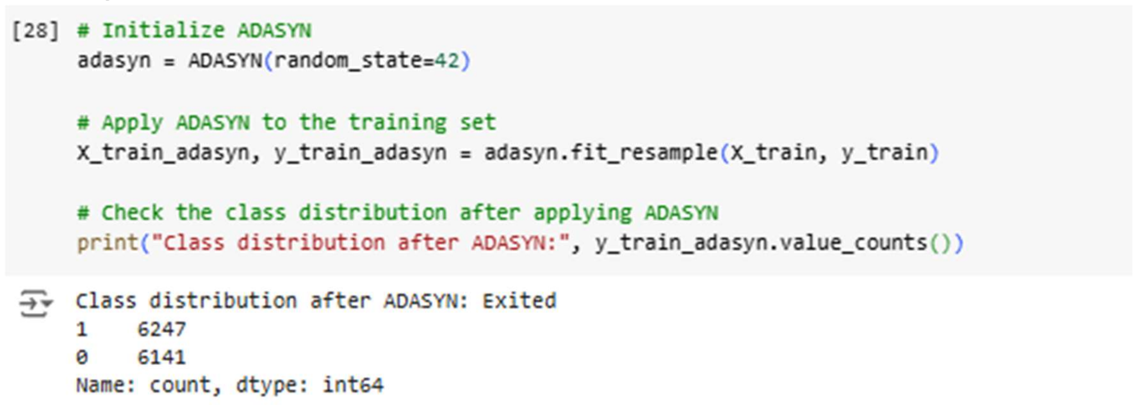
14. Applying SMOTE.



```
[21] # Apply SMOTE to the training data
     smote = SMOTE(random_state=42)
     X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)

     # Check the class distribution after applying SMOTE
     print("Class distribution after SMOTE:", y_train_smote.value_counts())
```

```
Class distribution after SMOTE: Exited
0    6141
1    6141
Name: count, dtype: int64
```

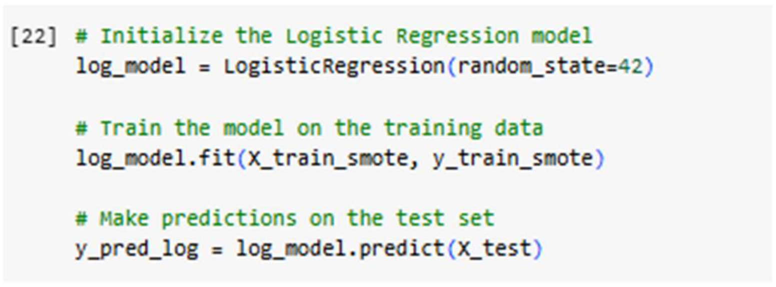
15. Applying ADASYN.



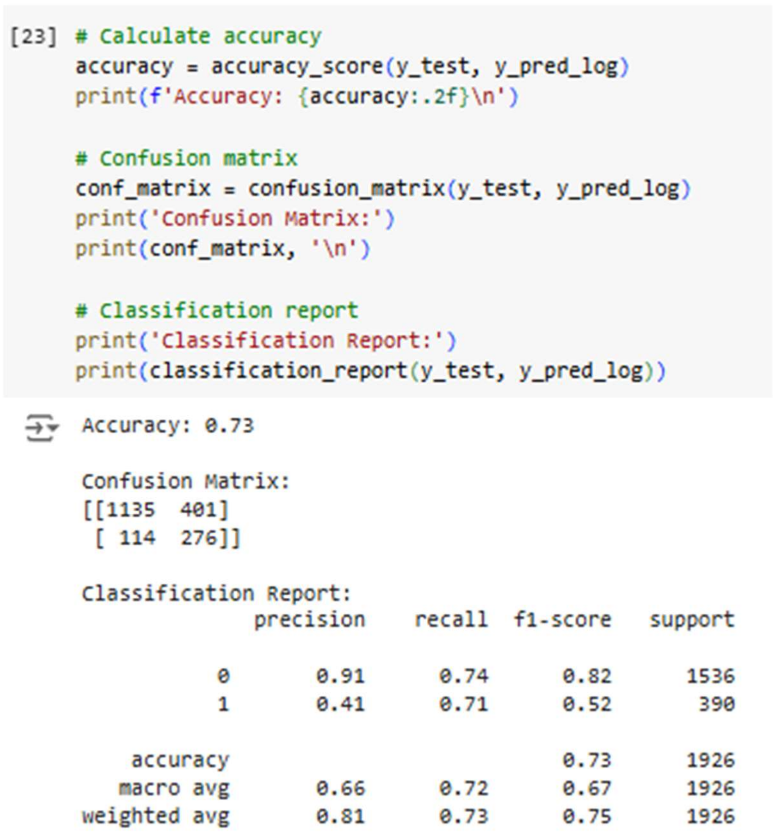
```
[28] # Initialize ADASYN
     adasyn = ADASYN(random_state=42)

     # Apply ADASYN to the training set
     X_train_adasyn, y_train_adasyn = adasyn.fit_resample(X_train, y_train)

     # Check the class distribution after applying ADASYN
     print("Class distribution after ADASYN:", y_train_adasyn.value_counts())
```

```
Class distribution after ADASYN: Exited
1    6247
0    6141
Name: count, dtype: int64
```

16. Logistic Regression Model.



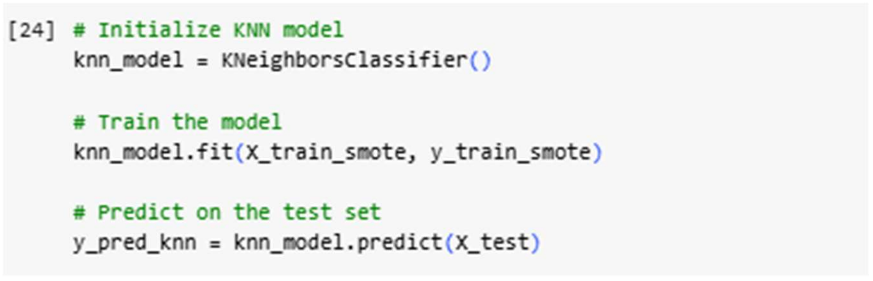
```
[22] # Initialize the Logistic Regression model
     log_model = LogisticRegression(random_state=42)

     # Train the model on the training data
     log_model.fit(X_train_smote, y_train_smote)

     # Make predictions on the test set
     y_pred_log = log_model.predict(X_test)
```



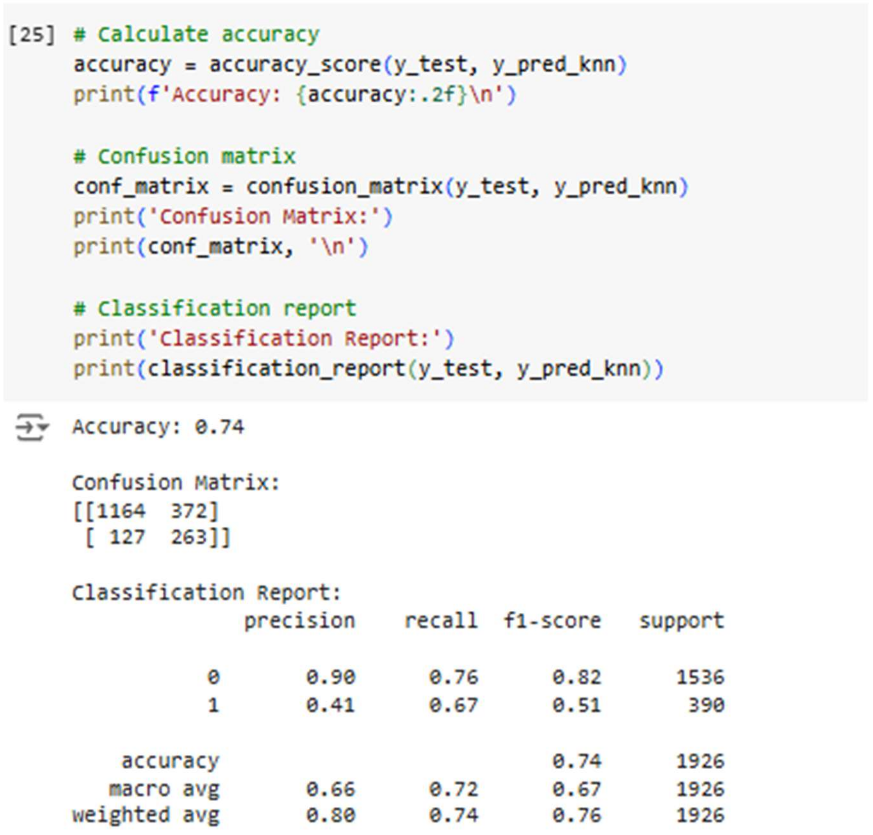
```
[23] # Calculate accuracy
     accuracy = accuracy_score(y_test, y_pred_log)
     print(f'Accuracy: {accuracy:.2f}\n')

     # Confusion matrix
     conf_matrix = confusion_matrix(y_test, y_pred_log)
     print('Confusion Matrix:')
     print(conf_matrix, '\n')

     # Classification report
     print('Classification Report:')
     print(classification_report(y_test, y_pred_log))
```

```
Accuracy: 0.73

Confusion Matrix:
[[1135  401]
 [ 114  276]]

Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.74      0.82      1536
           1       0.41      0.71      0.52       390

    accuracy                           0.73      1926
   macro avg       0.66      0.72      0.67      1926
weighted avg       0.81      0.73      0.75      1926
```

17. K-Nearest Neighbour Model.



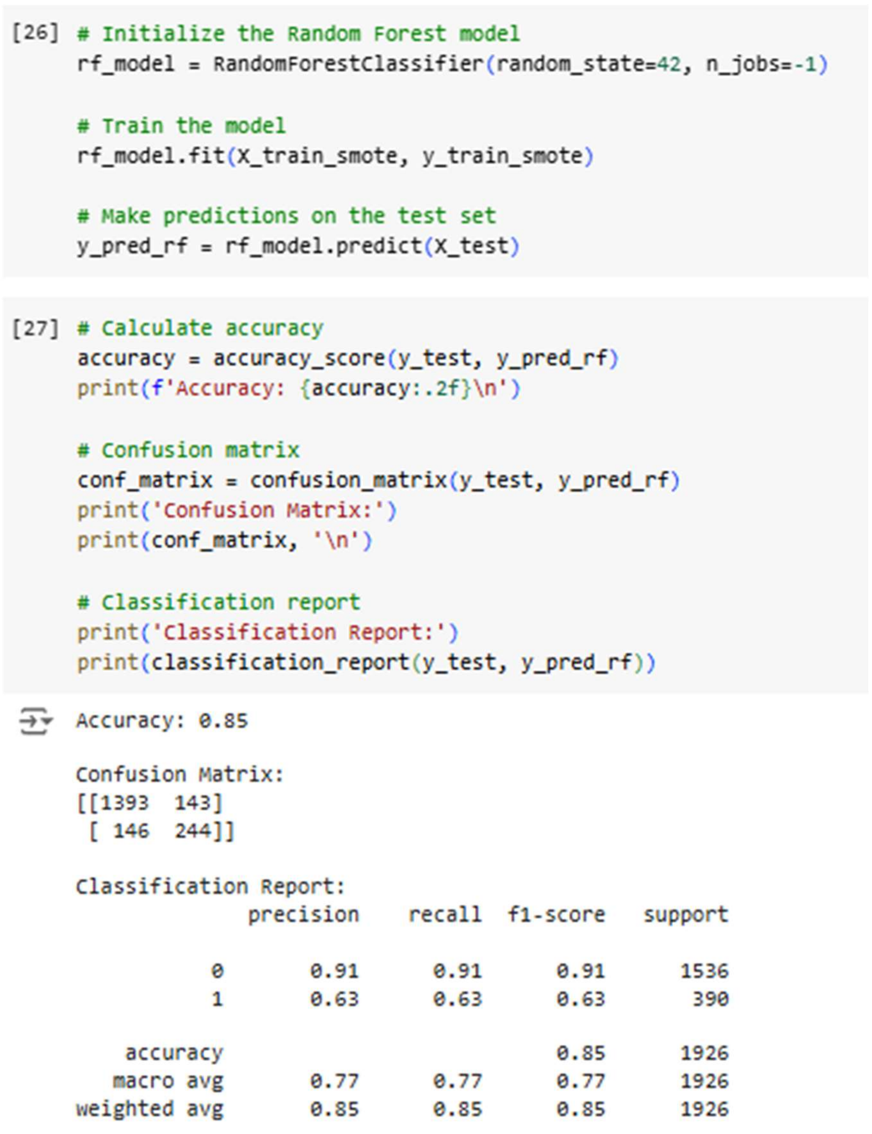
```
[24] # Initialize KNN model
     knn_model = KNeighborsClassifier()

     # Train the model
     knn_model.fit(X_train_smote, y_train_smote)

     # Predict on the test set
     y_pred_knn = knn_model.predict(X_test)
```



```
[25] # Calculate accuracy
     accuracy = accuracy_score(y_test, y_pred_knn)
     print(f'Accuracy: {accuracy:.2f}\n')

     # Confusion matrix
     conf_matrix = confusion_matrix(y_test, y_pred_knn)
     print('Confusion Matrix:')
     print(conf_matrix, '\n')

     # Classification report
     print('Classification Report:')
     print(classification_report(y_test, y_pred_knn))
```

```
Accuracy: 0.74

Confusion Matrix:
[[1164  372]
 [ 127  263]]

Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.76      0.82      1536
           1       0.41      0.67      0.51       390

    accuracy                           0.74      1926
   macro avg       0.66      0.72      0.67      1926
weighted avg       0.80      0.74      0.76      1926
```

18. Random Forest Model.

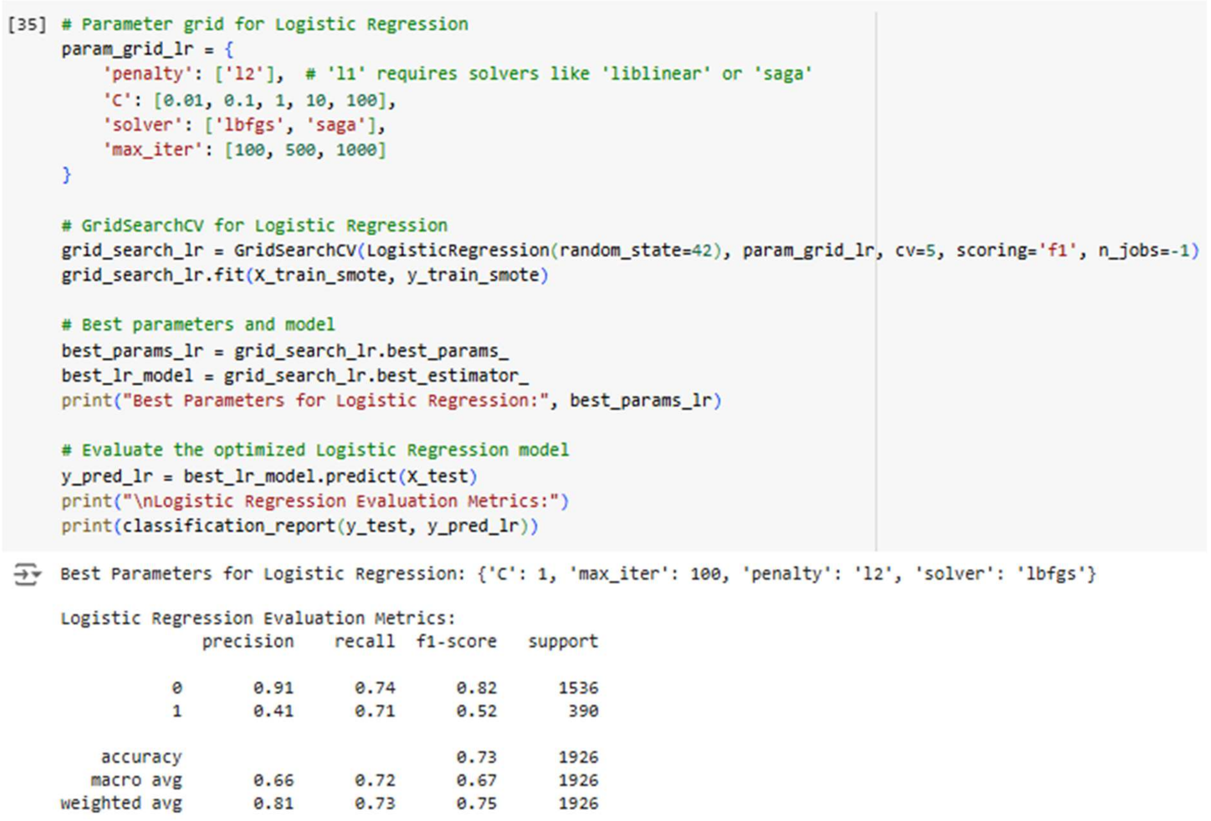


```
[26] # Initialize the Random Forest model
     rf_model = RandomForestClassifier(random_state=42, n_jobs=-1)

     # Train the model
     rf_model.fit(X_train_smote, y_train_smote)

     # Make predictions on the test set
     y_pred_rf = rf_model.predict(X_test)
```

```
[27] # Calculate accuracy
     accuracy = accuracy_score(y_test, y_pred_rf)
     print(f'Accuracy: {accuracy:.2f}\n')

     # Confusion matrix
     conf_matrix = confusion_matrix(y_test, y_pred_rf)
     print('Confusion Matrix:')
     print(conf_matrix, '\n')

     # Classification report
     print('Classification Report:')
     print(classification_report(y_test, y_pred_rf))
```

```
Accuracy: 0.85

Confusion Matrix:
[[1393  143]
 [ 146  244]]

Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.91      0.91      1536
           1       0.63      0.63      0.63       390

    accuracy                           0.85      1926
   macro avg       0.77      0.77      0.77      1926
weighted avg       0.85      0.85      0.85      1926
```

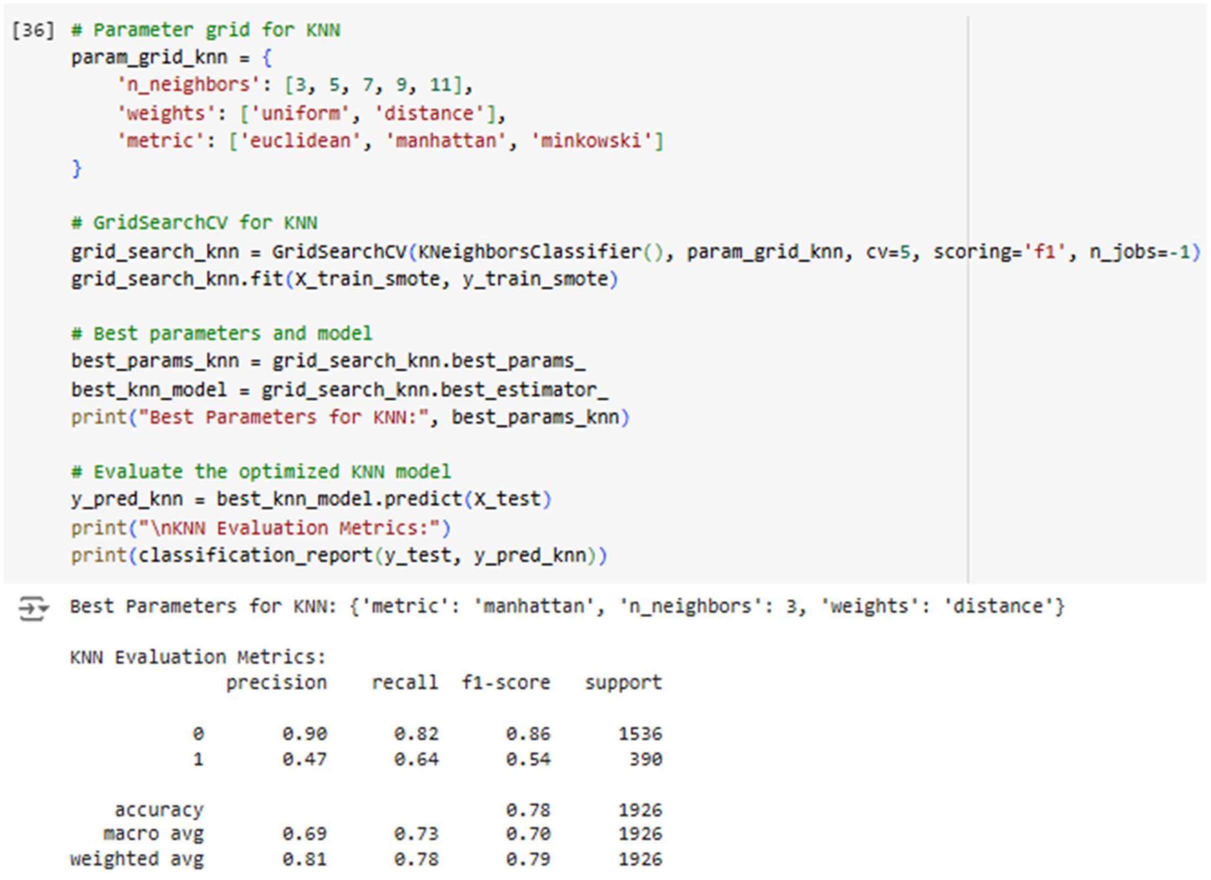
19. Logistic Regression Model (Hyperparameter Tuning).



```
[35] # Parameter grid for Logistic Regression
     param_grid_lr = {
         'penalty': ['l2'],  # 'l1' requires solvers like 'liblinear' or 'saga'
         'C': [0.01, 0.1, 1, 10, 100],
         'solver': ['lbfgs', 'saga'],
         'max_iter': [100, 500, 1000]
     }

     # GridSearchCV for Logistic Regression
     grid_search_lr = GridSearchCV(LogisticRegression(random_state=42), param_grid_lr, cv=5, scoring='f1', n_jobs=-1)
     grid_search_lr.fit(X_train_smote, y_train_smote)

     # Best parameters and model
     best_params_lr = grid_search_lr.best_params_
     best_lr_model = grid_search_lr.best_estimator_
     print("Best Parameters for Logistic Regression:", best_params_lr)

     # Evaluate the optimized Logistic Regression model
     y_pred_lr = best_lr_model.predict(X_test)
     print("\nLogistic Regression Evaluation Metrics:")
     print(classification_report(y_test, y_pred_lr))
```

```
Best Parameters for Logistic Regression: {'C': 1, 'max_iter': 100, 'penalty': 'l2', 'solver': 'lbfgs'}

Logistic Regression Evaluation Metrics:
              precision    recall  f1-score   support

           0       0.91      0.74      0.82      1536
           1       0.41      0.71      0.52       390

    accuracy                           0.73      1926
   macro avg       0.66      0.72      0.67      1926
weighted avg       0.81      0.73      0.75      1926
```

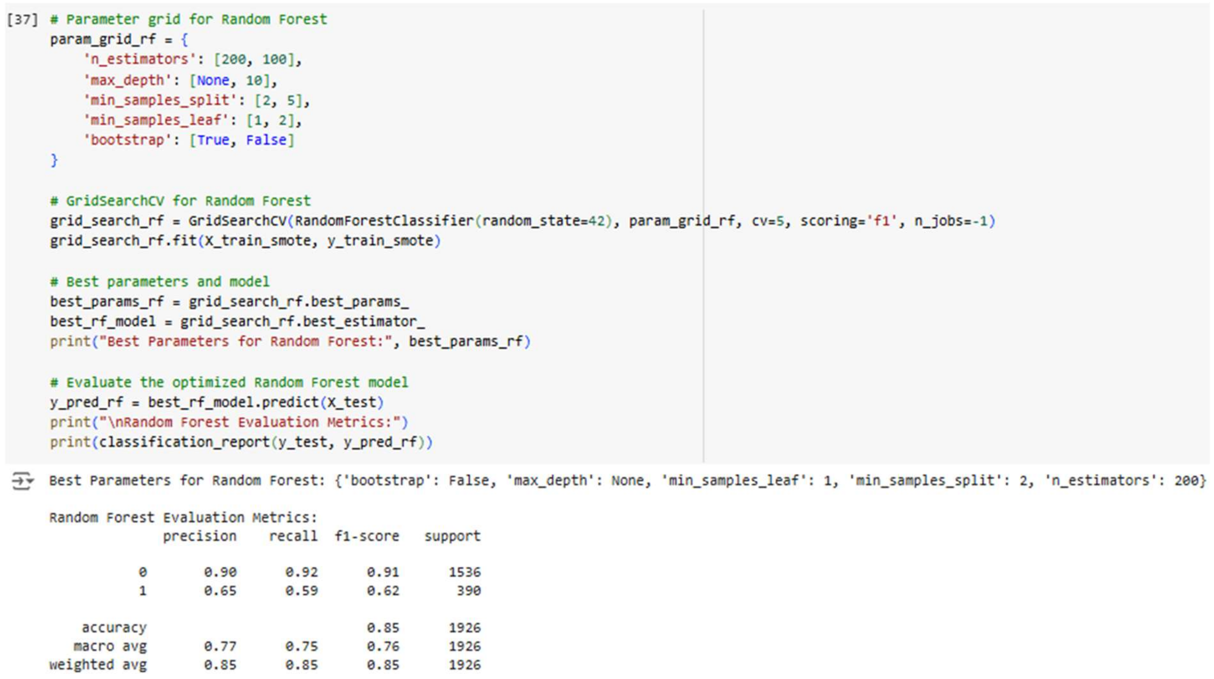
20. KNN (Hyperparameter Tuning).



```
[36] # Parameter grid for KNN
     param_grid_knn = {
         'n_neighbors': [3, 5, 7, 9, 11],
         'weights': ['uniform', 'distance'],
         'metric': ['euclidean', 'manhattan', 'minkowski']
     }

     # GridSearchCV for KNN
     grid_search_knn = GridSearchCV(KNeighborsClassifier(), param_grid_knn, cv=5, scoring='f1', n_jobs=-1)
     grid_search_knn.fit(X_train_smote, y_train_smote)

     # Best parameters and model
     best_params_knn = grid_search_knn.best_params_
     best_knn_model = grid_search_knn.best_estimator_
     print("Best Parameters for KNN:", best_params_knn)

     # Evaluate the optimized KNN model
     y_pred_knn = best_knn_model.predict(X_test)
     print("\nKNN Evaluation Metrics:")
     print(classification_report(y_test, y_pred_knn))
```

```
Best Parameters for KNN: {'metric': 'manhattan', 'n_neighbors': 3, 'weights': 'distance'}

KNN Evaluation Metrics:
              precision    recall  f1-score   support

           0       0.90      0.82      0.86      1536
           1       0.47      0.64      0.54       390

    accuracy                           0.78      1926
   macro avg       0.69      0.73      0.70      1926
weighted avg       0.81      0.78      0.79      1926
```

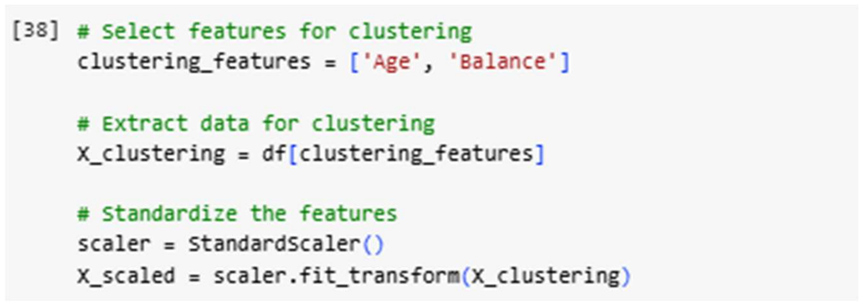
21. Random Forest (Hyperparameter Tuning).



```
[37] # Parameter grid for Random Forest
     param_grid_rf = {
         'n_estimators': [200, 100],
         'max_depth': [None, 10],
         'min_samples_split': [2, 5],
         'min_samples_leaf': [1, 2],
         'bootstrap': [True, False]
     }

     # GridSearchCV for Random Forest
     grid_search_rf = GridSearchCV(RandomForestClassifier(random_state=42), param_grid_rf, cv=5, scoring='f1', n_jobs=-1)
     grid_search_rf.fit(X_train_smote, y_train_smote)

     # Best parameters and model
     best_params_rf = grid_search_rf.best_params_
     best_rf_model = grid_search_rf.best_estimator_
     print("Best Parameters for Random Forest:", best_params_rf)

     # Evaluate the optimized Random Forest model
     y_pred_rf = best_rf_model.predict(X_test)
     print("\nRandom Forest Evaluation Metrics:")
     print(classification_report(y_test, y_pred_rf))
```

```
Best Parameters for Random Forest: {'bootstrap': False, 'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200}

Random Forest Evaluation Metrics:
              precision    recall  f1-score   support

           0       0.90      0.92      0.91      1536
           1       0.65      0.59      0.62       390

    accuracy                           0.85      1926
   macro avg       0.77      0.75      0.76      1926
weighted avg       0.85      0.85      0.85      1926
```

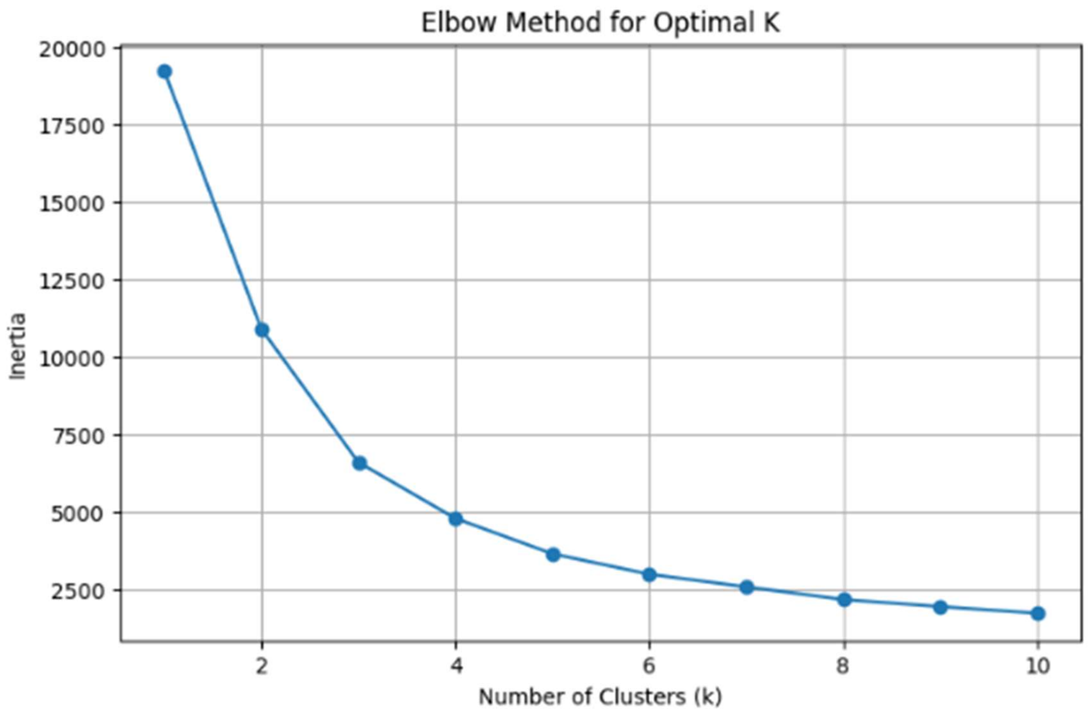
22. Customer Segementation.



```
[38] # Select features for clustering
     clustering_features = ['Age', 'Balance']

     # Extract data for clustering
     X_clustering = df[clustering_features]

     # Standardize the features
     scaler = StandardScaler()
     X_scaled = scaler.fit_transform(X_clustering)
```

23. Elbow method to determine optimal clusters.

```
[39] # Determine the optimal number of clusters using the Elbow Method
     inertia = []
     k_range = range(1, 11)
     for k in k_range:
         kmeans = KMeans(n_clusters=k, random_state=42)
         kmeans.fit(X_scaled)
         inertia.append(kmeans.inertia_)

     # Plot the Elbow Method graph
     plt.figure(figsize=(8, 5))
     plt.plot(k_range, inertia, marker='o')
     plt.title('Elbow Method for Optimal K')
     plt.xlabel('Number of Clusters (k)')
     plt.ylabel('Inertia')
     plt.grid(True)
     plt.show()
```

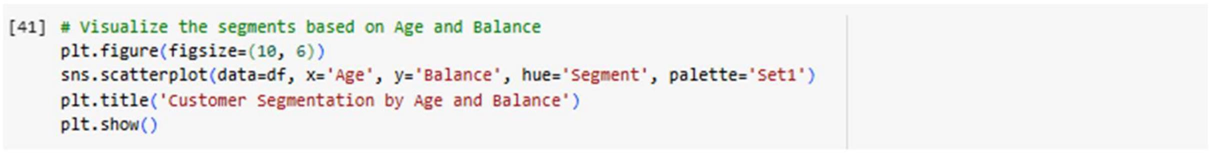


24. K-means Clustering.

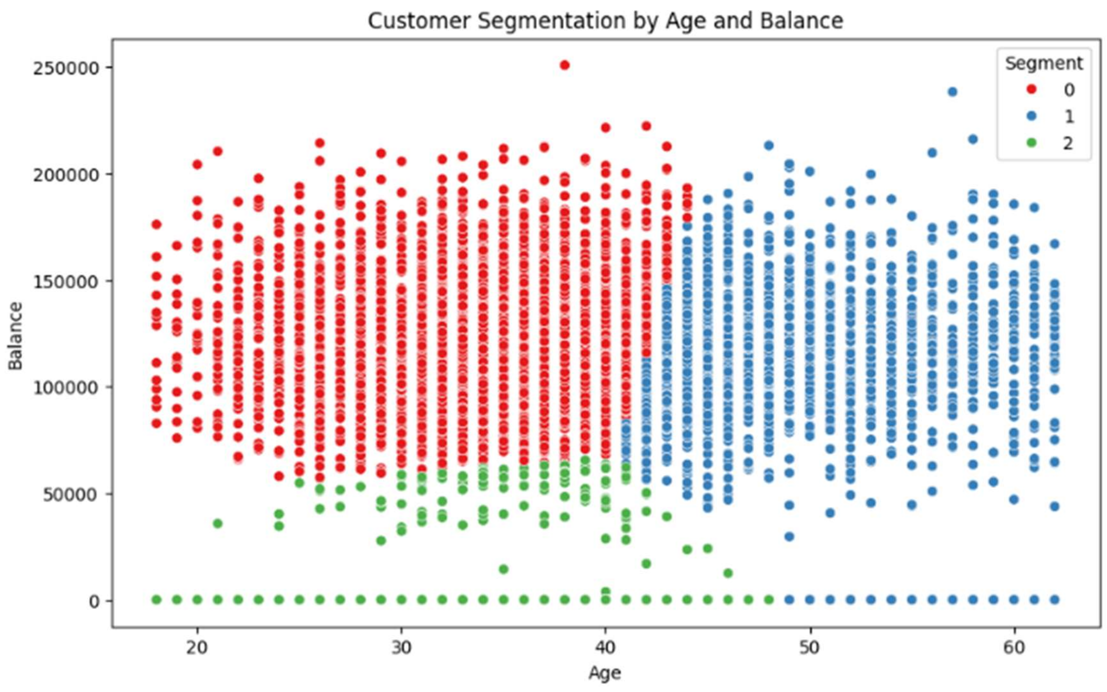
```
[40] # Fit the K-means model with the chosen number of clusters
     kmeans = KMeans(n_clusters=3, random_state=42)
     df['Segment'] = kmeans.fit_predict(X_scaled)

     # Analyze each cluster
     cluster_summary = df.groupby('Segment')[clustering_features].mean()
     print(cluster_summary)
```

```
               Age        Balance
Segment
0        33.584391  123132.189598
1        50.061358   96882.353981
2        35.202657    1909.738672
```

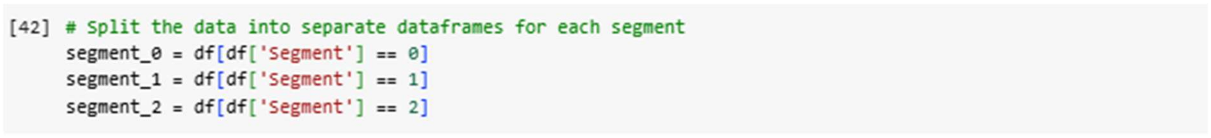
25. Visualize created segments.



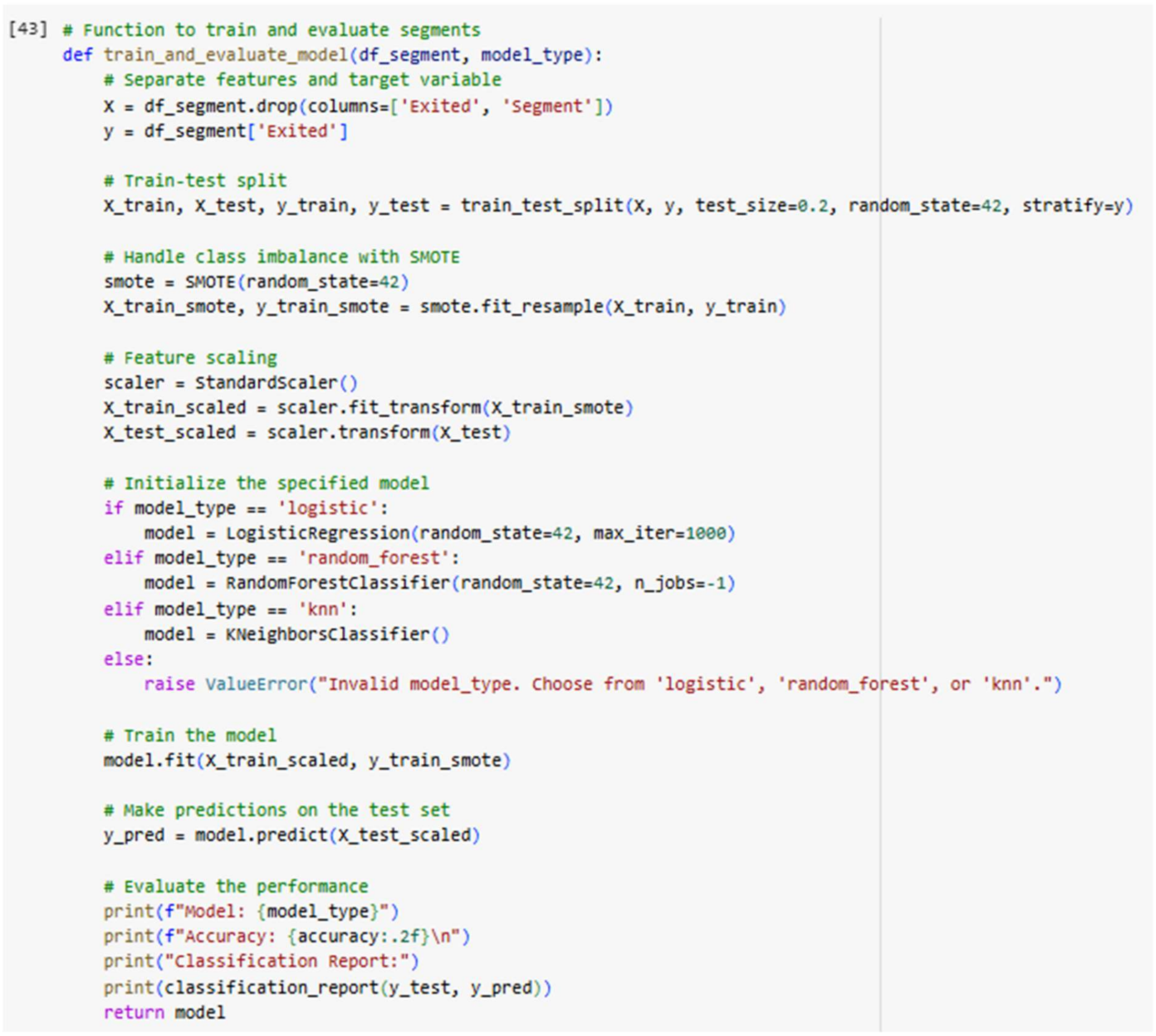
```
[41] # Visualize the segments based on Age and Balance
     plt.figure(figsize=(10, 6))
     sns.scatterplot(data=df, x='Age', y='Balance', hue='Segment', palette='Set1')
     plt.title('Customer Segmentation by Age and Balance')
     plt.show()
```



26. Apply ML models to each segments.

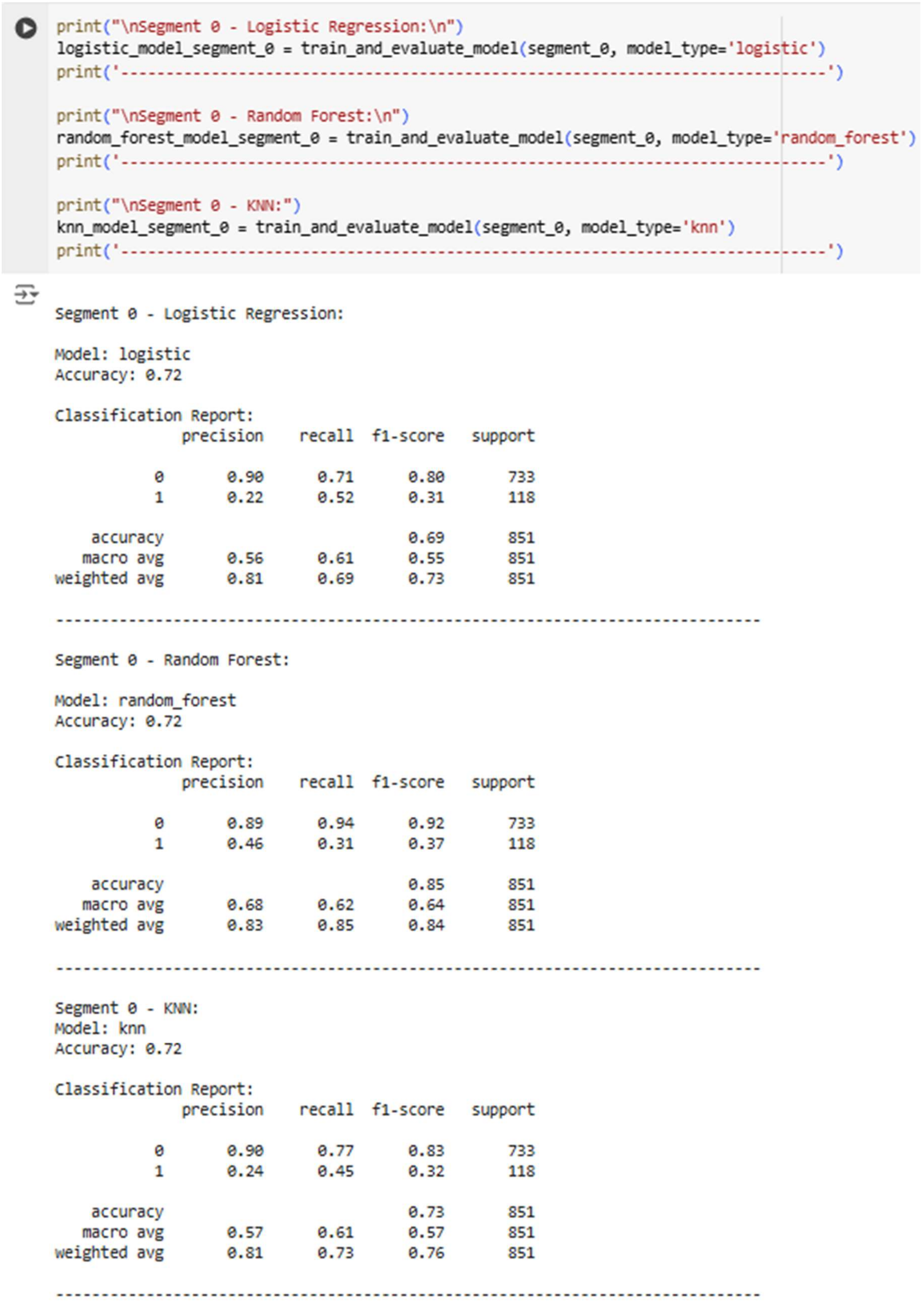


```
[42] # Split the data into separate dataframes for each segment
     segment_0 = df[df['Segment'] == 0]
     segment_1 = df[df['Segment'] == 1]
     segment_2 = df[df['Segment'] == 2]
```



```
[43] # Function to train and evaluate segments
     def train_and_evaluate_model(df_segment, model_type):
         # Separate features and target variable
         X = df_segment.drop(columns=['Exited', 'Segment'])
         y = df_segment['Exited']

         # Train-test split
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

         # Handle class imbalance with SMOTE
         smote = SMOTE(random_state=42)
         X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)

         # Feature scaling
         scaler = StandardScaler()
         X_train_scaled = scaler.fit_transform(X_train_smote)
         X_test_scaled = scaler.transform(X_test)

         # Initialize the specified model
         if model_type == 'logistic':
             model = LogisticRegression(random_state=42, max_iter=1000)
         elif model_type == 'random_forest':
             model = RandomForestClassifier(random_state=42, n_jobs=-1)
         elif model_type == 'knn':
             model = KNeighborsClassifier()
         else:
             raise ValueError("Invalid model_type. Choose from 'logistic', 'random_forest', or 'knn'.")

         # Train the model
         model.fit(X_train_scaled, y_train_smote)

         # Make predictions on the test set
         y_pred = model.predict(X_test_scaled)

         # Evaluate the performance
         print(f"Model: {model_type}")
         print(f"Accuracy: {accuracy:.2f}\n")
         print("Classification Report:")
         print(classification_report(y_test, y_pred))
         return model
```

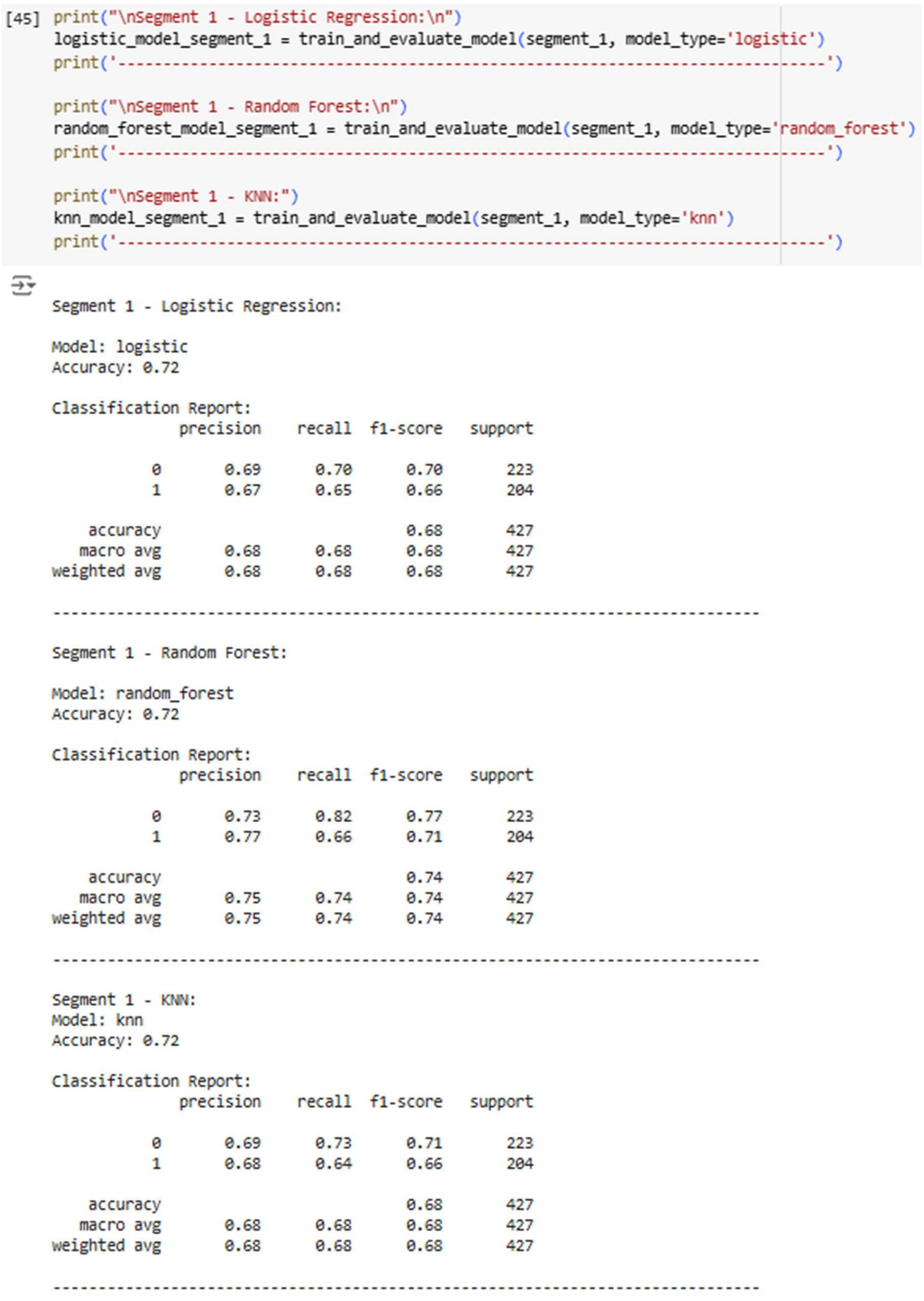
27. Segment 0.



```
print("\nSegment 0 - Logistic Regression:\n")
logistic_model_segment_0 = train_and_evaluate_model(segment_0, model_type='logistic')
print('----------------------------------------------------------------------------')

print("\nSegment 0 - Random Forest:\n")
random_forest_model_segment_0 = train_and_evaluate_model(segment_0, model_type='random_forest')
print('----------------------------------------------------------------------------')

print("\nSegment 0 - KNN:")
knn_model_segment_0 = train_and_evaluate_model(segment_0, model_type='knn')
print('----------------------------------------------------------------------------')
```

```
Segment 0 - Logistic Regression:

Model: logistic
Accuracy: 0.72

Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.71      0.80       733
           1       0.22      0.52      0.31       118

    accuracy                           0.69       851
   macro avg       0.56      0.61      0.55       851
weighted avg       0.81      0.69      0.73       851

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

Segment 0 - Random Forest:

Model: random_forest
Accuracy: 0.72

Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.94      0.92       733
           1       0.46      0.31      0.37       118

    accuracy                           0.85       851
   macro avg       0.68      0.62      0.64       851
weighted avg       0.83      0.85      0.84       851

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

Segment 0 - KNN:
Model: knn
Accuracy: 0.72

Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.77      0.83       733
           1       0.24      0.45      0.32       118

    accuracy                           0.73       851
   macro avg       0.57      0.61      0.57       851
weighted avg       0.81      0.73      0.76       851

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

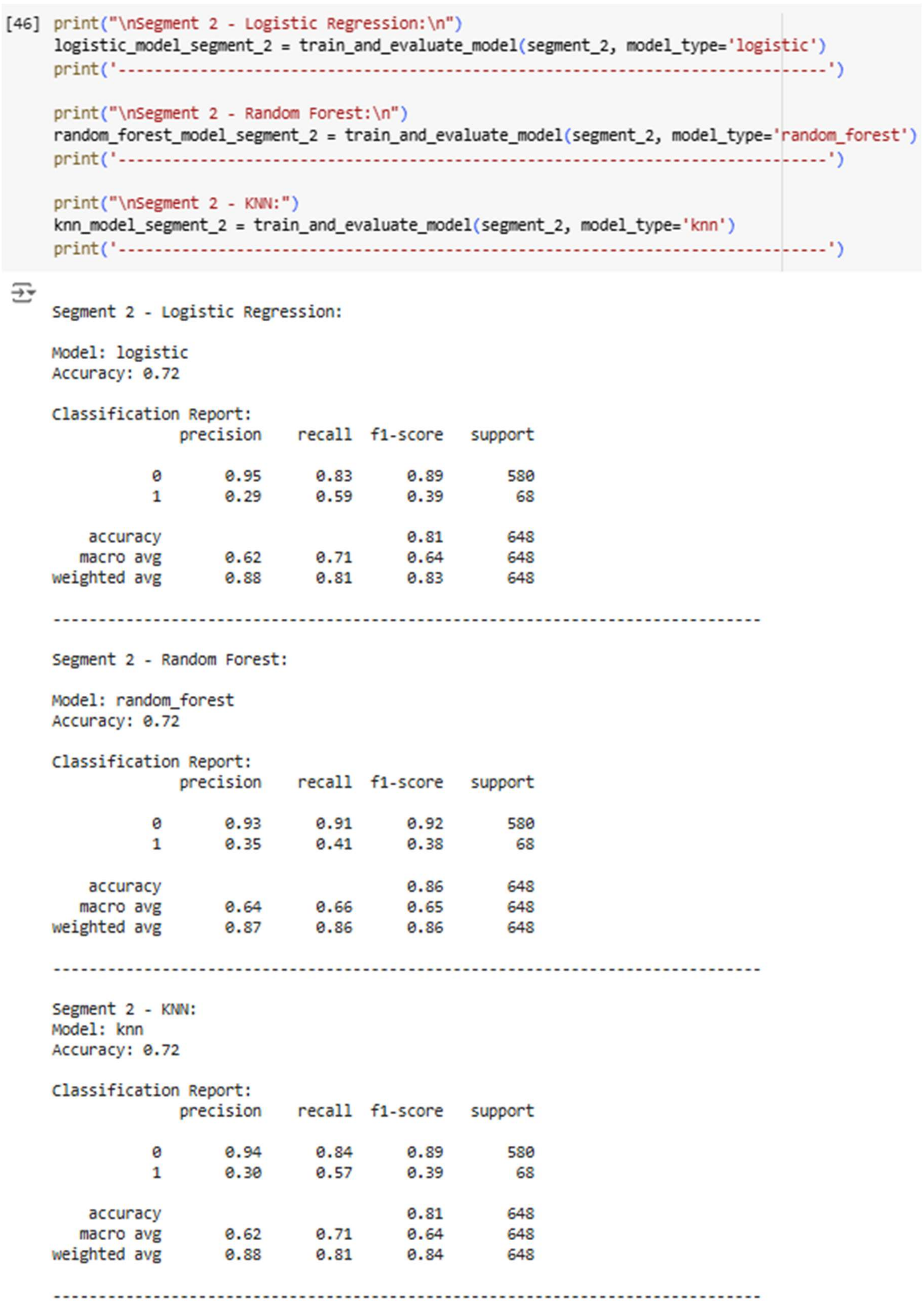
28. Segment 1.



```
[45] print("\nSegment 1 - Logistic Regression:\n")
     logistic_model_segment_1 = train_and_evaluate_model(segment_1, model_type='logistic')
     print('-----------------------------------------------------------------------------')

     print("\nSegment 1 - Random Forest:\n")
     random_forest_model_segment_1 = train_and_evaluate_model(segment_1, model_type='random_forest')
     print('-----------------------------------------------------------------------------')

     print("\nSegment 1 - KNN:")
     knn_model_segment_1 = train_and_evaluate_model(segment_1, model_type='knn')
     print('-----------------------------------------------------------------------------')
```

```
Segment 1 - Logistic Regression:

Model: logistic
Accuracy: 0.72

Classification Report:
              precision    recall  f1-score   support

           0       0.69      0.70      0.70       223
           1       0.67      0.65      0.66       204

    accuracy                           0.68       427
   macro avg       0.68      0.68      0.68       427
weighted avg       0.68      0.68      0.68       427

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

Segment 1 - Random Forest:

Model: random_forest
Accuracy: 0.72

Classification Report:
              precision    recall  f1-score   support

           0       0.73      0.82      0.77       223
           1       0.77      0.66      0.71       204

    accuracy                           0.74       427
   macro avg       0.75      0.74      0.74       427
weighted avg       0.75      0.74      0.74       427

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

Segment 1 - KNN:
Model: knn
Accuracy: 0.72

Classification Report:
              precision    recall  f1-score   support

           0       0.69      0.73      0.71       223
           1       0.68      0.64      0.66       204

    accuracy                           0.68       427
   macro avg       0.68      0.68      0.68       427
weighted avg       0.68      0.68      0.68       427

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

29. Segment 2.



```
[46] print("\nSegment 2 - Logistic Regression:\n")
     logistic_model_segment_2 = train_and_evaluate_model(segment_2, model_type='logistic')
     print('-----------------------------------------------------------------------------')

     print("\nSegment 2 - Random Forest:\n")
     random_forest_model_segment_2 = train_and_evaluate_model(segment_2, model_type='random_forest')
     print('-----------------------------------------------------------------------------')

     print("\nSegment 2 - KNN:")
     knn_model_segment_2 = train_and_evaluate_model(segment_2, model_type='knn')
     print('-----------------------------------------------------------------------------')
```

```
Segment 2 - Logistic Regression:

Model: logistic
Accuracy: 0.72

Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.83      0.89       580
           1       0.29      0.59      0.39        68

    accuracy                           0.81       648
   macro avg       0.62      0.71      0.64       648
weighted avg       0.88      0.81      0.83       648

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

Segment 2 - Random Forest:

Model: random_forest
Accuracy: 0.72

Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.91      0.92       580
           1       0.35      0.41      0.38        68

    accuracy                           0.86       648
   macro avg       0.64      0.66      0.65       648
weighted avg       0.87      0.86      0.86       648

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

Segment 2 - KNN:
Model: knn
Accuracy: 0.72

Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.84      0.89       580
           1       0.30      0.57      0.39        68

    accuracy                           0.81       648
   macro avg       0.62      0.71      0.64       648
weighted avg       0.88      0.81      0.84       648

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