Customer Churn Prediction

Project Report

Group 11
Aravind Swamy
Jayasurya Jagadeesan
Keshika Arunkumar
Samyuktha Kapoor
Sruthi Gandla

swamy.ar@northeastern.edu
jegadeesan.j@northeastern.edu
arunkumar.k@northeastern.edu
rajeshkapoor.s@northeastern.edu
gandla.s@northeastern.edu

Percentage of Effort Contributed by Student 1: 20%

Percentage of Effort Contributed by Student 2: 20%

Percentage of Effort Contributed by Student 3:20%

Percentage of Effort Contributed by Student 4:20%

Percentage of Effort Contributed by Student 5:20%

Signature of Student 1: Aravind Swamy

Signature of Student 2: Jayasurya Jagadeesan

Signature of Student 3: Keshika Arunkumar

Signature of Student 4: Samyuktha Kapoor

Signature of Student 5: Sruthi Gandla

Submission Date: 04/19/2024

Table of Contents

1. Introduction:

This project aims to analyze customer data from a bank to predict customer churn. By understanding the factors that influence customer decisions to leave, we can develop targeted strategies to improve customer retention. The dataset was sourced from Kaggle, which includes comprehensive customer attributes such as credit score, geography, gender, age, tenure, balance, number of products, cardholder status, activity level, estimated salary, and churn status. This analysis provides insights that could help tailor banking services to reduce churn rates.

2. Problem Statement:

Customer churn, or the rate at which customers cease their business relationship with a company, is a critical metric that significantly impacts the company's revenue. In the banking sector, understanding the factors that lead to customer churn can help in developing more effective customer retention strategies.The main objective of this project is to predict customer churn using machine learning techniques. The project aims to identify the key factors that influence customer decisions to leave the bank and to develop a prediction model that can distinguish between customers likely to churn and those who are not, so that the bank could take the required steps to reduce the churn.

3.Data Source:

https://www.kaggle.com/datasets/shrutimechlearn/churn-modelling/data

4.Data Description:

The dataset consists of 10,000 rows and 14 columns, the description of all the columns are provided below:
RowNumber: Gives a unique number for each row in the dataset
CustomerId: Gives a unique customer ID for all the customers in the dataset
Surname: Gives the surname of all the customers
CreditScore: Gives the credit score of the customers
Geography: Gives the details on where the customer is located
Gender: Gives the information about customer's gender
Age: Gives the information of customer's age
Tenure: Gives the number of years the customer has been associated with the bank
Balance: Gives the information of account balance
NumOfProducts: Gives the information on how many bank products the customers are currently using
HasCrCard: Gives info on whether the customer is using a credit card or not
IsActiveMember: Indicates whether the customer is an active member of the bank to not
EstimatedSalary: The customer's estimated salary
Exited: Indicates whether the customer left (churned) or not
Target Variable for churned or Not: Exited

5. Library Imports:

The project utilizes several Python libraries to facilitate data manipulation, visualization, and modeling. These include:

Pandas and NumPy: For data operations such as data cleaning, filtering, and transformations.
Matplotlib, Seaborn, and Plotly: Used for creating interactive and static visualizations to uncover patterns in the data.
Scikit-learn: Provides efficient tools for data mining and data analysis, including several classification algorithms.
Imbalanced-learn: Helpful for dealing with the imbalanced dataset by applying techniques like SMOTE for oversampling the minority class.

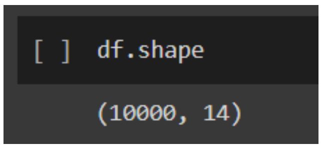
Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.graph_objects as go
import plotly.express as px
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import accuracy_score, precision_score,
recall_score ,f1_score
from sklearn.metrics import confusion_matrix, roc_curve, roc_auc_score
from imblearn.over_sampling import SMOTE
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from plotly.subplots import make_subplots
from sklearn.model_selection import RandomizedSearchCV
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
from sklearn.model_selection import RandomizedSearchCV
```

6. Data Inspection:

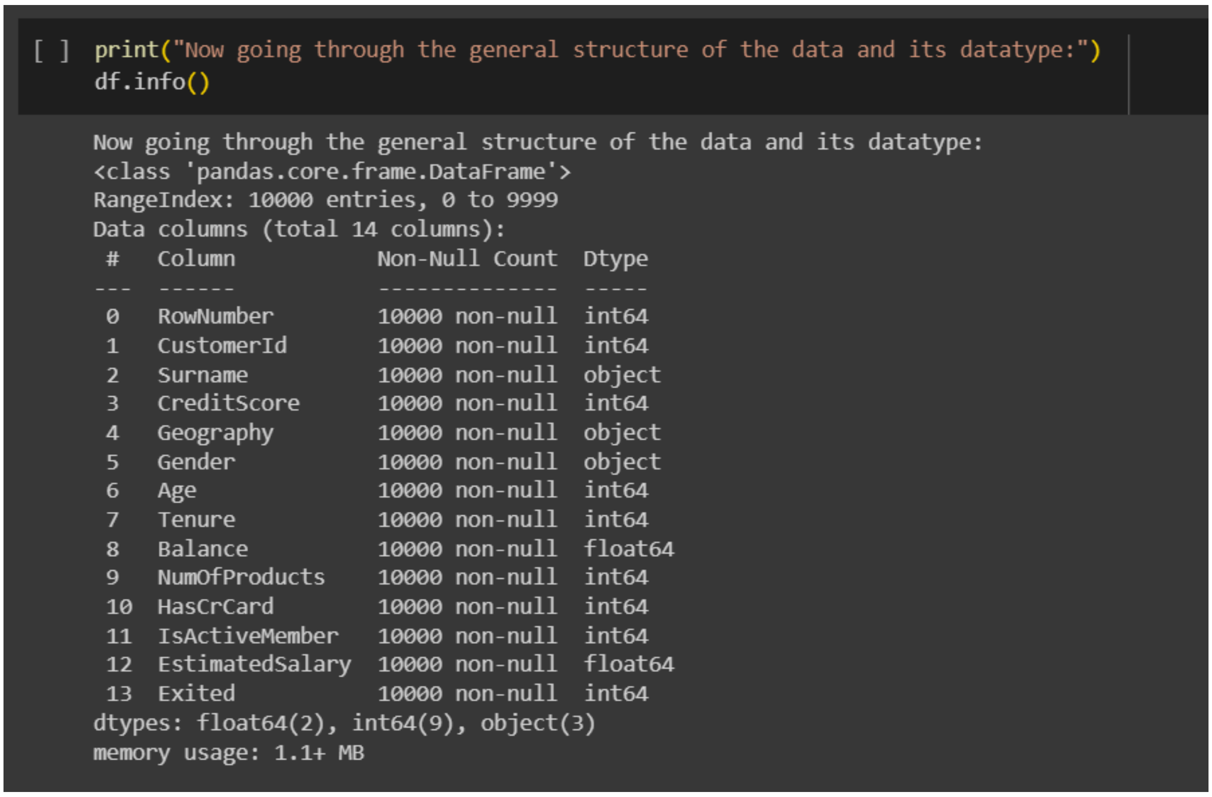
The initial dataset consists of 10,000 rows and 14 columns. Each entry represents a customer, with columns detailing personal and financial information, providing a robust basis for our analysis.

Code and Output:



```
[ ]  df.shape

     (10000, 14)
```

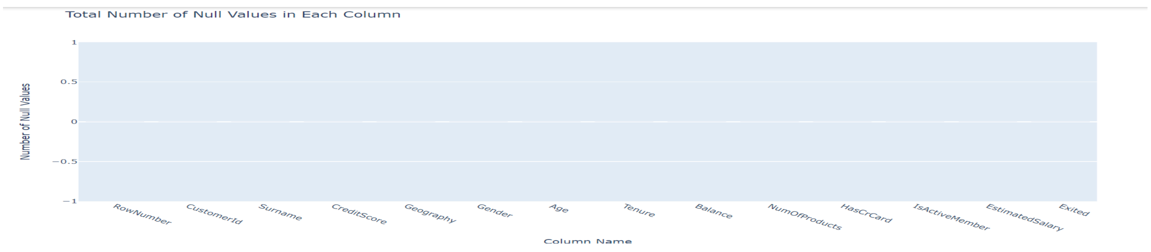
This code below shows the datatypes of all the columns.



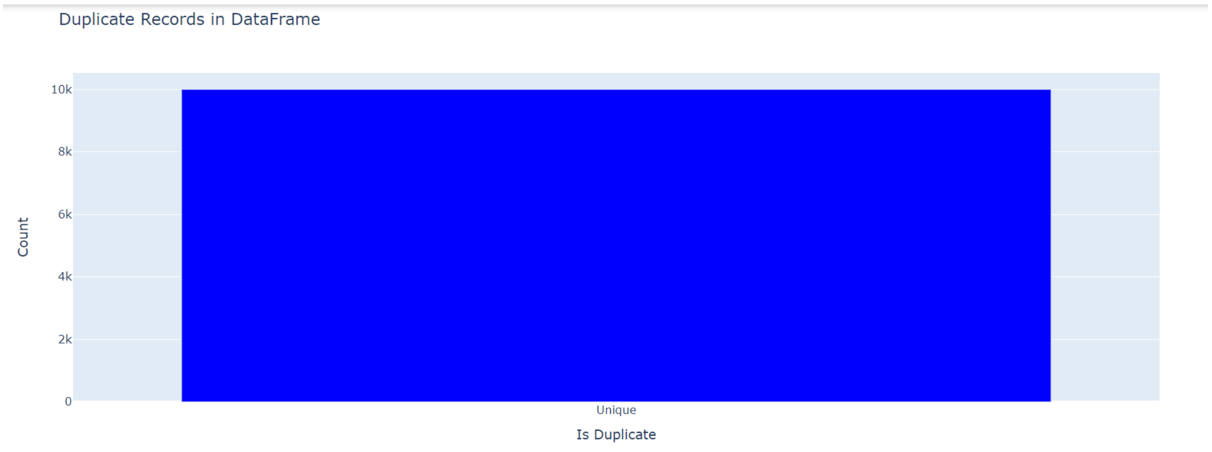
```
[ ]  print("Now going through the general structure of the data and its datatype:")
     df.info()

     Now going through the general structure of the data and its datatype:
     <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
```

7. Data Cleaning:

The integrity of data is paramount in predictive modeling. In our project, we ensured this through rigorous data cleaning processes, tailored to enhance the quality and relevance of the dataset. Herein, we detail the steps taken to prepare the data for subsequent analysis.A comprehensive search for null entries across the dataset was our initial step. The presence of null values can distort analysis outcomes; hence, their identification is crucial. Utilizing Python's Pandas library, we confirmed the absence of null values, as evidenced by a bar chart visualization (Figure 1). This affirmed that no further action was necessary regarding missing data.



Duplicate entries were the next target of our data cleaning endeavor. Repetitive data can lead to biased results; therefore, we employed Pandas' duplicated() function for detection. The resulting visualization (Figure 2) indicated a dataset free from duplicates, allowing us to proceed without the need for record deduplication.



We then moved to refine the dataset by dropping irrelevant columns: 'RowNumber', 'CustomerId', and 'Surname'. These fields, being mere identifiers, do not contribute to predicting customer churn and were thus excluded to narrow the focus to relevant predictors.

Finalizing the dataset involved renaming the 'Exited' column to 'ChurnedOrNot' to more accurately describe the target variable. Furthermore, we recoded its binary values to categorical labels 'No' and 'Yes', enhancing interpretability for later stages of analysis and visualization.

The data cleaning phase was a critical step in ensuring the reliability of our predictive analysis, eliminating unnecessary data while bolstering the dataset's clarity and relevance.

8. Exploratory Data Analysis (EDA):

We generated various plots to explore relationships between features and churn, including histograms for age distributions, scatter plots for balance vs. age, and correlation matrices to see interdependencies.

Graphical Analysis: Various plots were generated to explore relationships between features and churn, including histograms,boxplots , and bar graph

We have used the Plotly histogram and Box plots to visualize the numerical /continuous variables like Credit Score ,Age, Balance, Estimated Salary, and Tenure.

Target Variable Distribution:

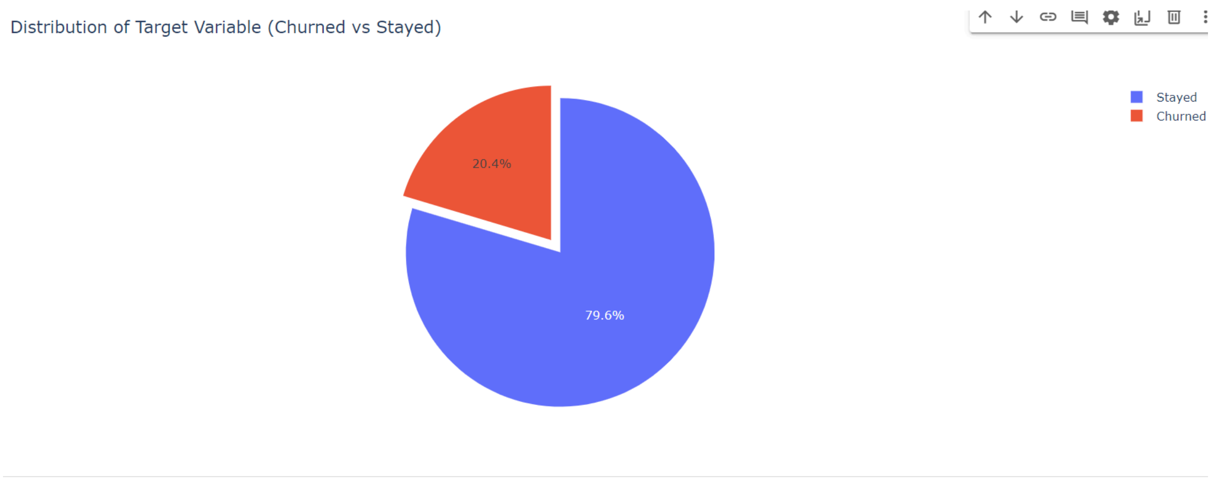


Fig.1 Distribution of Target Variable (Churned vs stayed)

This shows the distribution of the Target Variable , as we can see there is a huge imbalance in the target variable , this imbalance can be overcome by using the SMOTE oversampling technique

And now coming for categorical variables , we have used the Plotly Bar chart and Pie chart to visualize categorical variables like Gender ,Geography, NumofProducts, HasCrdCard, IsActiveMember.

Now after the general distribution of these features , we visualize the features based on churn status to gain more insights.

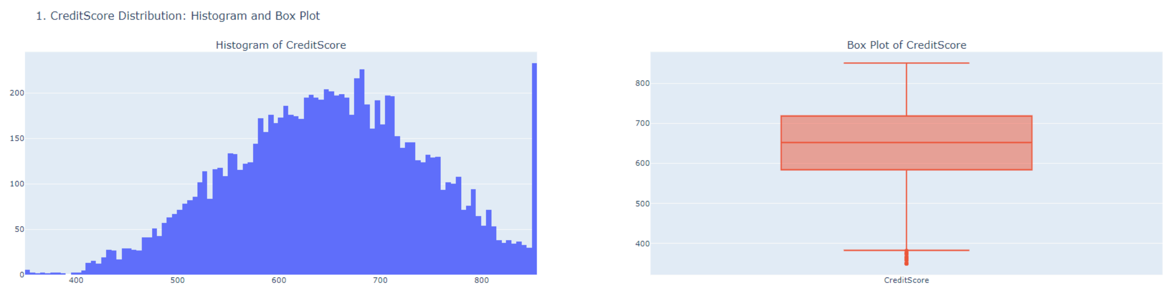


Fig2.CreditScore Distribution

The histogram indicates a generally normal distribution of credit scores with most customers clustered around the median, while the box plot reveals a median score in the mid-600s with some outliers on the lower end.

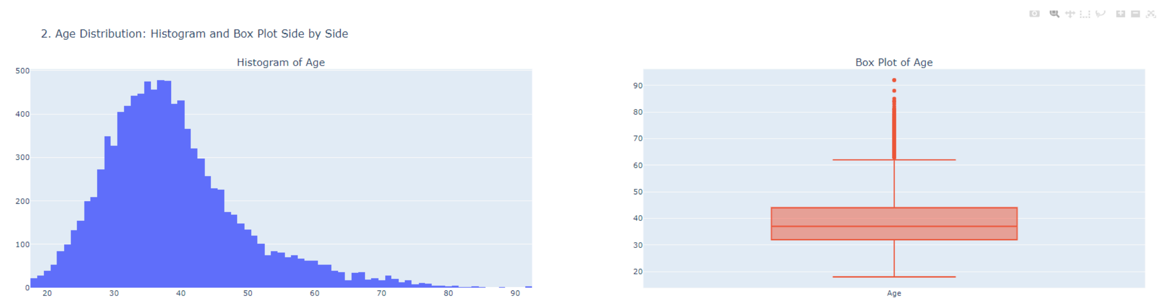


Fig3. Age distribution

The age distribution is skewed right, with a concentration of younger customers, and the box plot shows a median age in the late 30s with a few older age outliers. Also we can see that there is large number of outliers in the distribution.

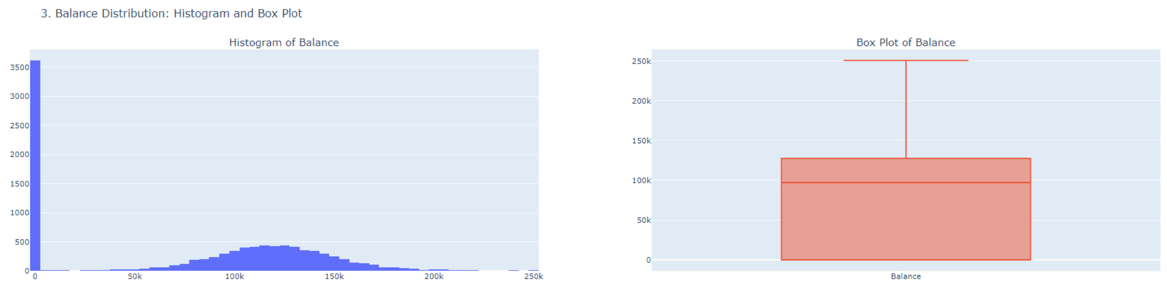


Fig4. Balance Distribution

The balance distribution is highly skewed with many customers having low balances, and the box plot indicates a wide range of balances.

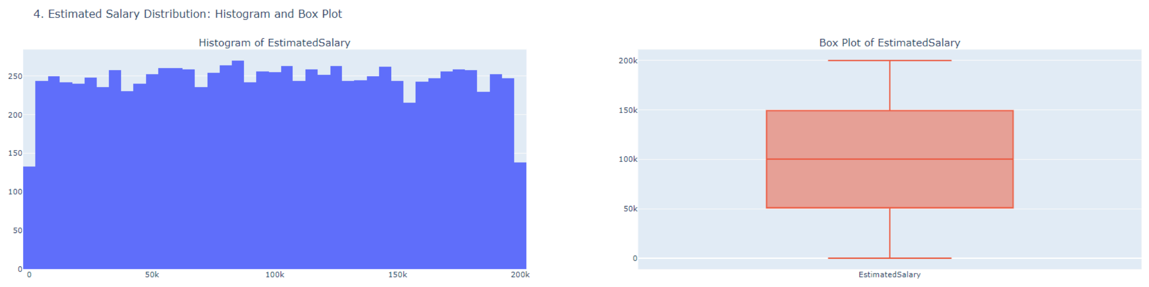


Fig5. Estimated Salary Distribution

The estimated salary distribution is uniformly spread across various ranges, and the box plot also shows an even distribution with no significant outliers.

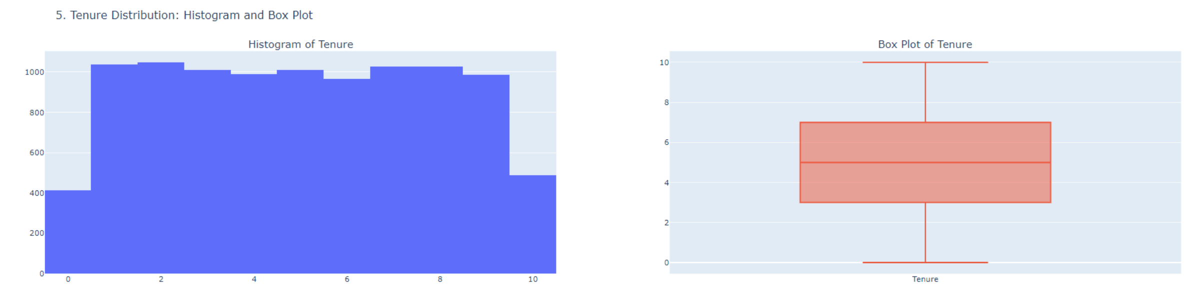


Fig6. Tenure Distribution

The tenure histogram displays a fairly uniform distribution across different lengths of service, while the box plot shows a median tenure of around 5 years with no outliers.

Now we are going to see the general distribution of categorical variables

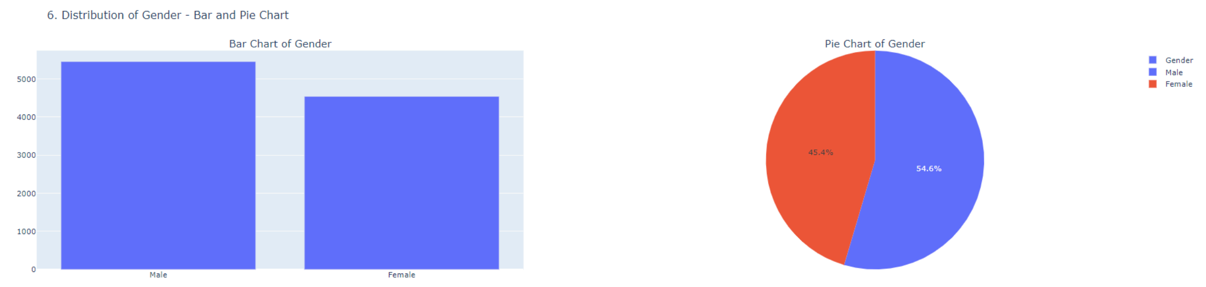


Fig7. Distribution of Gender

The bar chart and pie chart both indicate a slight majority of males compared to females in the dataset, with males representing just over half of the customers of the bank

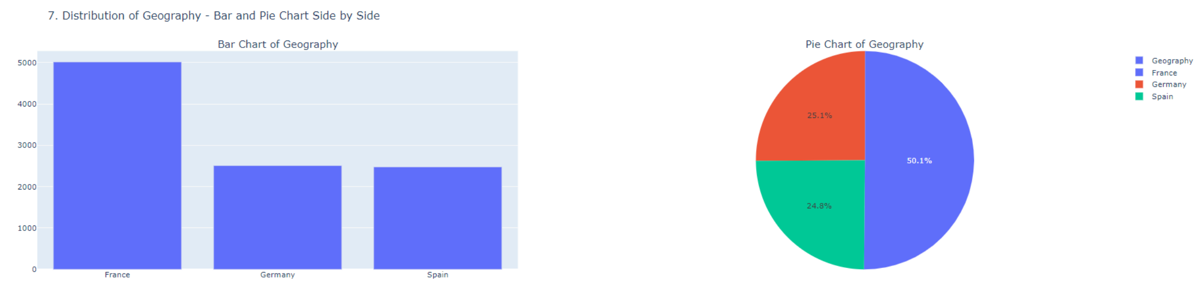


Fig8. Distribution of Geography

The graphs show France as having the highest customer count, with Germany and Spain having a similar and lower distribution of customers. This clearly shows the french form the most number of customers.

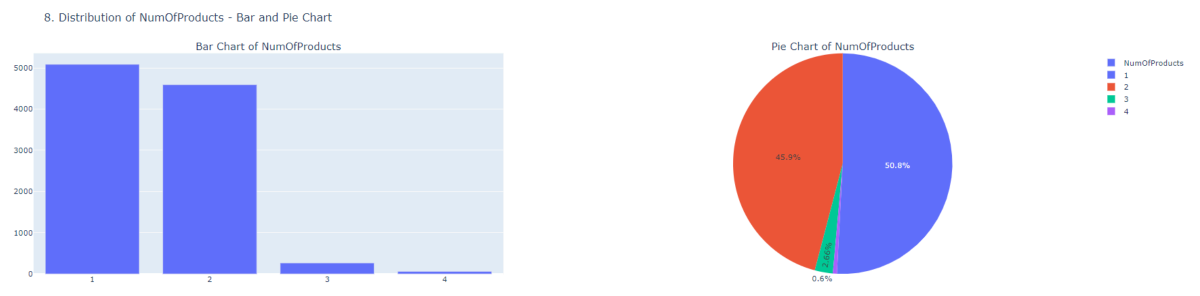


Fig9. Distribution of NumOfProducts

The visuals indicate that most customers have 1 or 2 bank products, with very few opting for 3 or 4 products. This indicates that most customers are not buying more than two products from the bank.

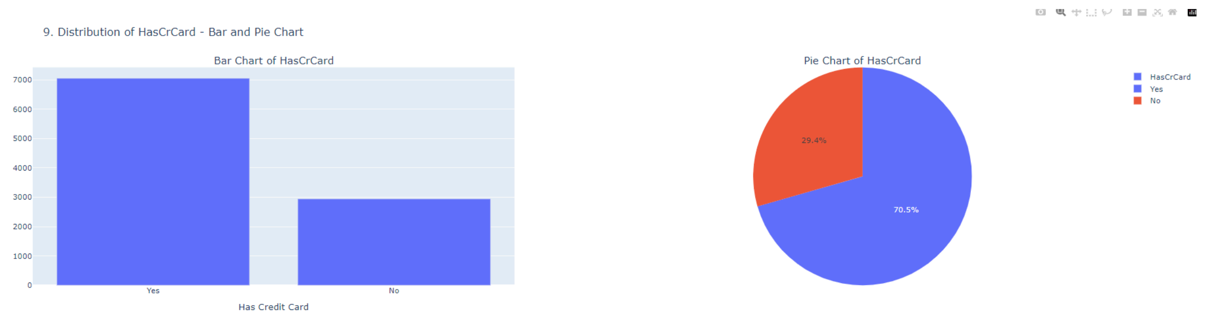


Fig10. Distribution of HasCrCard (has credit card)

A large majority of customers, over 70%, have a credit card, as shown by the bar and pie charts. Almost all the customers,around 7000/10,000 have credit cards.

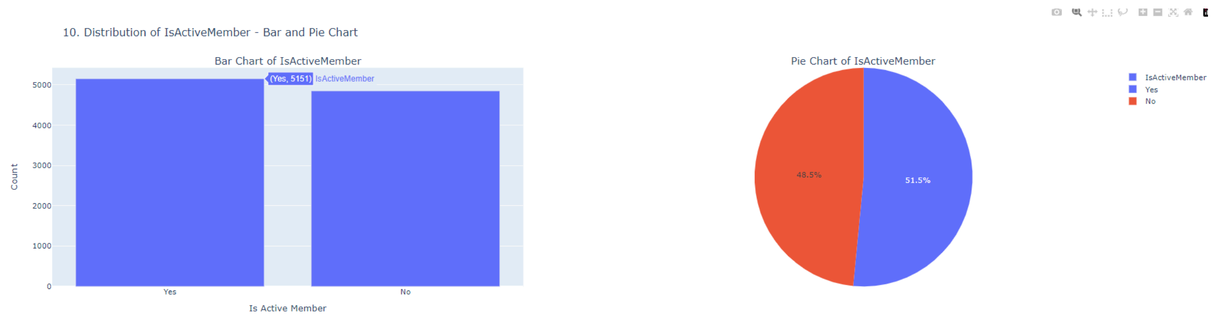


Fig11. Distribution of IsActiveMember

The customer base is nearly evenly split between active and inactive members, with a slight majority being active.

Now we are going to visualize the distribution of various features based on their churn status,

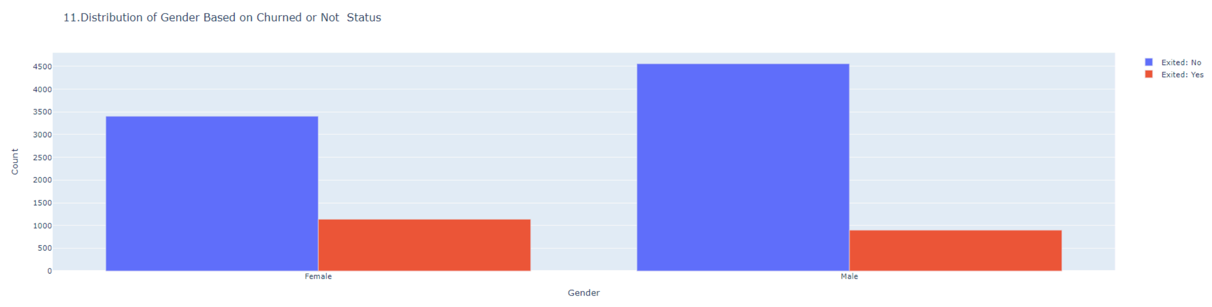


Fig12. Distribution of Gender Based on Churned or Not Status

The bar chart displays a higher count of female customers who have churned compared to male customers. This indicates that the bank must take more female friendly initiatives to retain more female customers. The bank can also offer lower interest rates for female customers and focus on marketing specifically targeted on women.

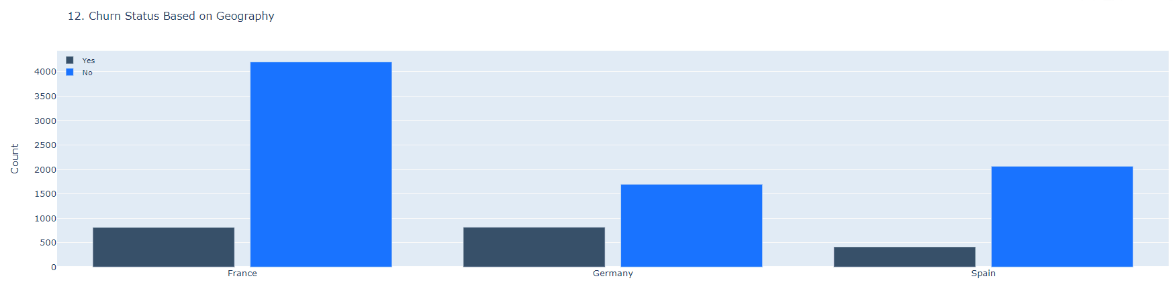


Fig13. Churn Status Based on Geography

Churn rates vary by geography, with France showing the highest number of retained customers and Germany having a notably higher churn rate in comparison. This clearly indicates that the bank needs to analyze their market in Germany. They should also see if they are providing the same facilities and following the same Modus operandi that they are following in the french market. This churn may also be due to the local market in germany and the bank may need to tailor thor strategies to perform better in Germany.

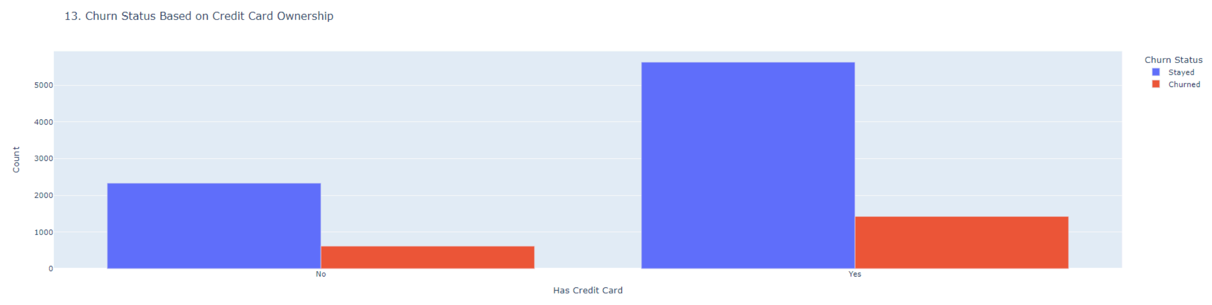


Fig14. Churn Status Based on Credit Card Ownership

The chart shows credit card ownership does not have a significant impact on churn, with both cardholders and non-cardholders displaying a mix of churned and retained statuses.

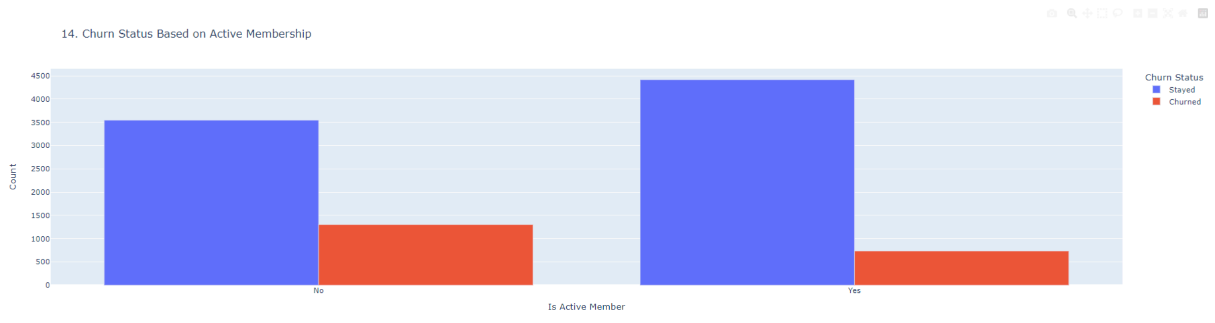


Fig15. Churn Status Based on Active Membership

Inactive members show a higher likelihood of churn compared to active members. This indicates that the bank should make sure that the customers are sufficiently engaged with the bank and this they can provide using enhanced digital services and regular updates.

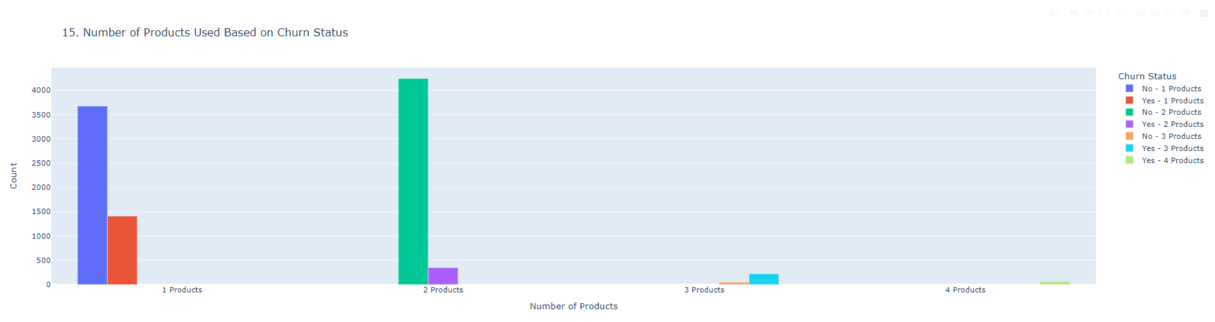


Fig16. Number of Products Used Based on Churn Status

Customers with only one product have a higher churn rate compared to those with two, while very few customers subscribe to three or four products. And a significant amount of people who have 3 products get churned. The least amount of churn happens when people are using two products. Also since the number of people having more than two products is very less, we can do feature engineering by combining the people who are having more than two products.

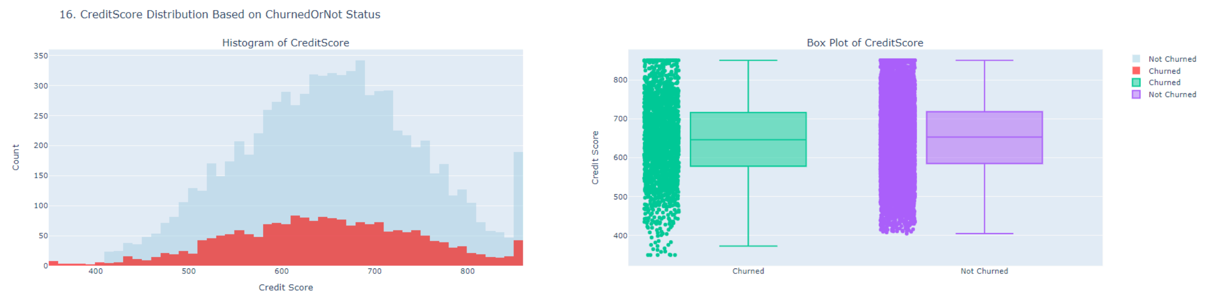


Fig17. Credit Score Distribution Based on Churned or Not Status

Both Churned and not churned customers have almost the same median and distribution looks almost similar too, this indicates that credit score doesn’t have much impact on the churn status.

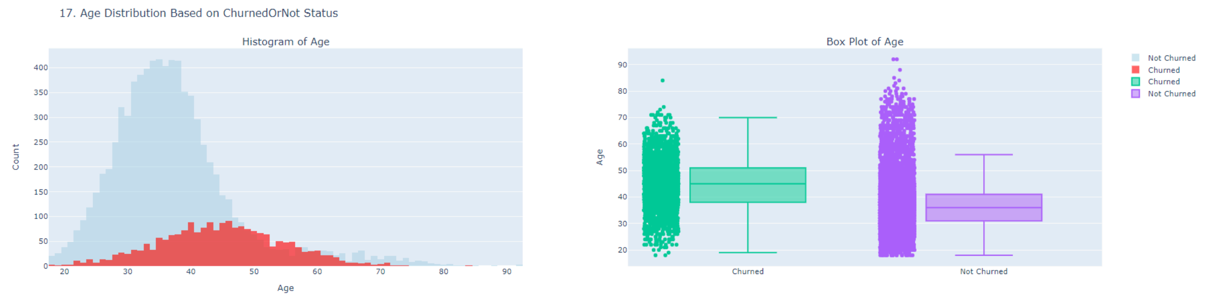


Fig18. Age Distribution Based on Churned or Not Status

Churned customers tend to be older, as shown by the age histogram and box plot, with a higher median age compared to those who haven't churned. Also the distribution looks quite skewed with a lot of outliers which could lead to overfitting. We will use the log normal transformation to overcome this.

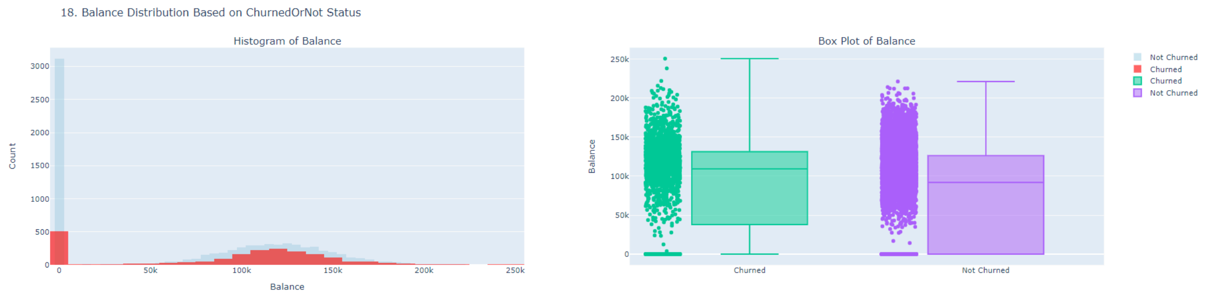


Fig19. Balance Distribution Based on Churned or Not Status

Customers who churned generally had lower bank balances, as suggested by the overlay of distributions and the shifted box plot. Also it is clear that the customers with zero bank are more likely to churn ,other than that the distribution is distributed normally. In feature engineering step we will group customers with zero balance and more than zero balance separately.

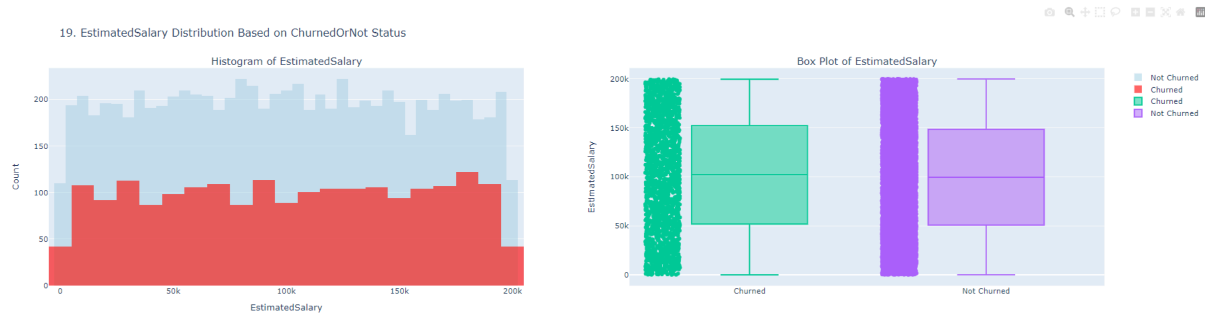


Fig20. Estimated salary Distribution Based on Churned or Not Status

There's no distinct difference in estimated salary distribution between customers who churned and those who did not, as both histograms and box plots overlap significantly. Plus as we can see from the boxplot the median estimated salary of customers who were churned and not churned are the same

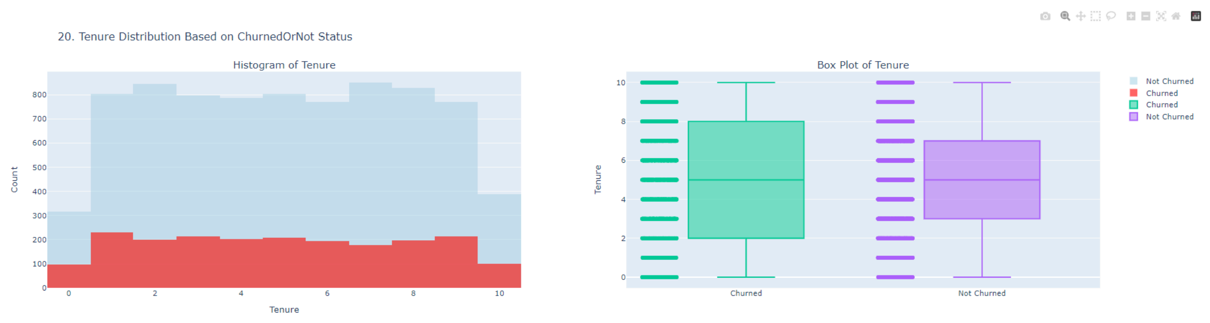


Fig21. Tenure Distribution Based on Churned or Not Status

Tenure does not appear to be a defining factor for churn, with similar distributions across churned and not churned customers.

9. Feature Engineering:

Description: New features were created to enhance model accuracy. For example, customer age groups were derived from the age feature to categorize customers into different life stages.

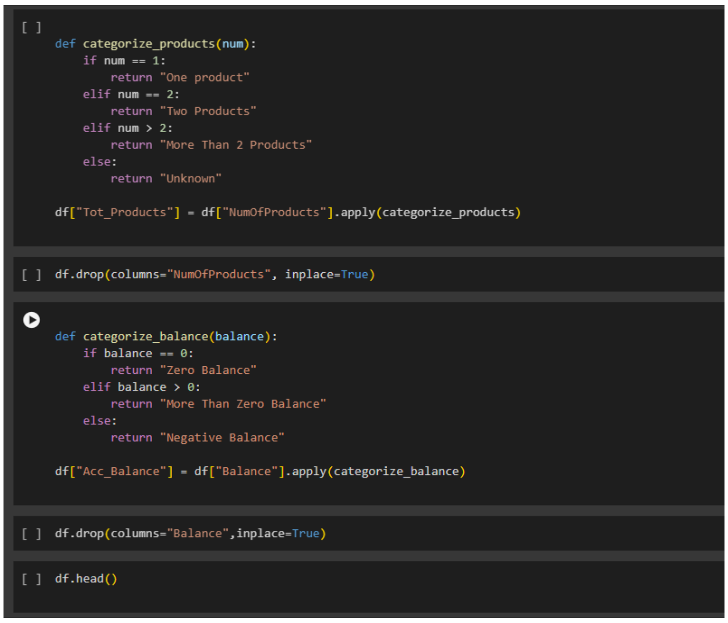
This code snippet demonstrates feature engineering techniques applied to a dataset for customer churn analysis.

Firstly, the function `categorize_products` takes the number of products as input (`num`) and categorizes them into three groups: "One product", "Two products", and "More Than 2 Products". If the number is less than or equal to 0, it's categorized as "Unknown". This function is then applied to the "NumOfProducts" column in the DataFrame, and the result is stored in a new column named "Tot_Products". Afterward, the original "NumOfProducts" column is dropped from the DataFrame.

Similarly, the function `categorize_balance` takes the balance value as input (`balance`) and categorizes it into two groups: "Zero Balance" if the balance is 0, and "More Than Zero Balance" if the balance is greater than 0. If the balance is negative, it's categorized as such. This function is applied to the "Balance" column in the DataFrame, and the result is stored in a new column named "Acc_Balance". Then, the original "Balance" column is dropped from the DataFrame.

Overall, these transformations aim to simplify and categorize the data, making it more suitable for analysis, particularly in the context of customer churn prediction. By transforming continuous variables into categorical ones, the model may better capture underlying patterns and relationships in the data, ultimately improving predictive performance.

Code and Output:



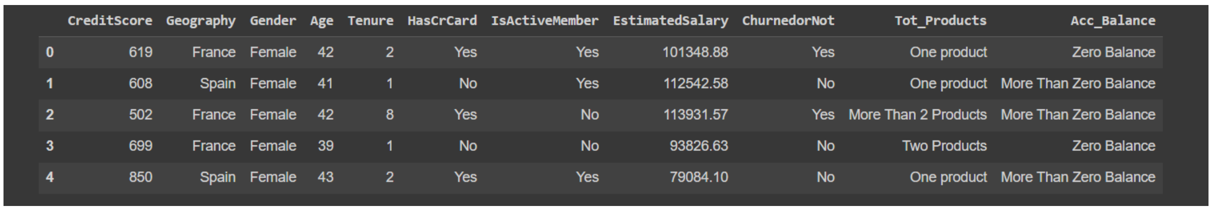
```
[ ]
    def categorize_products(num):
        if num == 1:
            return "One product"
        elif num == 2:
            return "Two Products"
        elif num > 2:
            return "More Than 2 Products"
        else:
            return "Unknown"

    df["Tot_Products"] = df["NumOfProducts"].apply(categorize_products)
```

```
[ ] df.drop(columns="NumOfProducts", inplace=True)
```

```
    def categorize_balance(balance):
        if balance == 0:
            return "Zero Balance"
        elif balance > 0:
            return "More Than Zero Balance"
        else:
            return "Negative Balance"

    df["Acc_Balance"] = df["Balance"].apply(categorize_balance)
```

```
[ ] df.drop(columns="Balance",inplace=True)
```

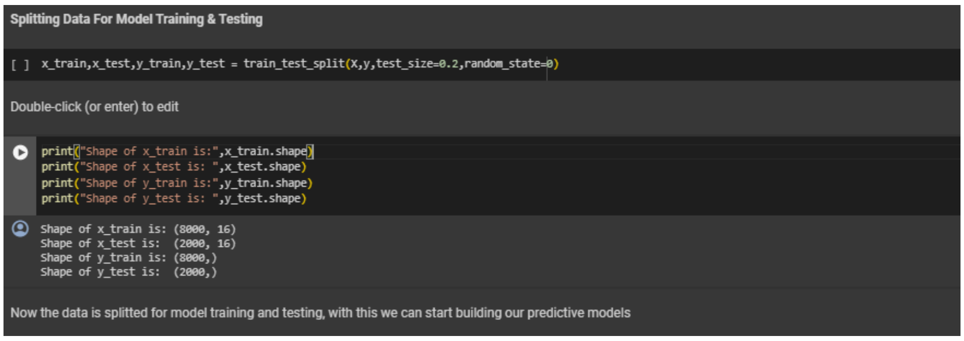
```
[ ] df.head()
```



	CreditScore	Geography	Gender	Age	Tenure	HasCrCard	IsActiveMember	EstimatedSalary	ChurnedorNot	Tot_Products	Acc_Balance
0	619	France	Female	42	2	Yes	Yes	101348.88	Yes	One product	Zero Balance
1	608	Spain	Female	41	1	No	Yes	112542.58	No	One product	More Than Zero Balance
2	502	France	Female	42	8	Yes	No	113931.57	Yes	More Than 2 Products	More Than Zero Balance
3	699	France	Female	39	1	No	No	93826.63	No	Two Products	Zero Balance
4	850	Spain	Female	43	2	Yes	Yes	79084.10	No	One product	More Than Zero Balance

10. Model Building and Evaluation:

This code efficiently splits the dataset into training and testing sets for model training and evaluation. It utilizes the `train_test_split` function from scikit-learn, partitioning the features (`X`) and target variable (`y`) with a test size of 20% and a specified random state for reproducibility. After the split, it prints the shapes of the resulting training and testing sets (`x_train`, `x_test`, `y_train`, `y_test`) to confirm the correct division of data. This ensures the integrity of subsequent model training and testing processes.



Splitting Data For Model Training & Testing

```
[ ] x_train,x_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=0)
```

Double-click (or enter) to edit

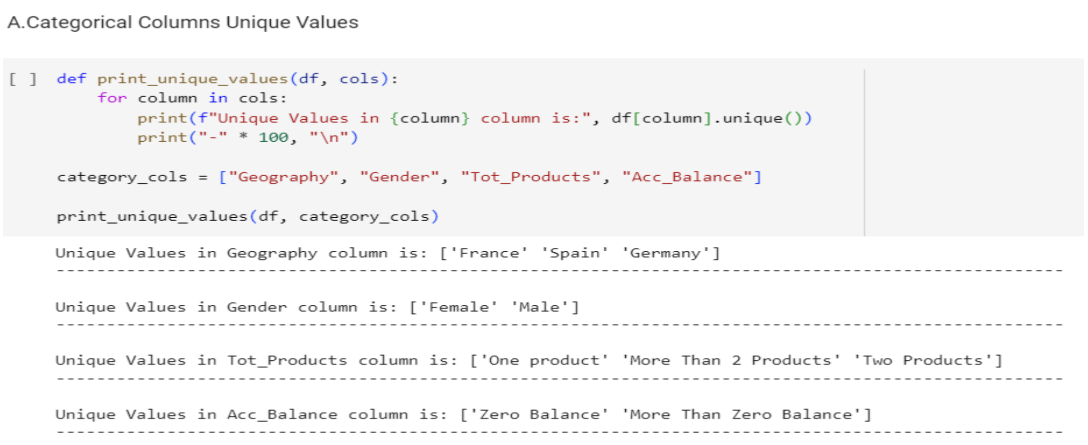
```
print("Shape of x_train is:",x_train.shape)
print("Shape of x_test is: ",x_test.shape)
print("Shape of y_train is:",y_train.shape)
print("Shape of y_test is: ",y_test.shape)
```

```
Shape of x_train is: (8000, 16)
Shape of x_test is:  (2000, 16)
Shape of y_train is: (8000,)
Shape of y_test is:  (2000,)
```

Now the data is splitted for model training and testing, with this we can start building our predictive models

10.1 Data Preprocessing:

Now that we have completed the feature evaluation, we will start with the data preprocessing steps. As a first step we will start with computing the unique values of all the categorical columns like Geography, Gender,Total_products.


A.Categorical Columns Unique Values

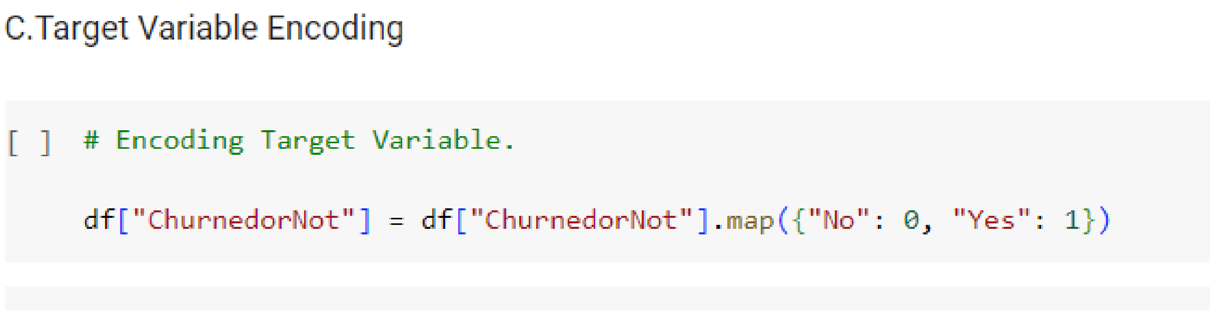
```
[ ] def print_unique_values(df, cols):
        for column in cols:
            print(f"Unique Values in {column} column is:", df[column].unique())
            print("-" * 100, "\n")

    category_cols = ["Geography", "Gender", "Tot_Products", "Acc_Balance"]

    print_unique_values(df, category_cols)
```

```
Unique Values in Geography column is: ['France' 'Spain' 'Germany']
----------------------------------------------------------------------------------------------------

Unique Values in Gender column is: ['Female' 'Male']
----------------------------------------------------------------------------------------------------

Unique Values in Tot_Products column is: ['One product' 'More Than 2 Products' 'Two Products']
----------------------------------------------------------------------------------------------------

Unique Values in Acc_Balance column is: ['Zero Balance' 'More Than Zero Balance']
----------------------------------------------------------------------------------------------------
```

As we can see the unique values of geography as France, Germany, and Spain and for Gender it is male and female and for Tot_Products it is One product , More than two products and Two Products and for Acc_Balance it is Zero Balance,More than Zero Balance.

One-hot encoding is a technique used in machine learning to represent categorical data numerically.It works by converting each unique category value into a new column and assigns a 1 or 0 (notation for true/false) value to the column. This is particularly useful for handling categorical data types for which the categories do not have an ordinal relationship, meaning that one category is not necessarily greater than another.After this operation, for every unique value in the original column, a new column is created where the presence of the value is marked with a 1 and the absence with a 0.

Next, we encode the target variable,

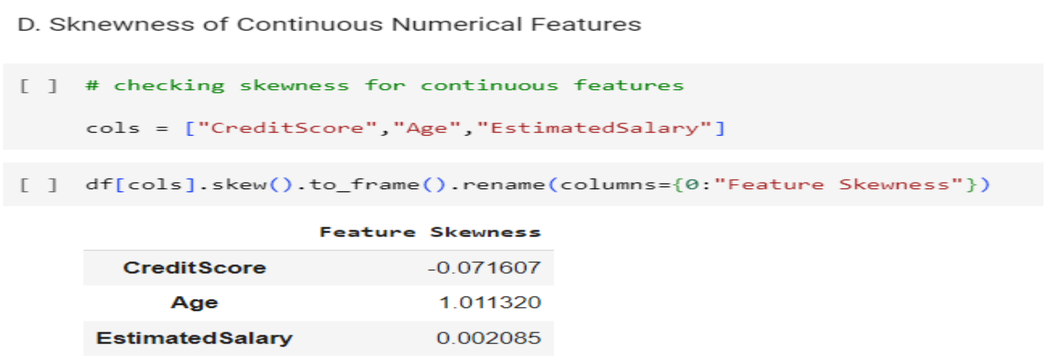

C.Target Variable Encoding

```
[ ] # Encoding Target Variable.

    df["ChurnedorNot"] = df["ChurnedorNot"].map({"No": 0, "Yes": 1})
```

This snippet of Python code is used for encoding a target variable, which in this case appears to be a binary classification variable named "ChurnedorNot".So after this operation, all instances of "No" in the "ChurnedorNot" column will be replaced with 0, and all instances of

"Yes" will be replaced with 1. This transforms the column into a format suitable for use as a target variable in machine learning algorithms, especially for binary classification tasks.

Next for the numerical/continuous variable we are checking for the skewness,



D. Skewness of Continuous Numerical Features

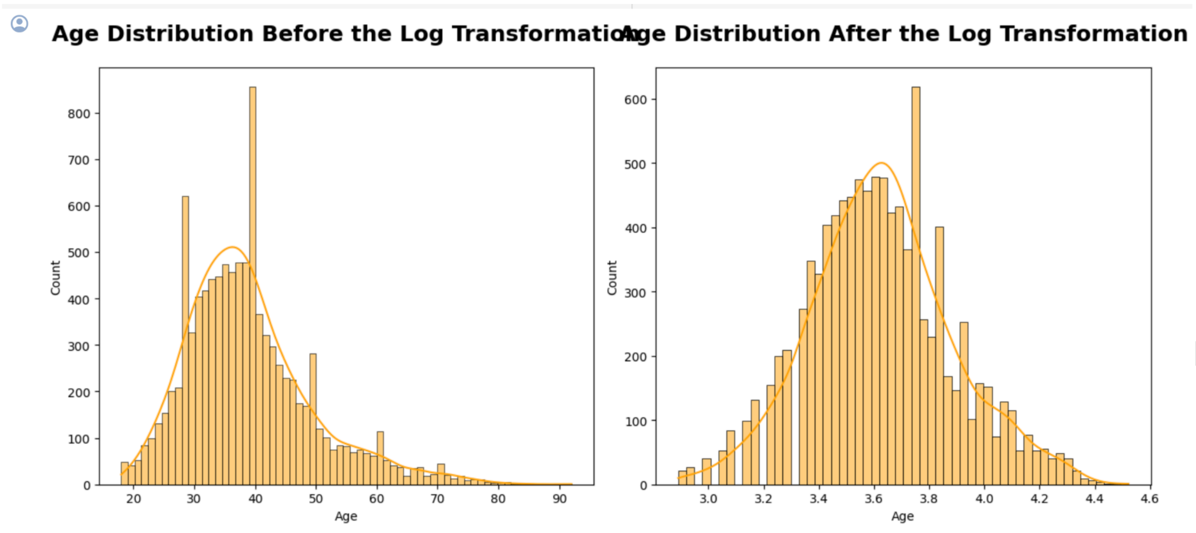
```
[ ]  # checking skewness for continuous features

     cols = ["CreditScore","Age","EstimatedSalary"]
```

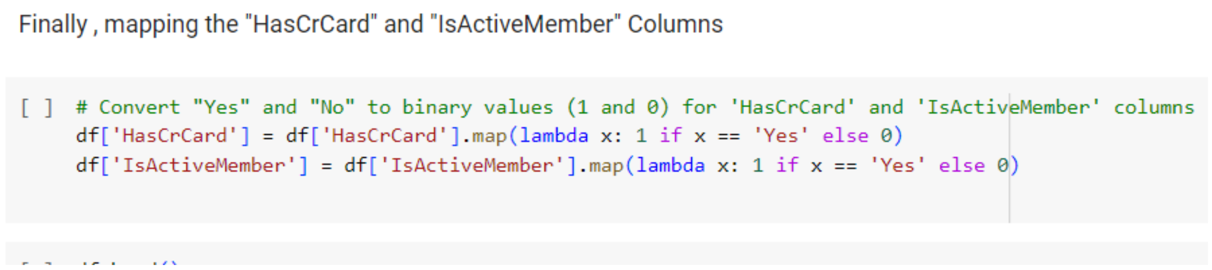
```
[ ]  df[cols].skew().to_frame().rename(columns={0:"Feature Skewness"})
```

	Feature Skewness
CreditScore	-0.071607
Age	1.011320
EstimatedSalary	0.002085

From the above table we can see that the age feature is skewed rightly which could be due to the presence of a lot of positive outliers , which can affect our models, so we do log normal transformation.

Now after the log normal transformation,

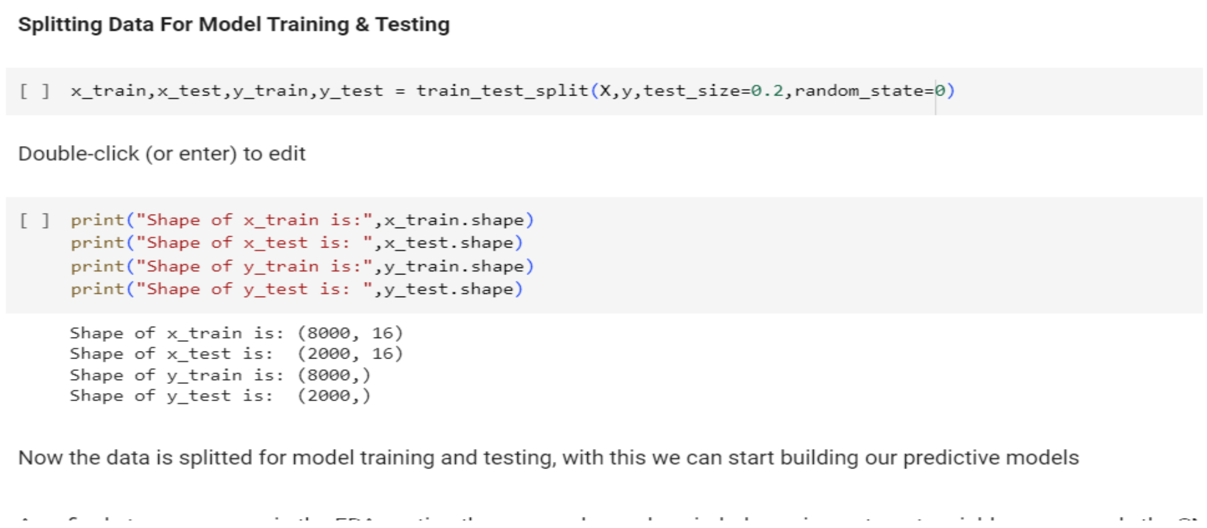




Finally , mapping the "HasCrCard" and "IsActiveMember" Columns

```
[ ]  # Convert "Yes" and "No" to binary values (1 and 0) for 'HasCrCard' and 'IsActiveMember' columns
     df['HasCrCard'] = df['HasCrCard'].map(lambda x: 1 if x == 'Yes' else 0)
     df['IsActiveMember'] = df['IsActiveMember'].map(lambda x: 1 if x == 'Yes' else 0)
```

As a final strep, these two lines of code convert the "HasCrCard" and "IsActiveMember" columns from categorical ('Yes'/'No') to binary numerical (1/0) form, which is a common preprocessing step before feeding the data to machine learning models that require numerical input.


Splitting Data For Model Training & Testing

```
[ ]  x_train,x_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=0)
```

Double-click (or enter) to edit

```
[ ]  print("Shape of x_train is:",x_train.shape)
     print("Shape of x_test is: ",x_test.shape)
     print("Shape of y_train is:",y_train.shape)
     print("Shape of y_test is: ",y_test.shape)
```

```
     Shape of x_train is: (8000, 16)
     Shape of x_test is:  (2000, 16)
     Shape of y_train is: (8000,)
     Shape of y_test is:  (2000,)
```

Now the data is splitted for model training and testing, with this we can start building our predictive models

The data was divided into two parts: a training set and a testing set. The training set consists of 80% of the data and is used to train the model, while the testing set comprises the remaining 20% and is used to evaluate the model's performance. This division was achieved using a standard function in machine learning libraries that ensures a randomized split, with a specific parameter set to ensure that the randomization is consistent on each run. As a result, the training set contains 8,000 examples, each with 16 features, and the testing set consists of 2,000 examples with the same number of features. Corresponding target values were also split accordingly, matching the number of examples in the training and testing sets. This process is crucial to avoid overfitting and to test the model’s ability to generalize to new, unseen data

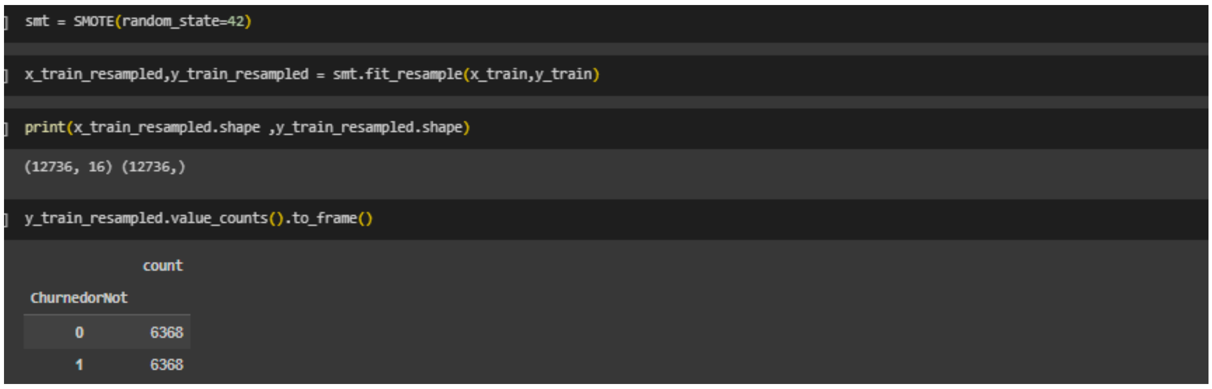
10.2 Smote Analysis:

This code snippet addresses the issue of class imbalance in the target variable observed during the Exploratory Data Analysis (EDA) phase by applying Synthetic Minority Over-sampling Technique (SMOTE). SMOTE is employed to balance the class distribution by generating synthetic samples for the minority class.

The `SMOTE` function from the `imbalanced-learn` library is used with a specified random state for reproducibility. It resamples the training data (`x_train` and `y_train`) to create a balanced dataset.

After applying SMOTE, the code prints out the shapes of the resampled training data (`x_train_resampled`, `y_train_resampled`) to confirm the new dimensions. Additionally, it displays the value counts of the resampled target variable (`y_train_resampled`) to illustrate the balanced class distribution.

By employing SMOTE, the code aims to mitigate the effects of class imbalance, ensuring more robust model training and better performance, especially for predictive models sensitive to class distribution.
Each model, including KNN, Decision Trees, and Random Forests, was evaluated using metrics like accuracy, precision, recall, and F1-score. The KNN model showed superior performance, making it the best candidate for deployment.



```
smt = SMOTE(random_state=42)
```

```
x_train_resampled,y_train_resampled = smt.fit_resample(x_train,y_train)
```

```
print(x_train_resampled.shape ,y_train_resampled.shape)
```

```
(12736, 16) (12736,)
```

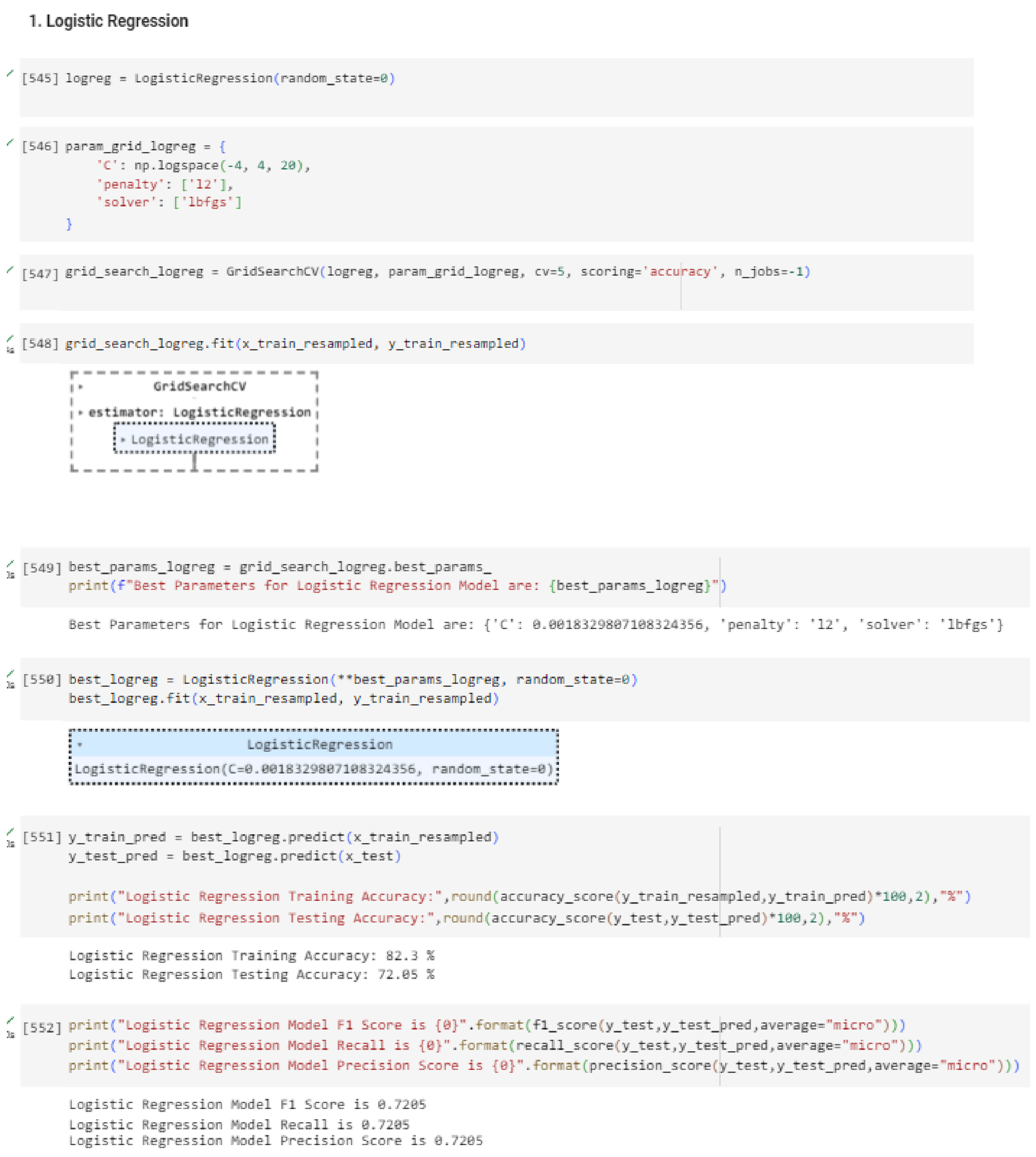
```
y_train_resampled.value_counts().to_frame()
```

	count
ChurnedorNot	
0	6368
1	6368

10.3 Logistic Regression:

Logistic regression is a popular choice for binary classification tasks due to its interpretability, efficiency, and ability to handle large datasets. It provides interpretable results by estimating the probability that an instance belongs to a particular class, making it easy to understand the influence of each feature on the classification outcome. Logistic regression is computationally efficient and can be regularized to prevent overfitting, ensuring robust performance even with noisy data. Its linear decision boundary assumption and probabilistic predictions further enhance its utility for various applications, including finance, healthcare, and marketing. However, logistic regression is limited to binary classification tasks and may not be suitable for problems with multiple classes or complex non-linear relationships between features and target variables.

This code segment presents a comprehensive approach to logistic regression modeling and evaluation. It begins by initializing a logistic regression model and defining a parameter grid for hyperparameter tuning. Subsequently, grid search cross-validation is employed to determine the optimal hyperparameters for the model. Once the best parameters are identified, the model is re-initialized with these parameters and trained on resampled training data generated through SMOTE to address class imbalance. Following model training, predictions are made on both training and testing datasets, and various evaluation metrics including accuracy, F1 score, recall, and precision are computed and displayed. This approach ensures the logistic regression model is trained and evaluated rigorously, taking into account the class imbalance issue and optimizing model performance.



1. Logistic Regression

```
[545] logreg = LogisticRegression(random_state=0)
```

```
[546] param_grid_logreg = {
          'C': np.logspace(-4, 4, 20),
          'penalty': ['l2'],
          'solver': ['lbfgs']
      }
```

```
[547] grid_search_logreg = GridSearchCV(logreg, param_grid_logreg, cv=5, scoring='accuracy', n_jobs=-1)
```

```
[548] grid_search_logreg.fit(x_train_resampled, y_train_resampled)
```

```
      GridSearchCV
 estimator: LogisticRegression
   LogisticRegression
```

```
[549] best_params_logreg = grid_search_logreg.best_params_
      print(f"Best Parameters for Logistic Regression Model are: {best_params_logreg}")

      Best Parameters for Logistic Regression Model are: {'C': 0.0018329807108324356, 'penalty': 'l2', 'solver': 'lbfgs'}
```

```
[550] best_logreg = LogisticRegression(**best_params_logreg, random_state=0)
      best_logreg.fit(x_train_resampled, y_train_resampled)
```

```
                    LogisticRegression
LogisticRegression(C=0.0018329807108324356, random_state=0)
```

```
[551] y_train_pred = best_logreg.predict(x_train_resampled)
      y_test_pred = best_logreg.predict(x_test)

      print("Logistic Regression Training Accuracy:",round(accuracy_score(y_train_resampled,y_train_pred)*100,2),"%")
      print("Logistic Regression Testing Accuracy:",round(accuracy_score(y_test,y_test_pred)*100,2),"%")

      Logistic Regression Training Accuracy: 82.3 %
      Logistic Regression Testing Accuracy: 72.05 %
```

```
[552] print("Logistic Regression Model F1 Score is {0}".format(f1_score(y_test,y_test_pred,average="micro")))
      print("Logistic Regression Model Recall is {0}".format(recall_score(y_test,y_test_pred,average="micro")))
      print("Logistic Regression Model Precision Score is {0}".format(precision_score(y_test,y_test_pred,average="micro")))

      Logistic Regression Model F1 Score is 0.7205
      Logistic Regression Model Recall is 0.7205
      Logistic Regression Model Precision Score is 0.7205
```

From the provided metrics:

- **F1 Score, Recall, and Precision**: With all three metrics having similar values (0.7205), it suggests that the model performs consistently across different aspects of classification evaluation. This indicates a good balance between correctly identifying positive cases (churn instances) and minimizing false positives.

- **Accuracy Score on Training Data (82.3%)**: The model performs relatively well on the training data, achieving an accuracy of 82.3%. This suggests that the model can capture the underlying patterns and relationships present in the training data.

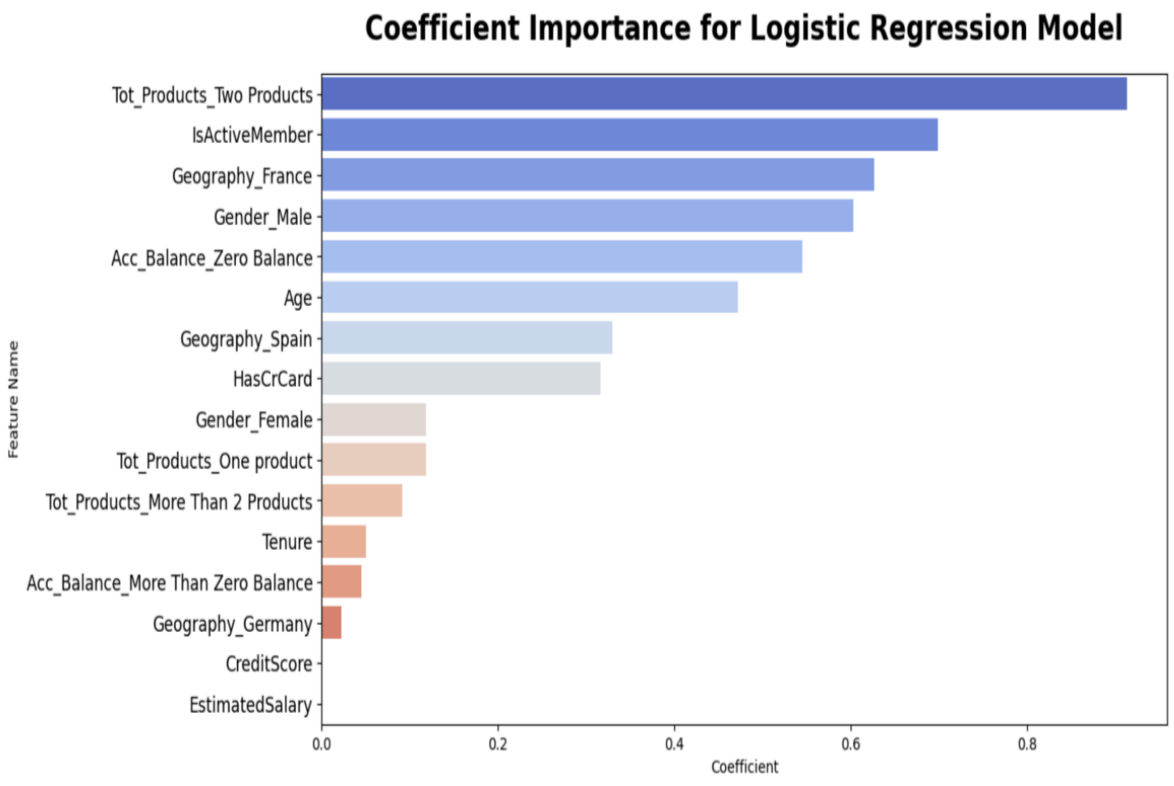
- **Accuracy Score on Testing Data (72.05%)**: The accuracy on the testing data is slightly lower than the training data but still respectable at 72.05%. It implies that the model generalizes reasonably well to unseen data, indicating robustness.

10.3.1 Feature Importance:

In the chart, 'Tot_Products_Two Products' has the highest positive coefficient, suggesting that having exactly two products is strongly associated with an increased likelihood of churn. The 'IsActiveMember' feature follows, indicating that active membership status is a significant predictor but with a negative relationship to churn—active members are less likely to leave.

Other features like 'Geography_France' and 'Gender_Male' also play a role, but to a lesser extent. The blue bars represent positive relationships with churn, whereas the brown bars represent negative relationships. Features with smaller absolute coefficient values, such as 'CreditScore' and 'EstimatedSalary', seem to have a weaker influence on the prediction of churn.

This coefficient importance chart can help business stakeholders understand which features contribute most to customer churn and tailor their customer retention strategies accordingly. The insights could lead to targeted interventions focused on the most influential factors to reduce churn rates effectively.



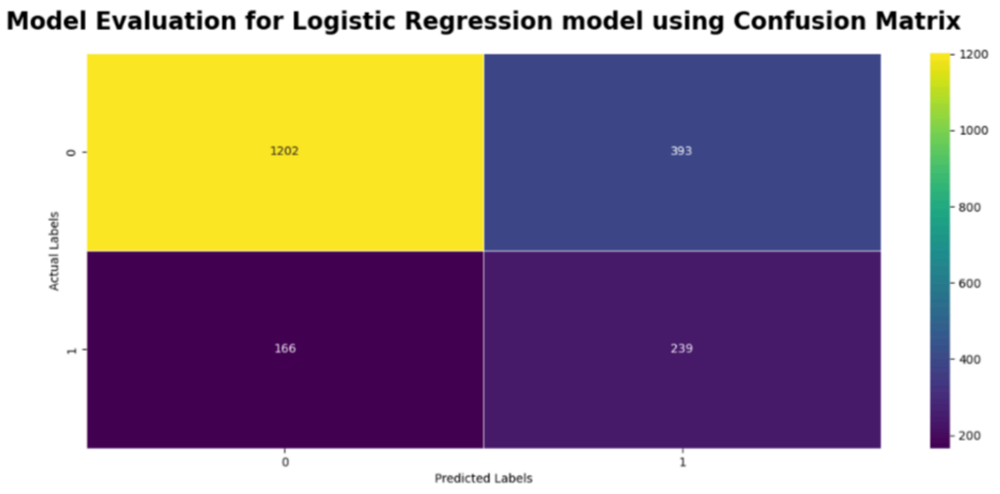
10.3.2 Confusion Matrix:

True Negatives (Actual label 0, Predicted label 0): There are 1202 instances where the model correctly predicted customers would not churn, which shows its ability to identify true non-churn events.

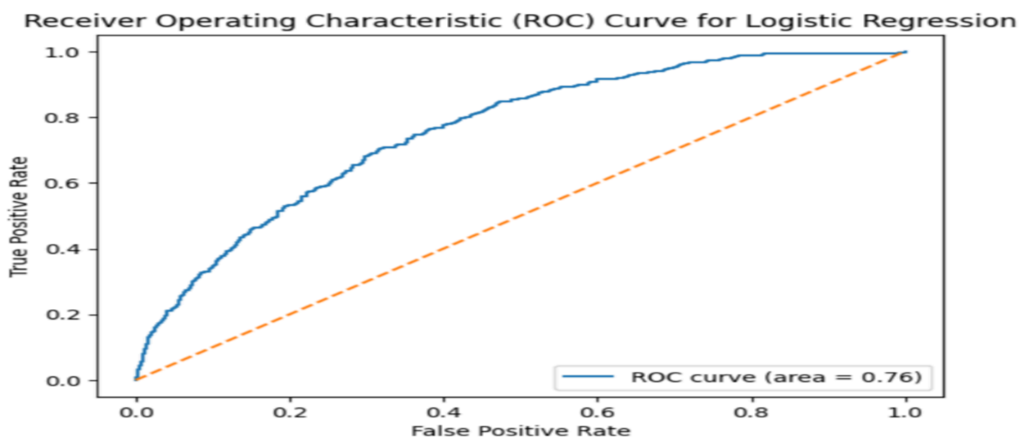
False Negatives (Actual label 1, Predicted label 0): There are 166 instances where the model failed to identify customers who would churn, marking these as non-churn instead.

False Positives (Actual label 0, Predicted label 1): There are 393 instances where the model incorrectly predicted customers would churn when they did not, representing a type I error.

True Positives (Actual label 1, Predicted label 1): There are 239 instances where the model correctly identified customers who would churn, demonstrating its effectiveness in detecting true churn events.



10.3.3 ROC Curve:

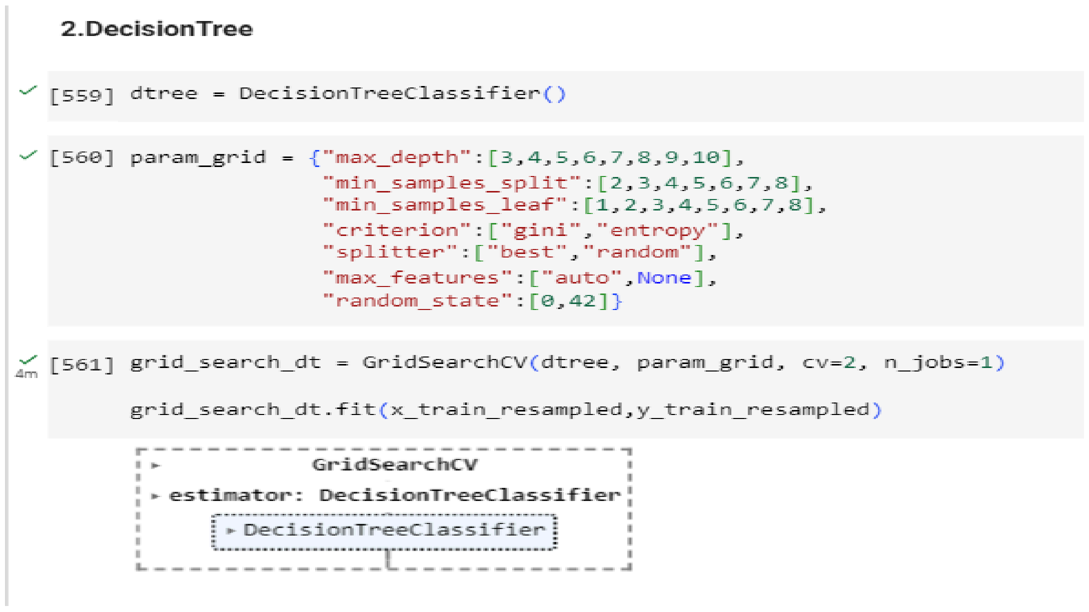


The ROC curve for the Logistic Regression model, with an AUC of 0.76, demonstrates a respectable predictive performance. This score indicates that the model has a good ability to distinguish between customers who will churn and those who will not. The curve's shape, rising towards the upper left corner but not hugging the corner closely, suggests that the model correctly identifies a substantial number of true positives while maintaining a moderate rate of false positives. Overall, the model offers a solid but not exceptional capability to classify outcomes correctly in the context of customer churn prediction.

10.4 Decision Tree:

The Decision Tree classifier is employed initially due to its interpretability, versatility in handling various data types, and inherent feature selection capabilities. Its transparency in decision-making and robustness to outliers and non-linear relationships make it an ideal starting point for model exploration. By leveraging Decision Trees, the code establishes a clear foundation for subsequent analysis and model refinement.

This code snippet executes a comprehensive process for training and evaluating a Decision Tree classifier while also visualizing feature importances. Firstly, it employs grid search cross-validation to identify the optimal hyperparameters for the Decision Tree model, such as maximum depth, minimum samples split, and criterion. Subsequently, the model is re-initialized with these best parameters and trained on resampled training data obtained through SMOTE to address class imbalance. Following model training, predictions are generated for both the training and testing datasets, allowing for the computation and printing of various evaluation metrics including accuracy, F1 score, recall, and precision. Moreover, feature importances are calculated from the trained model and presented visually using a bar plot, enabling insight into the relative importance of each feature in the model's predictions. This thorough approach ensures that the Decision Tree classifier is effectively trained, rigorously evaluated, and its predictive capabilities and feature importance are clearly understood.

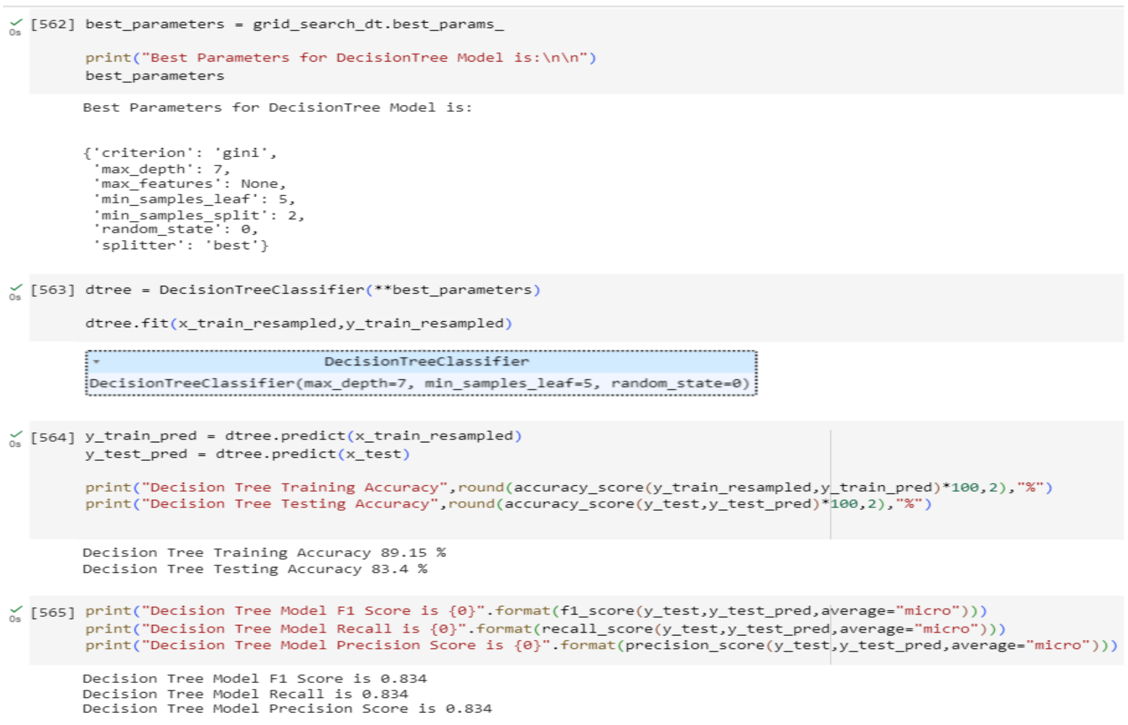


2.DecisionTree

```
[559] dtree = DecisionTreeClassifier()
```

```
[560] param_grid = {"max_depth":[3,4,5,6,7,8,9,10],
                    "min_samples_split":[2,3,4,5,6,7,8],
                    "min_samples_leaf":[1,2,3,4,5,6,7,8],
                    "criterion":["gini","entropy"],
                    "splitter":["best","random"],
                    "max_features":["auto",None],
                    "random_state":[0,42]}
```

```
[561] grid_search_dt = GridSearchCV(dtree, param_grid, cv=2, n_jobs=1)

      grid_search_dt.fit(x_train_resampled,y_train_resampled)
```



```
[562] best_parameters = grid_search_dt.best_params_

      print("Best Parameters for DecisionTree Model is:\n\n")
      best_parameters

      Best Parameters for DecisionTree Model is:


      {'criterion': 'gini',
       'max_depth': 7,
       'max_features': None,
       'min_samples_leaf': 5,
       'min_samples_split': 2,
       'random_state': 0,
       'splitter': 'best'}

[563] dtree = DecisionTreeClassifier(**best_parameters)

      dtree.fit(x_train_resampled,y_train_resampled)

                        DecisionTreeClassifier
      DecisionTreeClassifier(max_depth=7, min_samples_leaf=5, random_state=0)

[564] y_train_pred = dtree.predict(x_train_resampled)
      y_test_pred = dtree.predict(x_test)

      print("Decision Tree Training Accuracy",round(accuracy_score(y_train_resampled,y_train_pred)*100,2),"%")
      print("Decision Tree Testing Accuracy",round(accuracy_score(y_test,y_test_pred)*100,2),"%")

      Decision Tree Training Accuracy 89.15 %
      Decision Tree Testing Accuracy 83.4 %

[565] print("Decision Tree Model F1 Score is {0}".format(f1_score(y_test,y_test_pred,average="micro")))
      print("Decision Tree Model Recall is {0}".format(recall_score(y_test,y_test_pred,average="micro")))
      print("Decision Tree Model Precision Score is {0}".format(precision_score(y_test,y_test_pred,average="micro")))

      Decision Tree Model F1 Score is 0.834
      Decision Tree Model Recall is 0.834
      Decision Tree Model Precision Score is 0.834
```

- **Accuracy Score on Training Data (89.15%)**: This indicates that the Decision Tree classifier correctly predicts the class labels for 89.15% of the instances in the training dataset. A higher accuracy score suggests that the model effectively captures the underlying patterns and relationships present in the training data.

- **Accuracy Score on Testing Data (83.4%)**: The accuracy score on the testing data reflects the model's performance on unseen data. In this case, the model accurately predicts the class labels for 83.4% of the instances in the testing dataset. A high accuracy score on the testing data suggests that the model generalizes well and is robust to new data.

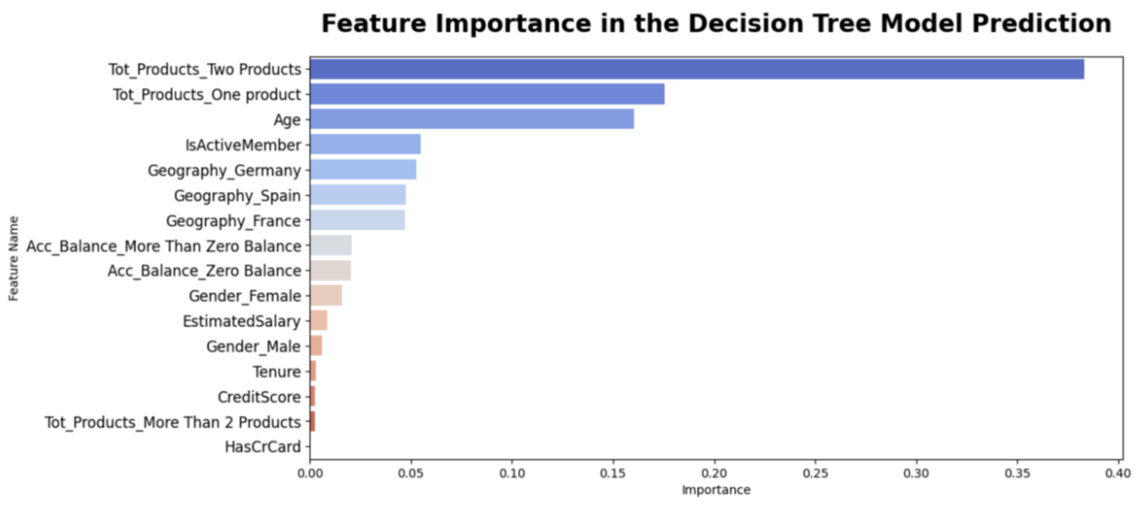
- **F1 Score of the Model (0.834)**: The F1 score is the harmonic mean of precision and recall, providing a balanced measure of the model's performance. In this context, an F1 score of 0.834 indicates that the model achieves a good balance between precision (ability to correctly identify positive cases) and recall (ability to capture all positive cases) on the testing data.

- **Recall Score of the Model (0.834)**: The recall score, also known as sensitivity or true positive rate, measures the proportion of actual positive cases that the model correctly identifies. A recall score of 0.834 indicates that the model effectively captures a large portion of the actual positive cases in the testing data.

- **Precision Score of the Model (0.834)**: The precision score measures the proportion of predicted positive cases that are correctly classified. A precision score of 0.834 indicates that the model correctly identifies a high percentage of positive cases among all instances predicted as positive, minimizing false positives.

10.4.1 Feature Importance:

This code snippet visualizes feature importances from a Decision Tree classifier. It constructs a DataFrame to organize feature names and their importances, sorts features by importance in descending order, and creates a horizontal bar plot using seaborn. The plot highlights key features influencing the model's predictions, such as "Total Products - Two Products," "Total Products - One Product," "Age," and "IsActiveMember." This aids in the interpretation of customer churn factors.



10.4.2 Confusion Matrix:

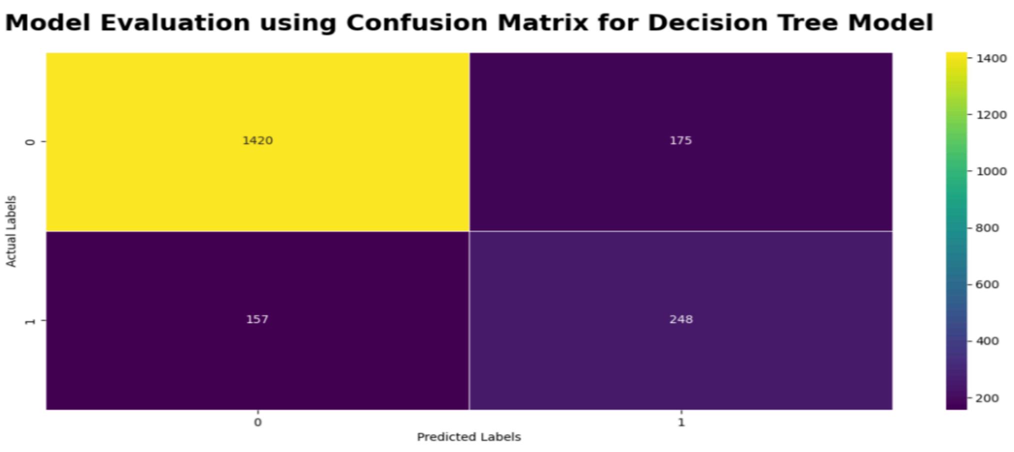
The confusion matrix provides detailed insights into the model's performance by summarizing the classification results.

- **True Negatives (Actual label 0, Predicted label 0)**: There are 1420 instances where the model correctly predicted negative outcomes (churn non-events).

- **False Negatives (Actual label 1, Predicted label 0)**: There are 157 instances where the model incorrectly predicted negative outcomes (churn events), missing these instances.

- **False Positives (Actual label 0, Predicted label 1)**: There are 175 instances where the model incorrectly predicted positive outcomes (churn events) when they were actually negative.

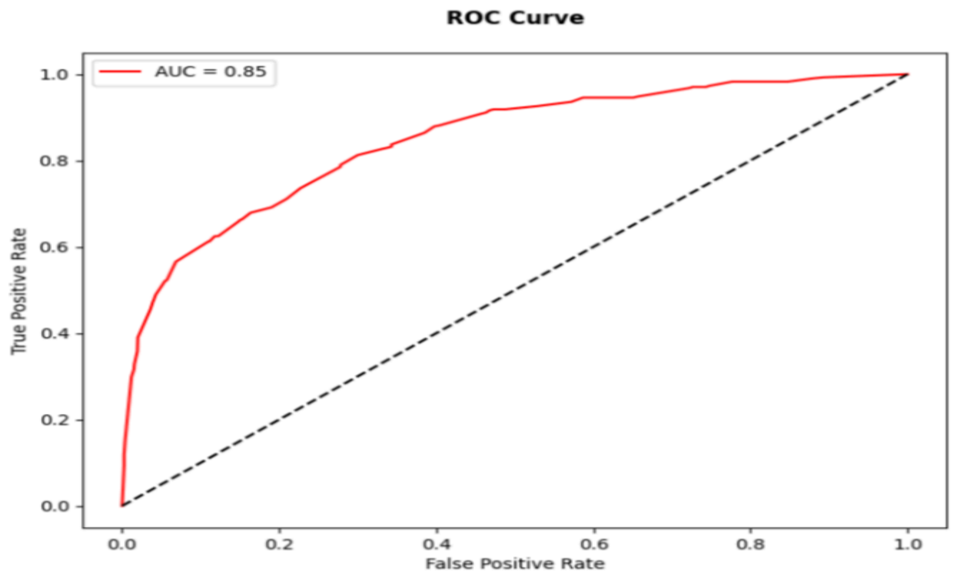
- **True Positives (Actual label 1, Predicted label 1)**: There are 248 instances where the model correctly predicted positive outcomes (churn events).



10.4.3 ROC Curve:

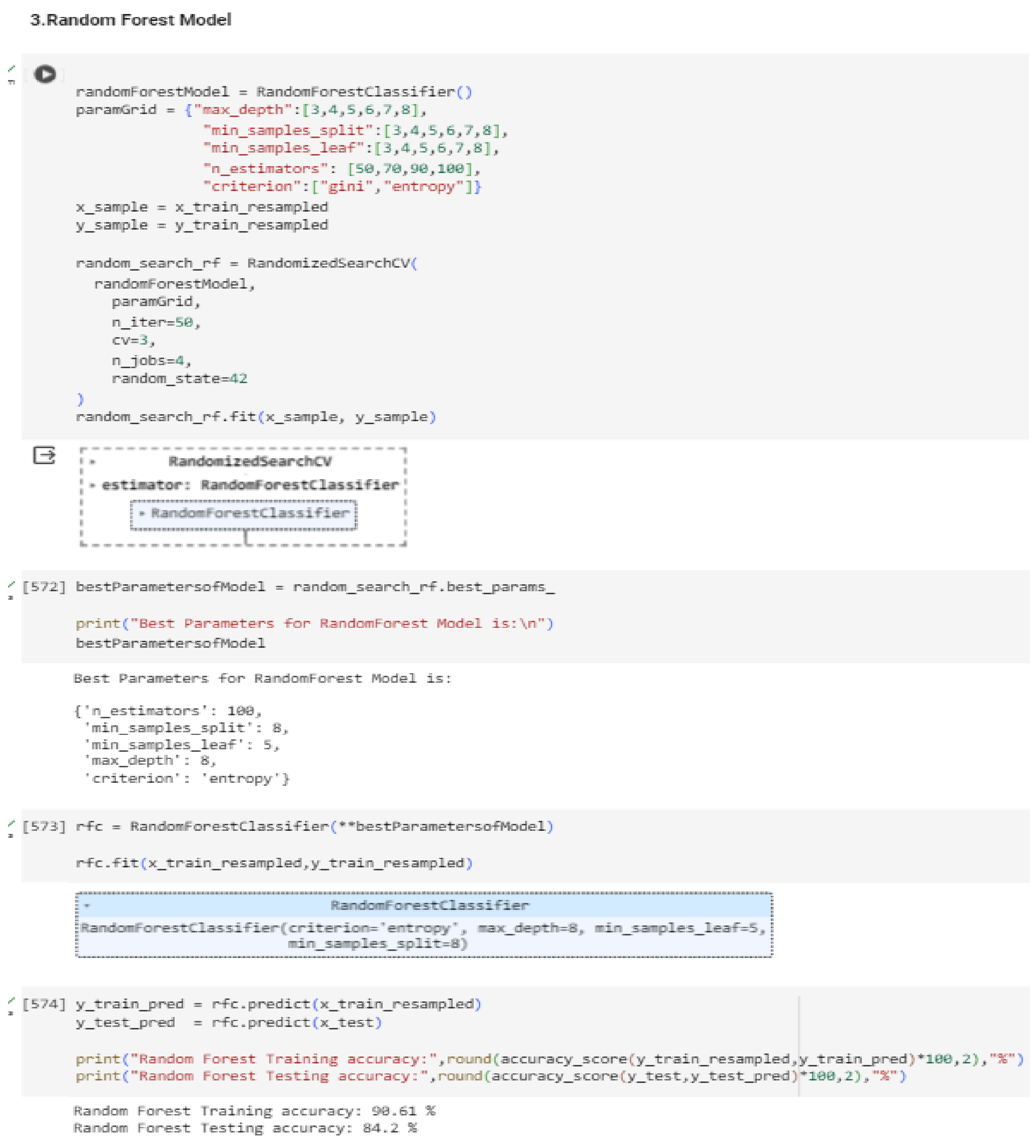
This code snippet constructs an ROC curve to evaluate the performance of the model in distinguishing between churn events and non-events. Initially, the model's predicted probabilities for churn events are obtained. These probabilities, along with the actual labels, are organized into a DataFrame for further analysis. Subsequently, the false positive rate (FPR), true positive rate (TPR), and corresponding thresholds are computed to generate the ROC curve. The area under the ROC curve (AUC) is calculated to quantify the model's discriminative ability, with a higher AUC indicating better performance. Finally, the ROC curve is plotted using Matplotlib, providing a visual representation of the model's sensitivity and specificity across different threshold values. This visualization offers valuable insights into the model's classification, performance and its ability to make accurate predictions.

With an AUC (Area Under the ROC Curve) of 0.85, the model demonstrates good discriminative ability in distinguishing between positive and negative instances (churn events and non-events). A higher AUC score suggests that the model performs well across various threshold values, effectively balancing the trade-off between true positive rate (sensitivity) and false positive rate (1-specificity). This indicates that the model has a strong capability to correctly classify instances, making it a promising tool for predicting customer churn.



10.5 Random Forest:

Random Forest is a versatile ensemble learning technique widely used for both classification and regression tasks in machine learning. It operates by constructing multiple decision trees during training and outputting the mode of the classes (classification) or the mean prediction (regression) of the individual trees. Random Forests are known for their robustness, scalability, and ability to handle high-dimensional datasets with a large number of features. They are particularly effective in handling non-linear relationships and capturing complex interactions between variables. In this context, the code snippet utilizes Randomized Search Cross Validation to optimize the Random Forest classifier's hyperparameters, enhancing its predictive performance. By systematically tuning the model's parameters, this approach aims to maximize the Random Forest's accuracy and robustness, ultimately improving its ability to predict customer churn accurately


3.Random Forest Model

```
randomForestModel = RandomForestClassifier()
paramGrid = {"max_depth":[3,4,5,6,7,8],
              "min_samples_split":[3,4,5,6,7,8],
              "min_samples_leaf":[3,4,5,6,7,8],
              "n_estimators": [50,70,90,100],
              "criterion":["gini","entropy"]}
x_sample = x_train_resampled
y_sample = y_train_resampled

random_search_rf = RandomizedSearchCV(
  randomForestModel,
    paramGrid,
    n_iter=50,
    cv=3,
    n_jobs=4,
    random_state=42
)
random_search_rf.fit(x_sample, y_sample)
```

```
RandomizedSearchCV
estimator: RandomForestClassifier
    RandomForestClassifier
```

```
[572] bestParametersofModel = random_search_rf.best_params_

      print("Best Parameters for RandomForest Model is:\n")
      bestParametersofModel
```

```
Best Parameters for RandomForest Model is:

{'n_estimators': 100,
 'min_samples_split': 8,
 'min_samples_leaf': 5,
 'max_depth': 8,
 'criterion': 'entropy'}
```

```
[573] rfc = RandomForestClassifier(**bestParametersofModel)

      rfc.fit(x_train_resampled,y_train_resampled)
```

```
                     RandomForestClassifier
RandomForestClassifier(criterion='entropy', max_depth=8, min_samples_leaf=5,
                       min_samples_split=8)
```

```
[574] y_train_pred = rfc.predict(x_train_resampled)
      y_test_pred  = rfc.predict(x_test)

      print("Random Forest Training accuracy:",round(accuracy_score(y_train_resampled,y_train_pred)*100,2),"%")
      print("Random Forest Testing accuracy:",round(accuracy_score(y_test,y_test_pred)*100,2),"%")
```

```
Random Forest Training accuracy: 90.61 %
Random Forest Testing accuracy: 84.2 %
```

- **Training Accuracy**: Achieving 90.61%, the model demonstrates high accuracy on the training data, indicating its capability to effectively capture the underlying patterns present in the data during the training phase.

- **Testing Accuracy**: With an accuracy of 84.2% on the testing data, the model showcases robustness and generalization ability, performing well on unseen data and suggesting its effectiveness in making accurate predictions beyond the training set.

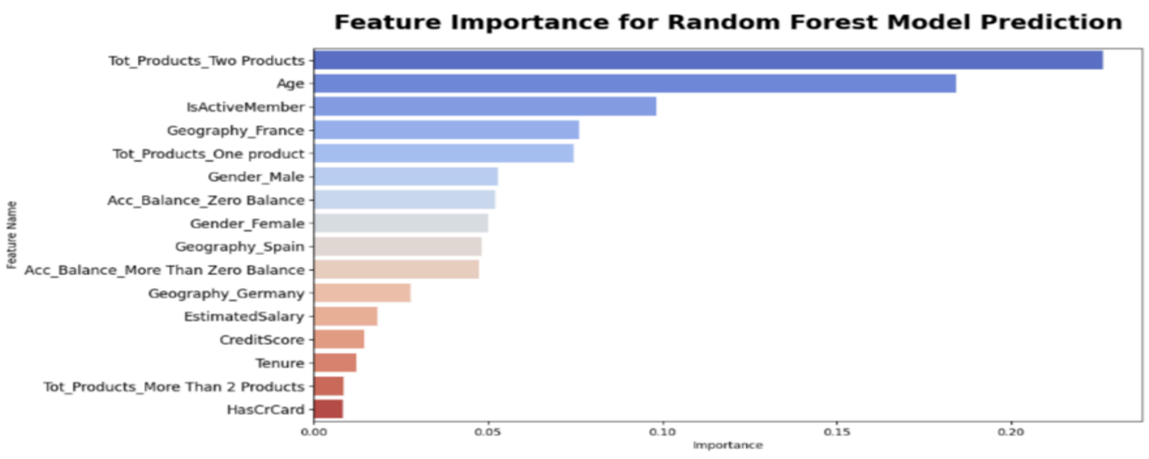
- **F1 Score**: The F1 score, calculated as 0.84, reflects a balanced harmonic mean of precision and recall. It demonstrates the model's ability to achieve a satisfactory balance between correctly identifying positive cases (churn events) and minimizing false positives.

- **Recall Score**: With a recall score of 0.844, the model effectively captures a high proportion of actual positive cases (churn events), indicating its sensitivity to detecting instances of interest.

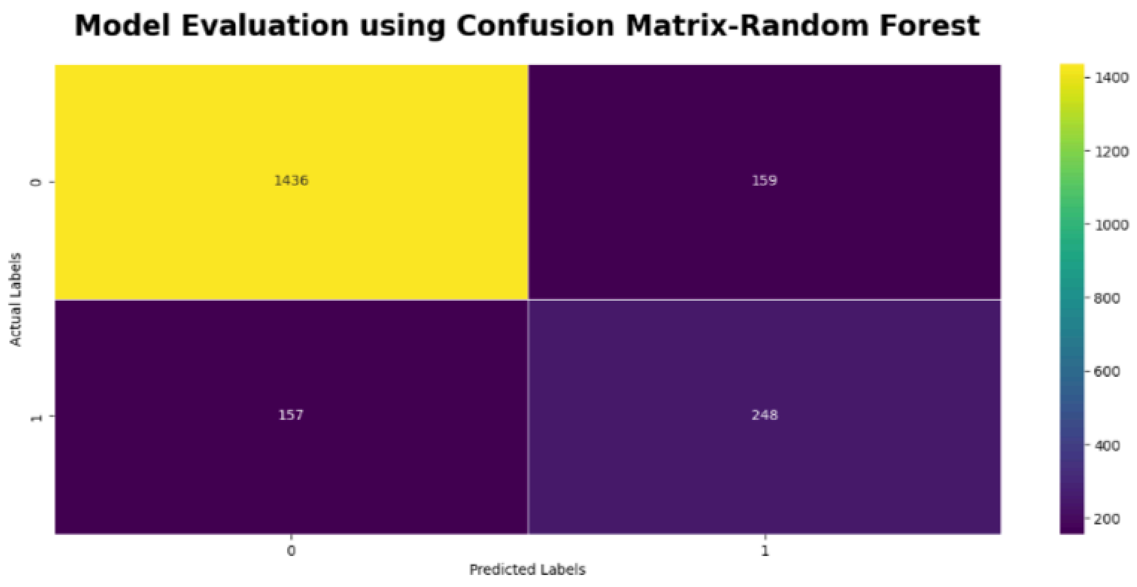
- **Precision Score:** Similarly, the precision score of 0.844 highlights the model's ability to correctly identify positive predictions (churn events) among all instances predicted as positive, minimizing false positives.

10.5.1 Feature Importance:

This code segment generates a bar plot visualizing the feature importances obtained from the trained Random Forest classifier. It begins by constructing a DataFrame (`impDf`) to store the feature names and their corresponding importances derived from the Random Forest model. The features are then sorted based on their importance scores in descending order, and a horizontal bar plot is created using seaborn's `barplot` function. The plot showcases the relative importance of each feature, with the x-axis representing the importance scores and the y-axis displaying the feature names. Notably, features such as "Total Products - Two Products," "Age," and "IsActiveMember" emerge with high importance, suggesting their significant influence on the model's predictions. This visualization aids in identifying key predictors impacting customer churn and informs feature selection strategies for enhancing model performance and interpretability.



10.5.2 Confusion Matrix:

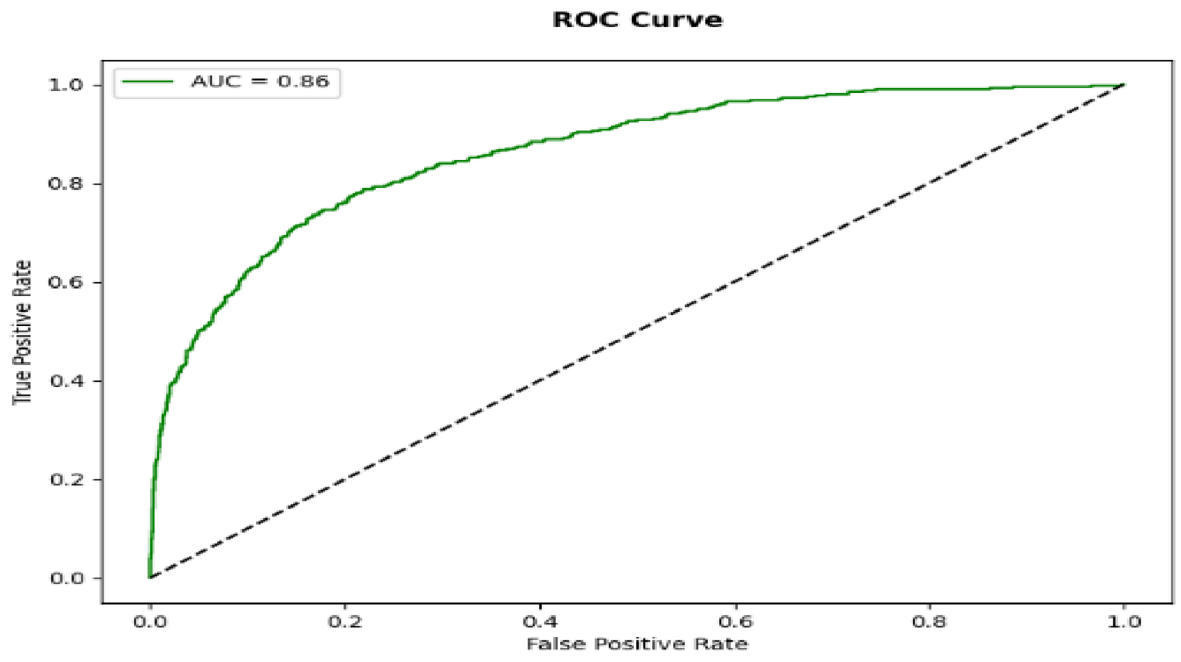


The provided confusion matrix results offer insights into the performance of the Random Forest model:

- **True Negatives (Actual label 0, Predicted label 0):** There are 1436 instances where the model correctly predicted negative outcomes (non-churn events).
- **False Negatives (Actual label 1, Predicted label 0):** There are 157 instances where the model incorrectly predicted negative outcomes (churn events), failing to identify them.
- **False Positives (Actual label 0, Predicted label 1):** There are 159 instances where the model incorrectly predicted positive outcomes (churn events) when they were actually negative.
- **True Positives (Actual label 1, Predicted label 1):** There are 248 instances where the model correctly predicted positive outcomes (churn events).

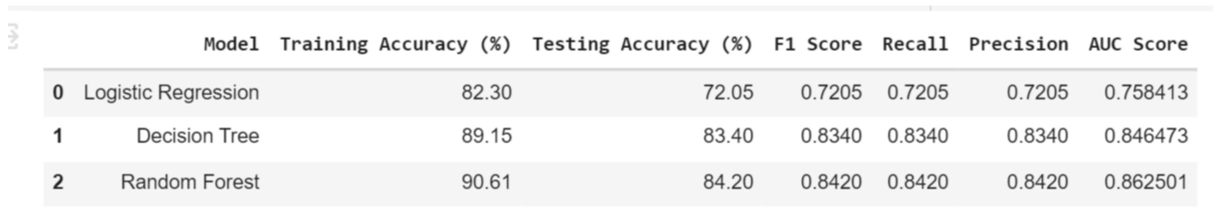
These results offer a detailed breakdown of the model's predictive performance, highlighting its ability to accurately classify instances of both churn and non-churn events. Further analysis of these metrics can inform strategies for model refinement and optimization.

10.5.3 ROC Curve:

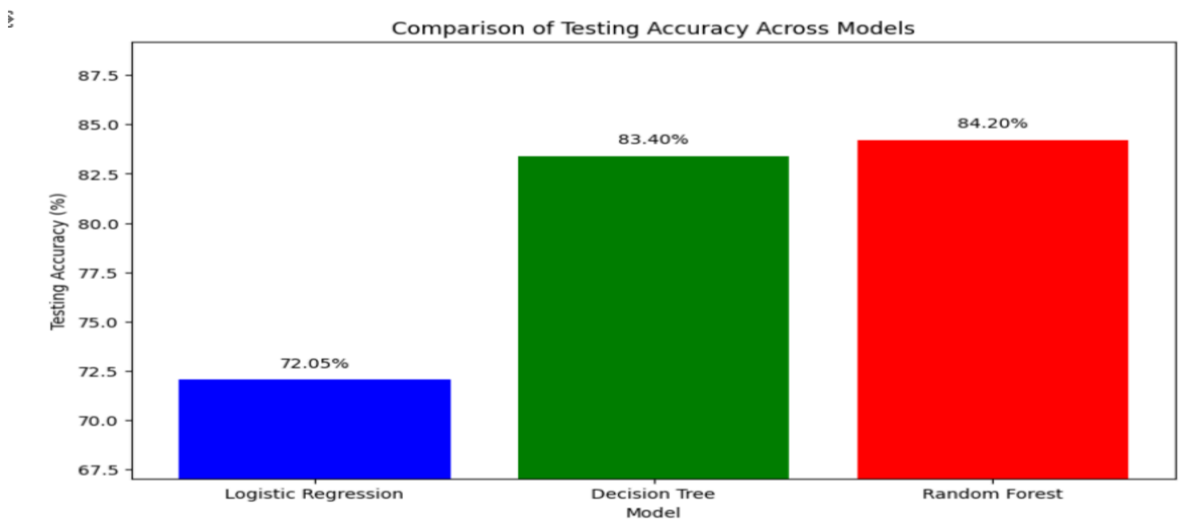


The provided code segment computes the ROC curve and AUC (Area Under the ROC Curve) to assess the Random Forest model's performance in predicting customer churn. With an AUC value of 0.86, the model demonstrates strong discriminative ability in distinguishing between positive and negative instances (churn events and non-events). A higher AUC score suggests that the model effectively balances the true positive rate and false positive rate across various threshold values, indicating its robustness and accuracy in classification. This evaluation provides valuable insights into the model's predictive performance and its ability to make accurate predictions of customer churn.

10.6 Results & Model Evaluation:



	Model	Training Accuracy (%)	Testing Accuracy (%)	F1 Score	Recall	Precision	AUC Score
0	Logistic Regression	82.30	72.05	0.7205	0.7205	0.7205	0.758413
1	Decision Tree	89.15	83.40	0.8340	0.8340	0.8340	0.846473
2	Random Forest	90.61	84.20	0.8420	0.8420	0.8420	0.862501



As we can see the Random Forest model emerges as the most effective for the customer churn prediction task based on the metrics from the comparison dataframe. It boasts the highest testing accuracy at 84.20%, suggesting superior generalization to new data. Its F1 score, recall, and precision all align at 0.8420, indicating a balanced performance in terms of both identifying true positives and the accuracy of the positive predictions. Furthermore, its AUC score of 0.8625 outperforms the other models, reflecting its stronger ability to discriminate between customers who will churn and those who won't.

11. Conclusion:

The customer churn prediction project effectively utilized a dataset from a banking environment to develop predictive models aimed at identifying factors leading to customer churn. Through comprehensive data preprocessing and feature engineering, the data was optimally prepared for modeling, allowing for the exploration of various predictive models. These included Logistic Regression, Decision Trees, and Random Forest, each providing valuable insights. The Random Forest model proved superior, showcasing high accuracy, robustness, and an ability to manage complex interactions between features, making it a vital tool for proactive customer retention strategies.

With an accuracy of 84.2%, an F1 score of 0.84, and an AUC of 0.86, the Random Forest model demonstrated excellent discriminative power and a balanced approach in identifying likely churners while minimizing false positives. This performance indicates the model's reliability in assisting the bank to strategically intervene and potentially retain at-risk customers. The project highlights the importance of data-driven approaches in enhancing customer retention efforts in the banking sector, suggesting further exploration into advanced modeling techniques and deeper feature engineering could provide even greater insights and improved outcomes.