MACHINE LEARNING REINFORCEMENT

BANK CUSTOMER CHURN PREDICTION

NAME	**KAVIYA C**
DOAMIN	**DADS[JULY]**
DATE	**10 - 11 – 2025**

PROJECT OVERVIEW

The main objective of this project is to predict whether a bank customer is likely to churn based on their demographic and financial details. The dataset includes features such as credit score, age, tenure, account balance, number of products, and activity status. Data preprocessing techniques like Label Encoding for categorical variables and StandardScaler for numerical feature standardization are applied to prepare the data for modeling. Several Machine Learning algorithms including Logistic Regression, Support Vector Machine (SVM), K-Nearest Neighbors (KNN), Decision Tree, Random Forest, and XGBoost are implemented to classify customers as likely to churn or remain loyal.

Each model's performance is evaluated using key metrics such as accuracy, precision, recall, and F1-score to identify the most effective one. Among all models, the Random Forest classifier provides the best results in terms of accuracy and overall performance. The trained Random Forest model is then deployed using Streamlit in PyCharm, allowing users to input customer details and receive instant churn predictions. This project demonstrates how machine learning can help banks proactively identify customers at risk of leaving and take necessary steps to improve customer retention and satisfaction.

DATA LOADING AND UNDERSTANDING

The dataset contains customer information related to banking behavior and churn prediction. It includes columns such as Customer ID which is a unique identifier for each customer, Credit Score which represents the customer's creditworthiness, Country and Gender which are categorical features, Age and Tenure indicating customer demographics and relationship duration, Balance and Estimated Salary reflecting financial details, Number of Products, Credit Card, and Active Member showing customer engagement with the bank, and Churn which is the target variable indicating whether the customer has left the bank.

Importing the libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

read_csv() – This function from the pandas library is used to load the dataset from a CSV file into a Data Frame for analysis.

```
df = pd.read_csv('/content/Bank Customer Churn.csv')
```

head() – This function displays the first few rows of the dataset, helping to understand the structure and sample records.

```
df.head()
```

	customer_id	credit_score	country	gender	age	tenure	balance	products_number	credit_card	active_me
0	15634602	619	France	Female	42	2	0.00	1	1	
1	15647311	608	Spain	Female	41	1	83807.86	1	0	
2	15619304	502	France	Female	42	8	159660.80	3	1	
3	15701354	699	France	Female	39	1	0.00	2	0	
4	15737888	850	Spain	Female	43	2	125510.82	1	1	

info() – This function provides details about each column, including data types and the number of non-null values, helping identify missing data or incorrect data types.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 12 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   customer_id       10000 non-null  int64  
 1   credit_score      10000 non-null  int64  
 2   country           10000 non-null  object 
 3   gender            10000 non-null  object 
 4   age               10000 non-null  int64  
 5   tenure            10000 non-null  int64  
 6   balance           10000 non-null  float64
 7   products_number   10000 non-null  int64  
 8   credit_card       10000 non-null  int64  
 9   active_member     10000 non-null  int64  
 10  estimated_salary  10000 non-null  float64
 11  churn             10000 non-null  int64  
dtypes: float64(2), int64(8), object(2)
memory usage: 937.6+ KB
```

describe() – This function gives summary statistics such as mean, minimum, maximum, and standard deviation for numerical columns, helping understand data distribution.

```
df.describe()
```

	customer_id	credit_score	age	tenure	balance	products_number	credit_card	a
count	1.000000e+04	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.00000	
mean	1.569094e+07	650.528800	38.921800	5.012800	76485.889288	1.530200	0.70550	
std	7.193619e+04	96.653299	10.487806	2.892174	62397.405202	0.581654	0.45584	
min	1.556570e+07	350.000000	18.000000	0.000000	0.000000	1.000000	0.00000	
25%	1.562853e+07	584.000000	32.000000	3.000000	0.000000	1.000000	0.00000	
50%	1.569074e+07	652.000000	37.000000	5.000000	97198.540000	1.000000	1.00000	
75%	1.575323e+07	718.000000	44.000000	7.000000	127644.240000	2.000000	1.00000	
max	1.581569e+07	850.000000	92.000000	10.000000	250898.090000	4.000000	1.00000	

columns – This attribute lists all the column names in the dataset, useful for identifying features and target variables.

```
df.columns
```

```
Index(['customer_id', 'credit_score', 'country', 'gender', 'age', 'tenure',
       'balance', 'products_number', 'credit_card', 'active_member',
       'estimated_salary', 'churn'],
      dtype='object')
```

DATA CLEANING

The Data Cleaning process includes the dataset was thoroughly examined to ensure it was accurate and ready for analysis. Missing or null values were checked and handled appropriately to maintain data quality. The Customer ID column was removed since it only served as a unique identifier and did not provide any meaningful information for analysis. Categorical features such as Gender and Country were converted into numerical values to make the dataset compatible with machine learning models. The Gender column was mapped into numeric codes, while the Country column was transformed using a label encoder. These steps ensured that the dataset was clean, consistent, and properly formatted for further modeling and evaluation

isnull() – These functions are used to check for missing or null values in the dataset, helping identify incomplete data that needs to be handled

```
df.isnull().sum()
```

	0
customer_id	0
credit_score	0
country	0
gender	0
age	0
tenure	0
balance	0
products_number	0
credit_card	0
active_member	0
estimated_salary	0
churn	0

drop() – This function is used to remove unnecessary columns or rows from the dataset. The Customer ID column was dropped since it did not contribute to analysis or prediction.

```
df.drop('customer_id', axis=1, inplace=True)
```

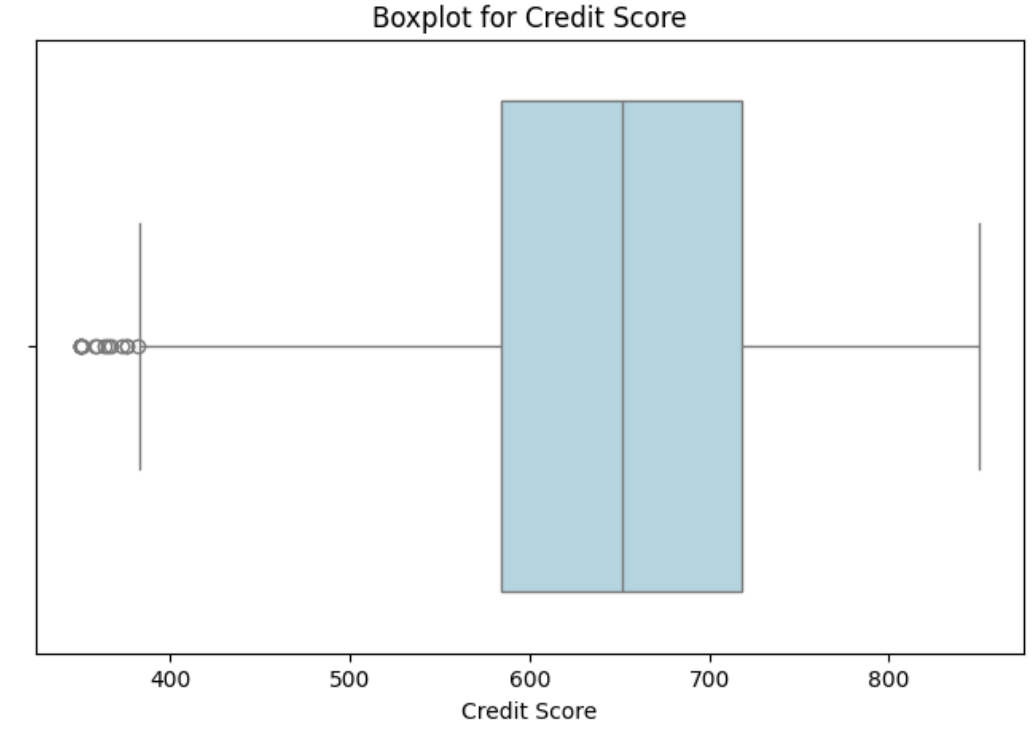
map() – The map function was used to convert the Gender column into numeric values, for example mapping “Male” to 1 and “Female” to 0, making it suitable for machine learning models.

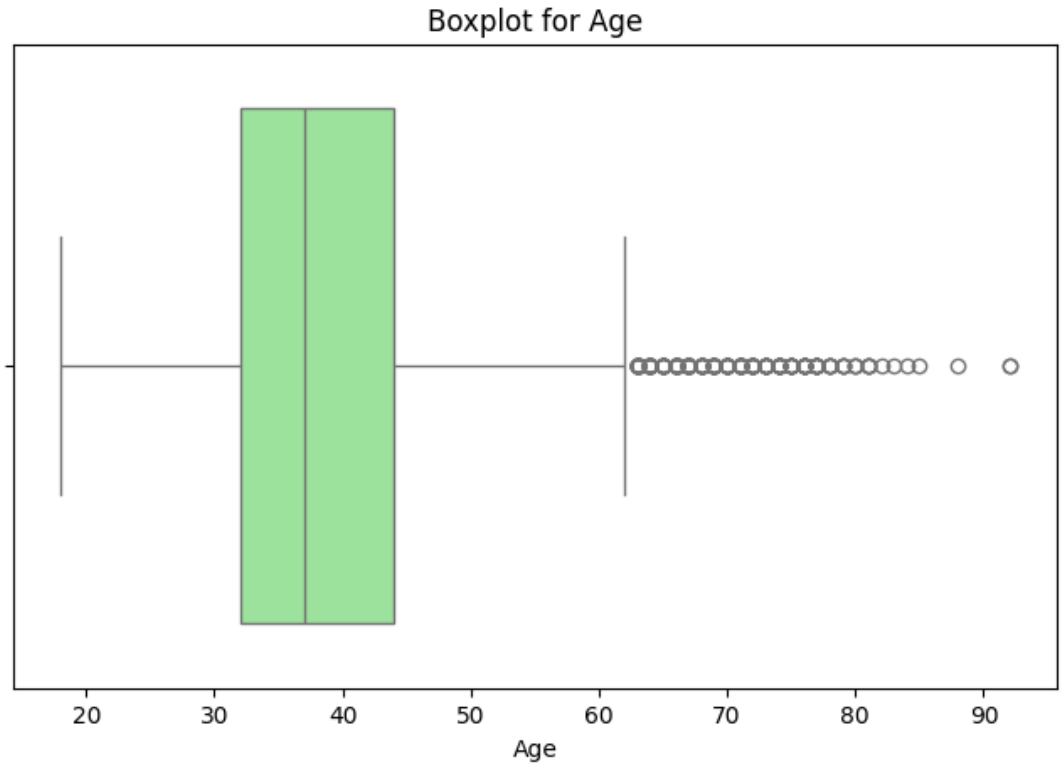
```
df['gender'] = df['gender'].replace({'Male': 0, 'Female': 1})

country_mapping = {'France': 0, 'Germany': 1, 'Spain': 2}
df['country'] = df['country'].map(country_mapping)
```

OUTLIER DETECTION AND HANDLING

Outliers were identified in the Credit Score, Age, and Number of Products columns using the boxplot() function, which helped visualize extreme values beyond the normal range. After detection, the Interquartile Range (IQR) method was applied to remove these outliers. The IQR was calculated as the difference between the third quartile (Q3) and the first quartile (Q1), and any data points lying outside 1.5 times the IQR below Q1 or above Q3 were removed. This process helped maintain the accuracy and consistency of the dataset by eliminating extreme and unrealistic values.

boxplot() – This function is used to visually identify outliers in numerical columns by displaying data distribution and extreme values beyond the whiskers of the plot.



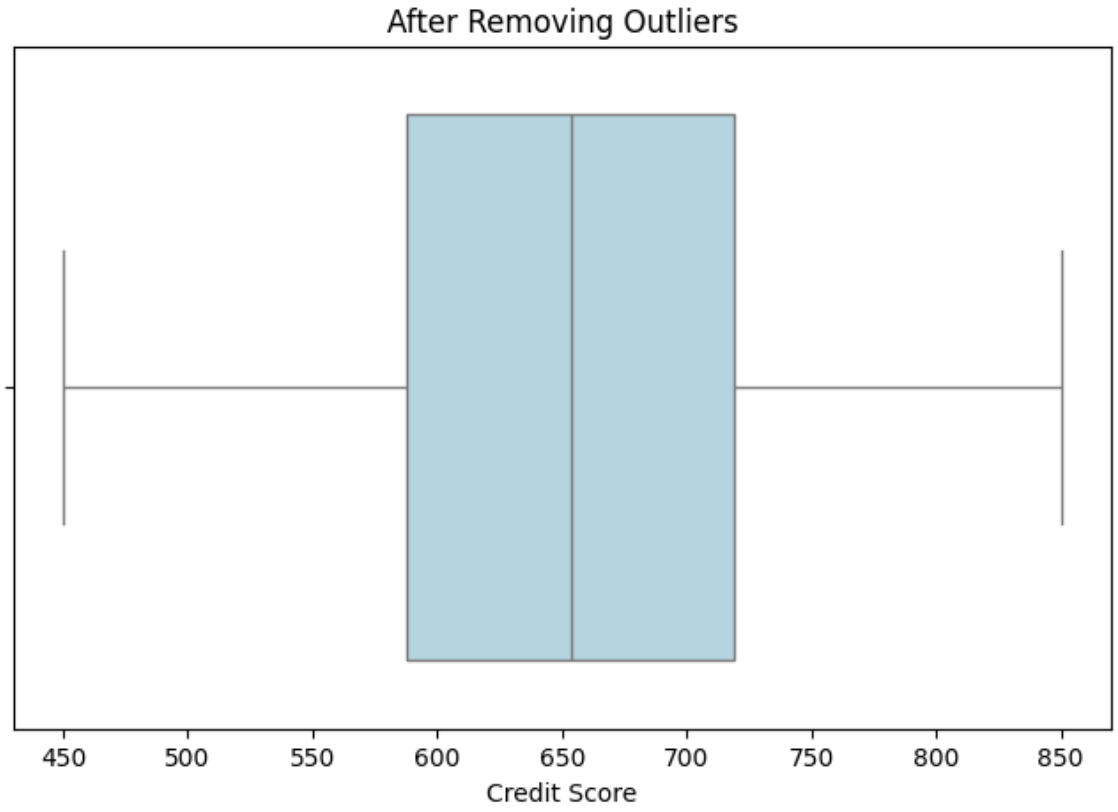


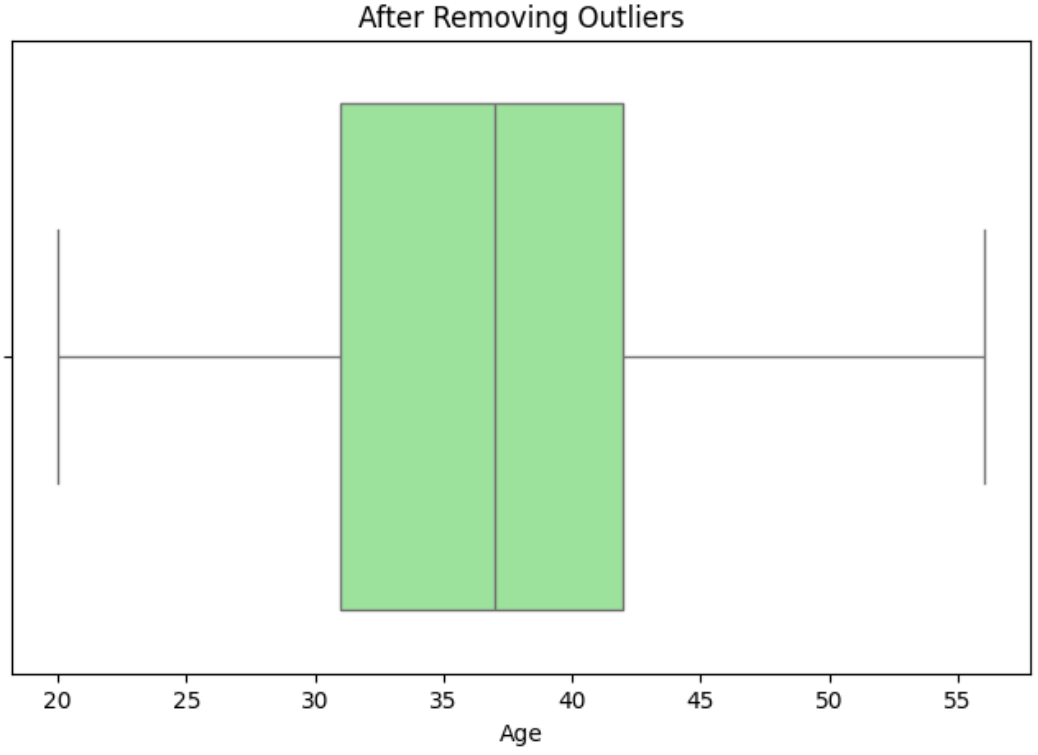
IQR (Interquartile Range) – This statistical technique identifies outliers by calculating the range between the first and third quartiles (Q1 and Q3). Data points outside 1.5 times the IQR are considered outliers and can be removed or adjusted.

```
Q1 = df['credit_score'].quantile(0.25)
Q3 = df['credit_score'].quantile(0.75)
IQR = Q3 - Q1
lower_limit = Q1 - 1.0 * IQR
upper_limit = Q3 + 1.0 * IQR

df = df[(df['credit_score'] >= lower_limit) & (df['credit_score'] <= upper_limit)]
```

```
Q1_age = df['age'].quantile(0.25)
Q3_age = df['age'].quantile(0.75)
IQR_age = Q3_age - Q1_age
lower_limit_age = Q1_age - 1.0 * IQR_age
upper_limit_age = Q3_age + 1.0 * IQR_age

df = df[(df['age'] >= lower_limit_age) & (df['age'] <= upper_limit_age)]
```

AFTER REMOVING OUTLIERS



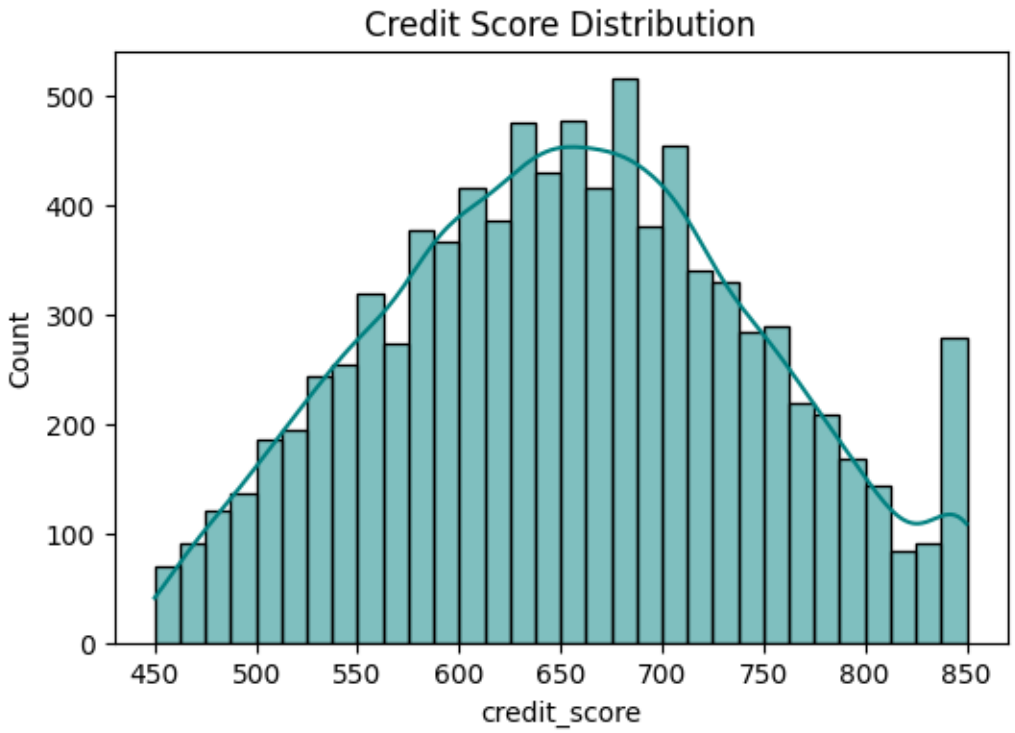


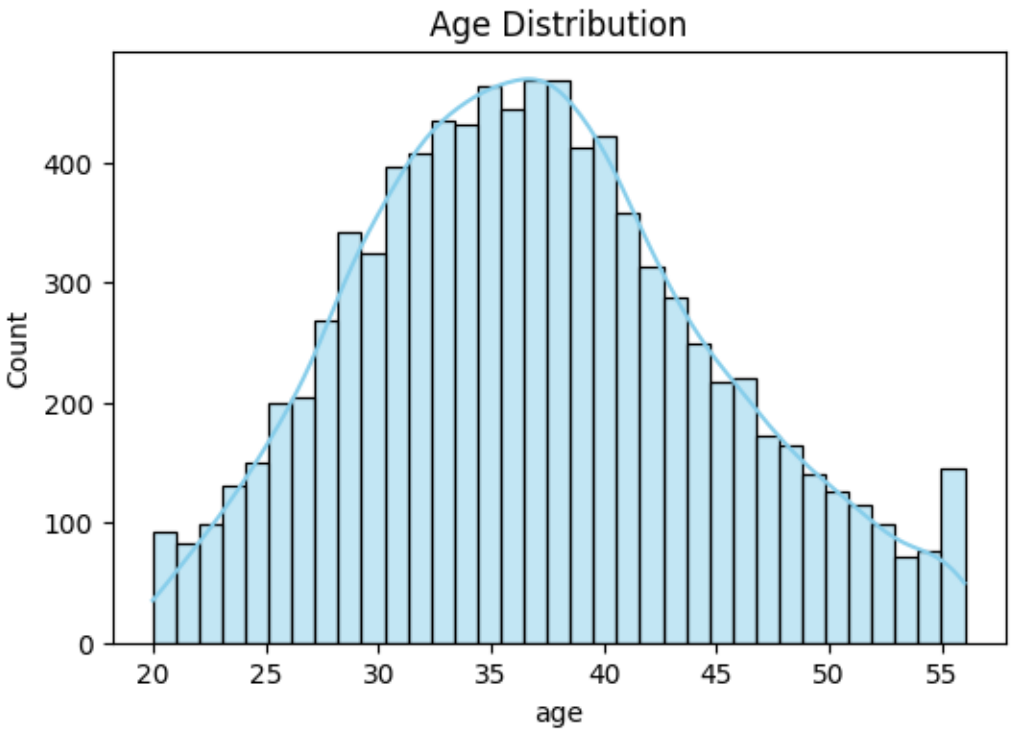
EXPLORATORY DATA ANALYSIS

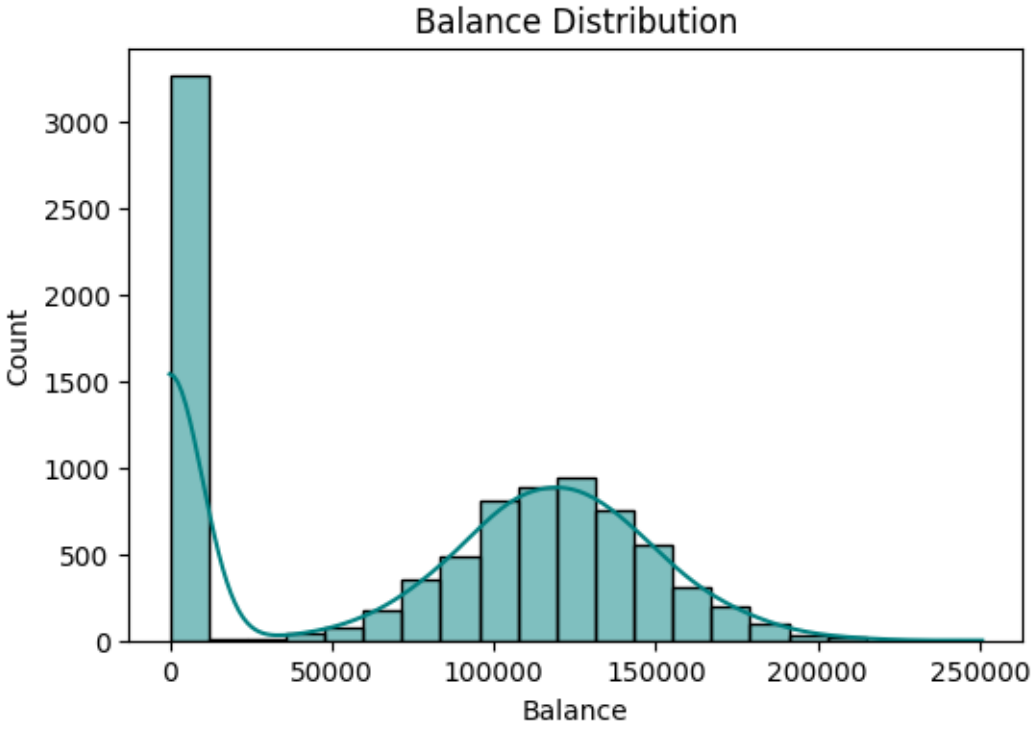
Exploratory Data Analysis (EDA) is the process of examining and understanding a dataset before building any model. It helps to identify patterns, trends, and relationships between variables, detect missing values or outliers, and gain insights into the data's overall structure. In simple terms, EDA helps you get to know your data better using graphs, charts, and summary statistics before performing deeper analysis or predictions

UNIVARIATE ANALYSIS

This analysis focused on understanding individual features in the dataset. Numerical columns such as Credit Score, Age, Balance, and Estimated Salary were visualized using histograms and boxplots to observe their distribution, spread, and presence of any skewness or outliers. Categorical columns like Gender, Country, and Active Member were analyzed using **count plots** to understand the frequency of each category. This helped in understanding the overall data composition and identifying any imbalance.

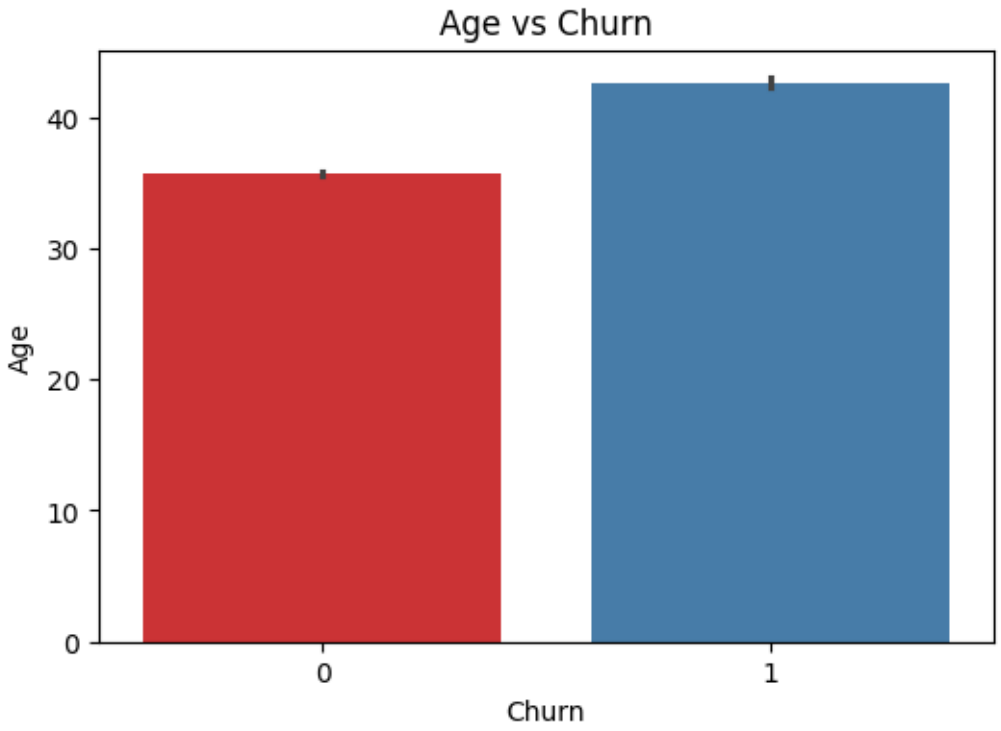


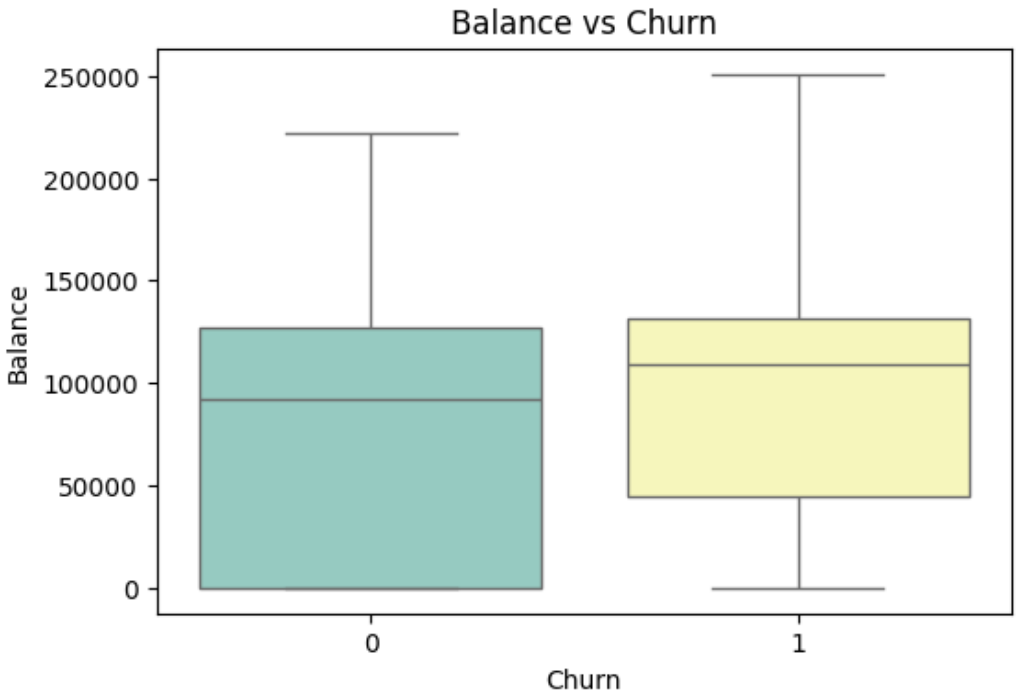

Age Distribution
Count
0
100
200
300
400
20
25
30
35
40
45
50
55
age

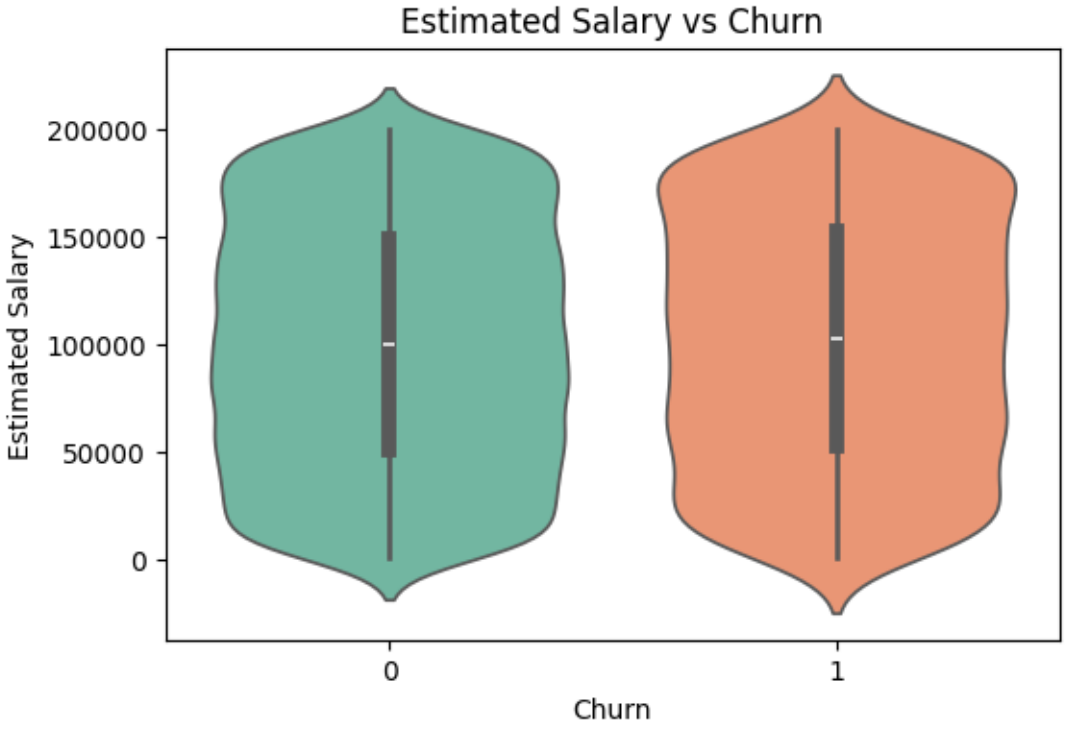

Balance Distribution
Count
0
500
1000
1500
2000
2500
3000
0
50000
100000
150000
200000
250000
Balance

BIVARIATE ANALYSIS

This analysis relationships between two variables were explored to understand how independent features affect the target variable, Churn. Bar charts and boxplots were used to compare how churn rates vary with different features such as Age, Balance, and Credit Score. This helped identify significant patterns like which customer groups are more likely to churn.

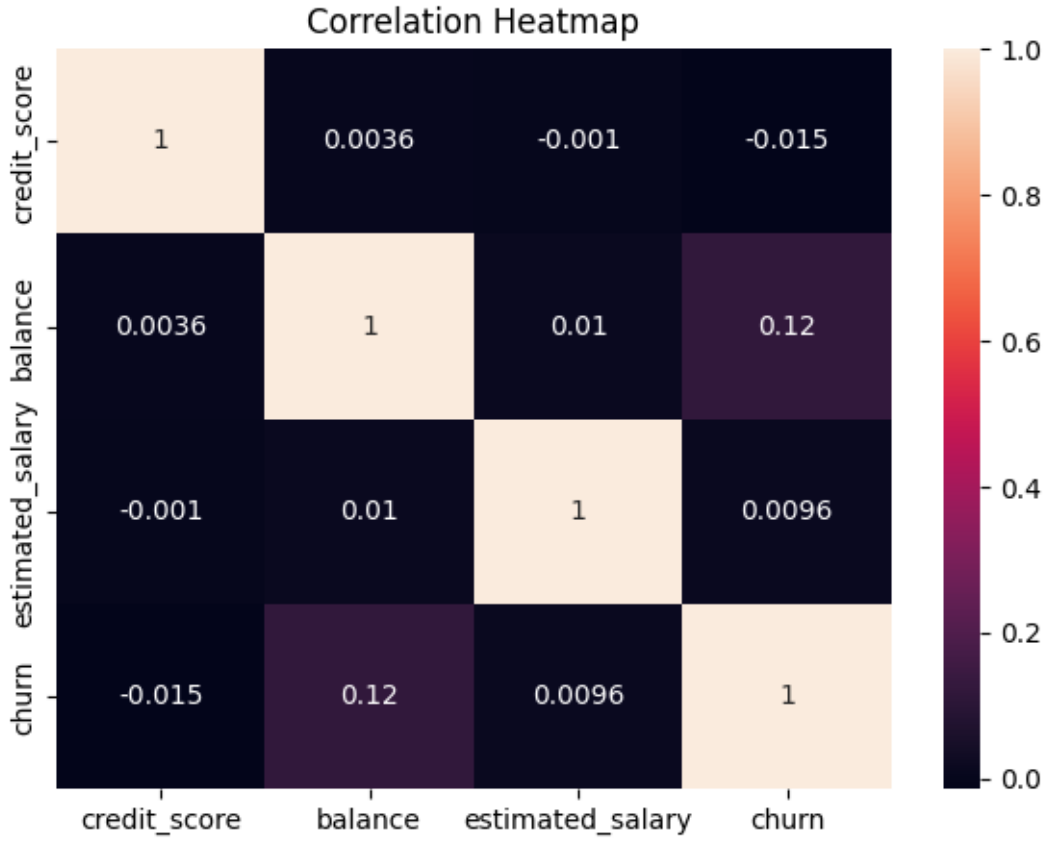



Balance vs Churn
250000
200000
150000
100000
50000
0
Balance
0
1
Churn


Estimated Salary vs Churn
200000
150000
100000
50000
0
Estimated Salary
0
1
Churn

MULTI VARIATE ANALYSIS

Correlation Heatmap – A heatmap was created using the correlation matrix to visualize the relationships between numerical variables. It helped identify strong positive or negative correlations among features such as Credit Score, Age, Balance, and Estimated Salary. This step provided useful insights for feature selection and understanding how different attributes are related to customer churn.



DATA PREPROCESSING

The dataset was prepared for model training and evaluation. The features (X) and target variable (y) were separated, where X included all independent variables such as Credit Score, Age, Balance, and Estimated Salary, and y represented the Churn column. The data was then split into training and testing sets, typically using an 80-20 ratio, to train the model on one part and test its performance on unseen data. Numerical features were scaled or normalized using techniques like Standard Scaler to ensure all variables were on a similar scale, improving model accuracy and performance.

```
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
```

```
label_encoders = {}
for column in df.select_dtypes(include='object').columns:
    le = LabelEncoder()
    df[column] = le.fit_transform(df[column])
    label_encoders[column] = le
```

```
features = df.drop(columns = ['churn'], axis=1)
target = df['churn']
```

```
scaler = StandardScaler()
scaler.fit(features)
```

```
StandardScaler
StandardScaler()
```

```
standarized_data = scaler.transform(features)
```

```
features = standarized_data
target = df['churn']
```

```
x_train,x_test,y_train,y_test = train_test_split(features,target,test_size=0.2,random_state=42)
```

MODEL TRAINING

The model training phase includes the multiple machine learning algorithms were applied to predict customer churn and compare their performance. Models such as Logistic Regression, Support Vector Machine (SVM), K-Nearest Neighbors (KNN), Decision Tree, Random Forest, and XGBoost were trained using the prepared training dataset. Each model learned patterns and relationships between the features and the target variable. After training, the models were evaluated on the testing set using performance metrics such as accuracy, precision, recall, and F1-score to assess their prediction capability. This step helped identify the most efficient and reliable model for churn prediction.

1.LOGISTIC REGRESSION

Logistic Regression is a simple and effective classification algorithm that predicts the probability of a customer churning or staying. It uses a linear relationship between the input features and the target variable and applies a sigmoid function to output values between 0 and 1 for classification.

Logistic Regression

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
```

```
classifier = LogisticRegression()
classifier.fit(x_train,y_train)
```

```
▾ LogisticRegression ⓘ ⓘ
LogisticRegression()
```

```
y_pred_lr = classifier.predict(x_test)
accuracy = accuracy_score(y_test,y_pred_lr)
print('Accuracy score : ',accuracy)
```

```
Accuracy score :  0.8229744728079911
```

```
cm = confusion_matrix(y_test,y_pred_lr)
cf = classification_report(y_test,y_pred_lr)
print('\n Confusion Matrix :\n',cm)
print('\n Classification Report : \n',cf)
```

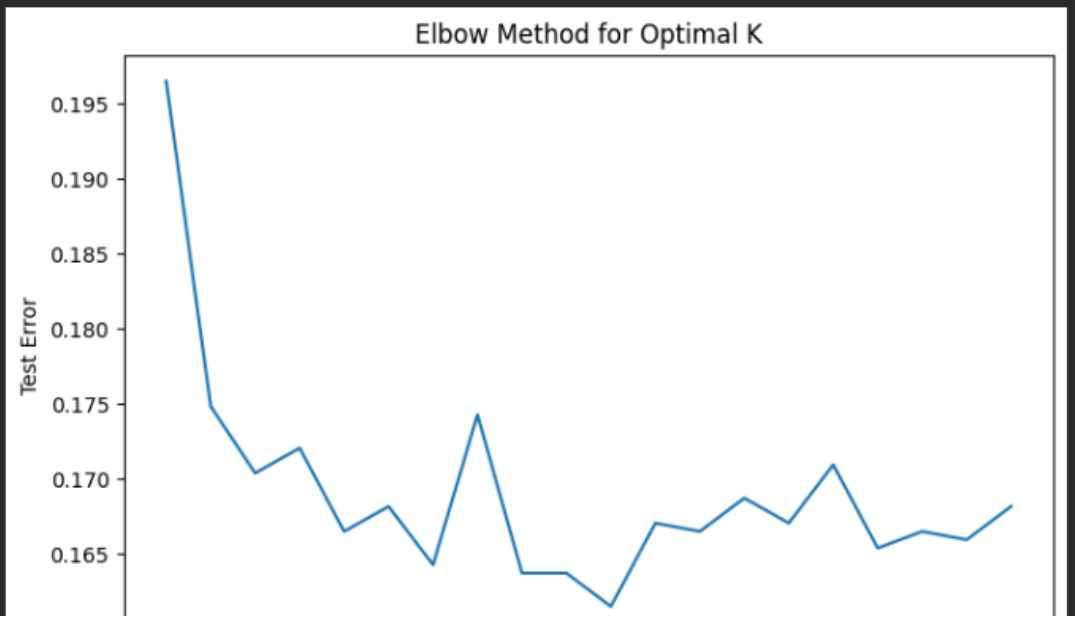
```
 Confusion Matrix :
 [[1397   48]
 [ 271   86]]

 Classification Report :
              precision    recall  f1-score   support

           0       0.84      0.97      0.90      1445
           1       0.64      0.24      0.35       357

    accuracy                           0.82      1802
   macro avg       0.74      0.60      0.62      1802
weighted avg       0.80      0.82      0.79      1802
```

2. K-NEAREST NEIGHBORS (KNN)

KNN predicts churn by comparing a customer with its nearest neighbors based on feature similarity. It assigns the class most common among the nearest data points. KNN is simple, non-parametric, and effective when patterns in data depend on proximity or similarity.

```python
from sklearn.neighbors import KNeighborsClassifier as KNN
from sklearn.metrics import accuracy_score,accuracy_score,classification_report
```

```python
test_error = []
K = range(1, 21)

for k in K:
    knn = KNN(n_neighbors=k)
    knn.fit(x_train, y_train)
    y_pred_knn = knn.predict(x_test)
    error = 1 - accuracy_score(y_test, y_pred_knn)
    test_error.append(error)



plt.figure(figsize=(8, 5))
plt.plot(K, test_error)
plt.title('Elbow Method for Optimal K')
plt.xlabel('Number of Neighbors')
plt.ylabel('Test Error')
plt.show()
```



```
best_k_index = test_error.index(min(test_error))
best_k = K[best_k_index]
best_accuracy = 1 - test_error[best_k_index]
print("Best K Value:", best_k)
print("Best Accuracy:", round(best_accuracy, 3))
```

```
Best K Value: 11
Best Accuracy: 0.839
```

```
knn_final = KNN(n_neighbors=best_k)
knn_final.fit(x_train, y_train)
```

```
KNeighborsClassifier(n_neighbors=11)
```

```
y_pred = knn_final.predict(x_test)
print("Accuracy  Score : ", accuracy_score(y_test, y_pred_knn))
```

```
Accuracy  Score :  0.8318534961154272
```

```
print("\n Confusion Matrix:\n", confusion_matrix(y_test, y_pred_knn))
print("\n Classification Report:\n", classification_report(y_test, y_pred_knn))
```

```
 Confusion Matrix:
 [[1424   21]
 [ 282   75]]

 Classification Report:
               precision    recall  f1-score   support

           0       0.83      0.99      0.90      1445
           1       0.78      0.21      0.33       357

    accuracy                           0.83      1802
   macro avg       0.81      0.60      0.62      1802
weighted avg       0.82      0.83      0.79      1802
```

3. SUPPORT VECTOR MACHINE (SVM)

SVM classifies customers by finding the optimal hyperplane that best separates churners from non-churners. It works well in high-dimensional spaces and uses kernel functions to handle non-linear data, making it powerful for identifying clear boundaries between two categories.

```
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
```

```
svc = SVC()
svc.fit(x_train, y_train)
```

```
SVC
SVC()
```

```
y_pred_svc = svc.predict(x_test)
accuracy = accuracy_score(y_test,y_pred_svc)
print('Accuracy score : ',accuracy)
```

```
Accuracy score :  0.8473917869034406
```

```
cm = confusion_matrix(y_test,y_pred_svc)
cf = classification_report(y_test,y_pred_svc)
print('\n Confusion Matrix :\n',cm)
print('\n Classification Report : \n',cf)
```

```
 Confusion Matrix :
 [[1410   35]
 [ 240  117]]

 Classification Report :
               precision    recall  f1-score   support

           0       0.85      0.98      0.91      1445
           1       0.77      0.33      0.46       357

    accuracy                           0.85      1802
   macro avg       0.81      0.65      0.69      1802
weighted avg       0.84      0.85      0.82      1802
```

4. DECISION TREE

Decision Tree is a rule-based algorithm that splits data into branches using feature conditions. It makes predictions through a series of if-else decisions, making it easy to interpret. It can handle both categorical and numerical data and works well for understanding key influencing factors.

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

```
dt  = DecisionTreeClassifier(random_state = 42)
dt.fit(x_train,y_train)
```

```
DecisionTreeClassifier(random_state=42)
```

```
y_pred_dt = dt.predict(x_test)
accuracy = accuracy_score(y_test,y_pred_dt)
print('Accuracy score : ',accuracy)
```

```
Accuracy score :  0.7902330743618202
```

```
cm = confusion_matrix(y_test,y_pred_dt)
cf = classification_report(y_test,y_pred_dt)
print('\n Confusion Matrix :\n',cm)
print('\n Classification Report : \n',cf)
```

```
 Confusion Matrix :
 [[1262  183]
 [ 195  162]]

 Classification Report :
               precision    recall  f1-score   support

           0       0.87      0.87      0.87      1445
           1       0.47      0.45      0.46       357

    accuracy                           0.79      1802
   macro avg       0.67      0.66      0.67      1802
weighted avg       0.79      0.79      0.79      1802
```

5. RANDOM FOREST

Random Forest combines multiple decision trees to improve accuracy and reduce overfitting. Each tree makes a prediction, and the final result is based on majority voting. It is robust, handles large datasets efficiently, and performs well on both classification and regression problems.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
```

```
rf = RandomForestClassifier()
rf.fit(x_train,y_train)
```

```
▾ RandomForestClassifier ❶ ❷
RandomForestClassifier()
```

```
y_pred_rf = rf.predict(x_test)
accuracy = accuracy_score(y_test,y_pred_rf)
print("Accuracy Score : ",accuracy)
```

```
Accuracy Score :  0.8518312985571587
```

```
cm = confusion_matrix(y_test,y_pred_rf)
cf = classification_report(y_test,y_pred_rf)
print('\n Confusion Matrix :\n',cm)
print('\n Classification Report : \n',cf)
```

```
 Confusion Matrix :
 [[1388   57]
 [ 210  147]]

 Classification Report :
              precision    recall  f1-score   support

           0       0.87      0.96      0.91      1445
           1       0.72      0.41      0.52       357

    accuracy                           0.85      1802
   macro avg       0.79      0.69      0.72      1802
weighted avg       0.84      0.85      0.84      1802
```

6. XGBOOST

XGBoost (Extreme Gradient Boosting) is an advanced boosting algorithm that builds multiple trees sequentially, where each new tree corrects the errors of the previous ones. It is highly efficient, scalable, and known for achieving high accuracy in classification and prediction tasks.

```
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
```

```
xgb = XGBClassifier(eval_metric='logloss')
xgb.fit(x_train, y_train)
```

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric='logloss',
              feature_types=None, feature_weights=None, gamma=None,
              grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=None, n_jobs=None,
```

```
y_pred_xgb = xgb.predict(x_test)
accuracy = accuracy_score(y_test,y_pred_xgb)
print("Accuracy Score :",accuracy)
```

```
Accuracy Score : 0.8485016648168702
```

MODEL EVALUATION

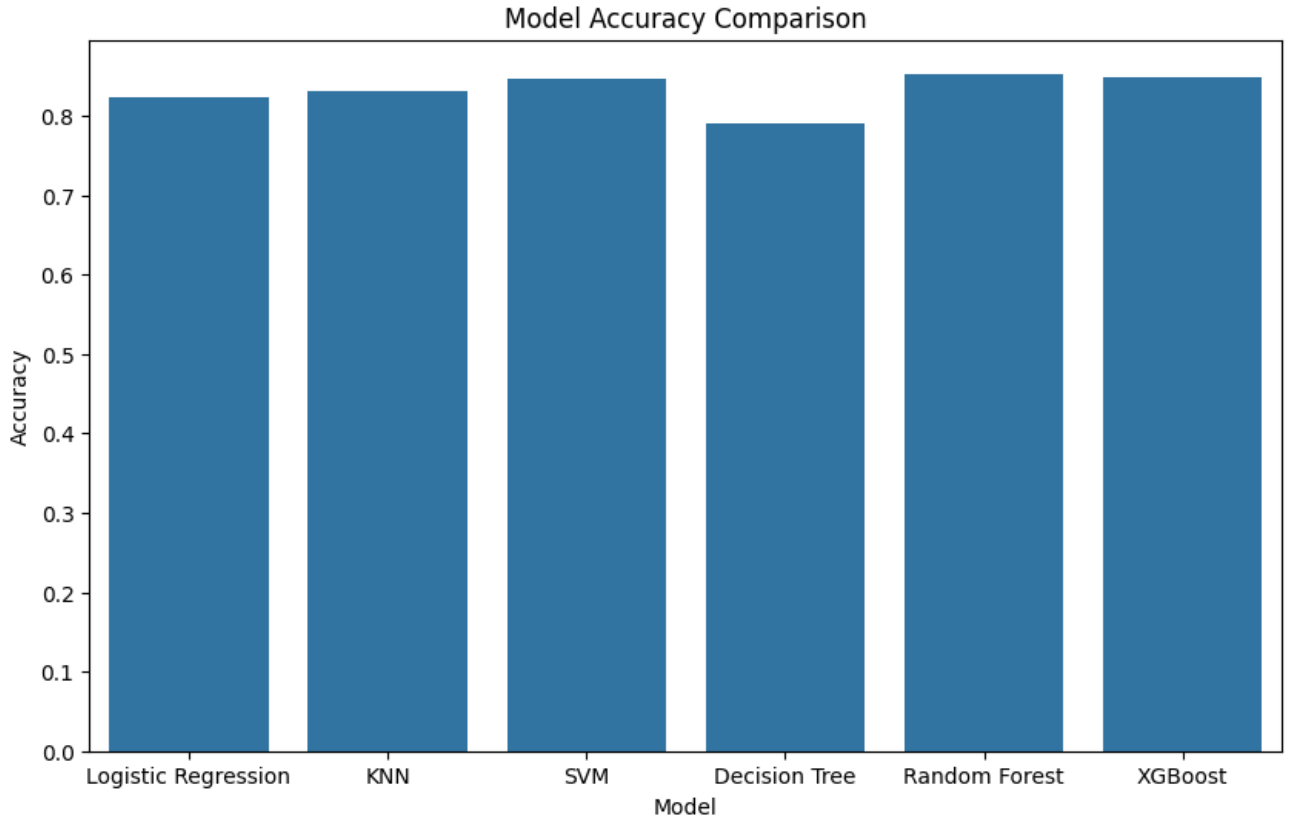
Model evaluation is the process of assessing how well a trained model performs on unseen data. It helps determine the model's accuracy, reliability, and generalization ability. Metrics such as accuracy, precision, recall, and F1-score are used to measure performance. These metrics ensure the model correctly identifies churners while minimizing misclassifications and errors.

```
results = {
    "Logistic Regression": accuracy_score(y_test, y_pred_lr),
    "KNN": accuracy_score(y_test, y_pred_knn),
    "SVM": accuracy_score(y_test, y_pred_svc),
    "Decision Tree": accuracy_score(y_test, y_pred_dt),
    "Random Forest": accuracy_score(y_test, y_pred_rf),
    "XGBoost":accuracy_score(y_test,y_pred_xgb)
}

result_df = pd.DataFrame(list(results.items()), columns=['Model', 'Accuracy'])
result_df
```

	Model	Accuracy
0	Logistic Regression	0.822974
1	KNN	0.831853
2	SVM	0.847392
3	Decision Tree	0.790233
4	Random Forest	0.851831
5	XGBoost	0.848502

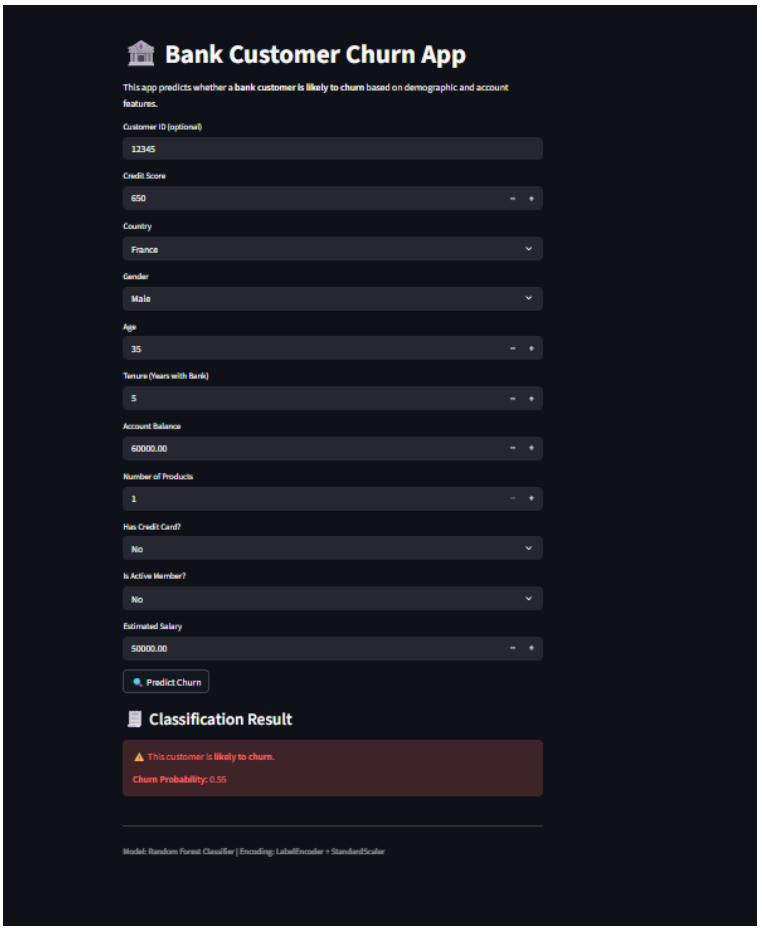
MODEL COMPARISION

After training multiple machine learning models, their performance was compared using evaluation metrics such as accuracy, precision, recall, and F1-score. Each model's results were analyzed to identify which one provided the best balance between accuracy and generalization. The model that achieved the highest overall performance and consistency was selected as the final model. This best-performing model was then saved as a .pkl file using the joblib library for future use and deployment without the need for retraining.



MODEL DEPLOYMENT

Model deployment is a crucial stage in any machine learning project, as it allows the trained model to be used in real-world applications. In this project, the Random Forest classifier is deployed using Streamlit, a lightweight and user-friendly Python framework designed for building interactive web applications. The deployment process begins with training the Random Forest model and saving it using libraries like joblib. Once the model is saved, a Streamlit application is created to provide a simple and intuitive interface for users. The application includes input widgets such as text boxes, sliders, and dropdowns to capture customer information or any relevant features required by the model. These inputs are then preprocessed to match the format used during training. When the user clicks the prediction button, the model loads the processed inputs and generates a prediction instantly. The output is displayed clearly, indicating whether the customer is likely to churn or not. Streamlit makes deployment seamless by eliminating the need for complex front-end development, enabling developers to create clean, interactive interfaces with minimal code. The final deployed model can be shared easily and used by stakeholders to explore scenarios and make informed decisions based on model predictions.



CONCLUSION

The analysis of the banking customer dataset provided valuable insights into factors influencing customer churn. Data cleaning, preprocessing, and EDA revealed patterns such as lower credit scores, younger age, fewer products, and inactive accounts being associated with higher churn rates. Machine learning models including Logistic Regression, SVM, KNN, Decision Tree, Random Forest, and XGBoost were trained and evaluated, with Random Forest emerging as the best-performing model due to its high accuracy and balanced precision-recall scores. The final model was saved for deployment, enabling the bank to predict potential churners effectively and take proactive measures to improve customer retention and engagement.

INSIGHTS

- Customers with lower Credit Score tend to have a higher likelihood of churn.
- Age influences churn: younger customers are more likely to leave than older customers.
- Customers with fewer Products or no active engagement are more likely to churn.
- Active Members are less likely to churn, showing higher loyalty.
- Balance and Estimated **Salary** affect churn: customers with higher balances and salaries are less likely to leave.
- Country distribution shows that some countries have higher churn rates than others.
- Gender has a minor impact, with slight differences in churn rates between male and female customers.