DESIGNING MACHINE LEARNING PREDICTION MODEL

Introduction to Artificial Intelligence

Kayode Ohundayo (U2476474)

Year: 2025

Introduction

This project explore the designing and building of a machine learning prediction model to train customer churn in banks sector.

Customer churn is defined by the capability of a customer to leave an essential service and here based on my project, it is the likelihood of the customer stops banking with his bank due to several reasons. Customer churn is important because it helps organisations like banks to take preventive measures for the retention customers

The project aims to analyse the customer churn dataset thoroughly, preprocess the data, classify and train the model and lastly evaluate the performance of the model.

Dataset Description

I imported a dataset from Kaggle the customer churn dataset, which contains all the information regarding bank customers and whether they exited the service.

In the project, the data set used will be the Customer Churn data set that holds data about customers of a bank and their churn status.

> The data set provides customer demographic data, account data, and activity data. The response variable is Exited, which equals one if the customer churned, otherwise zero. Number of rows (10,000)
> Our Target variable = **Exited**
> Our Type of task = **Classification**

We carried out the data preprocessing on the dataset to prepare it for modeling.

Categorical variables were handled using one-hot encoding. Stratified splitting was used to split the dataset into a training dataset and a testing dataset in a 70:30 ratio.

Section 1. Data Collection and preprocessing

The project is based on the Bank Customer Churn dataset, which includes 10,000 customers described by demographic, account, and activity information with a churn label Exited. I'm solving a binary classification problem: whether a bank customer will churn (Exited = 1) or stay (Exited = 0).

Main features include:

•CreditScore: customer's credit score in numeric format

•Geography: customer's country - France, Spain, Germany

• Gender: Male/Female

•Age and Tenure (years with bank),

•Balance, Num0fProducts, HasCrCard, IsActiveMember, EstimatedSalary.

•Identity-like fields: RowNumber, Customer Id, Surname

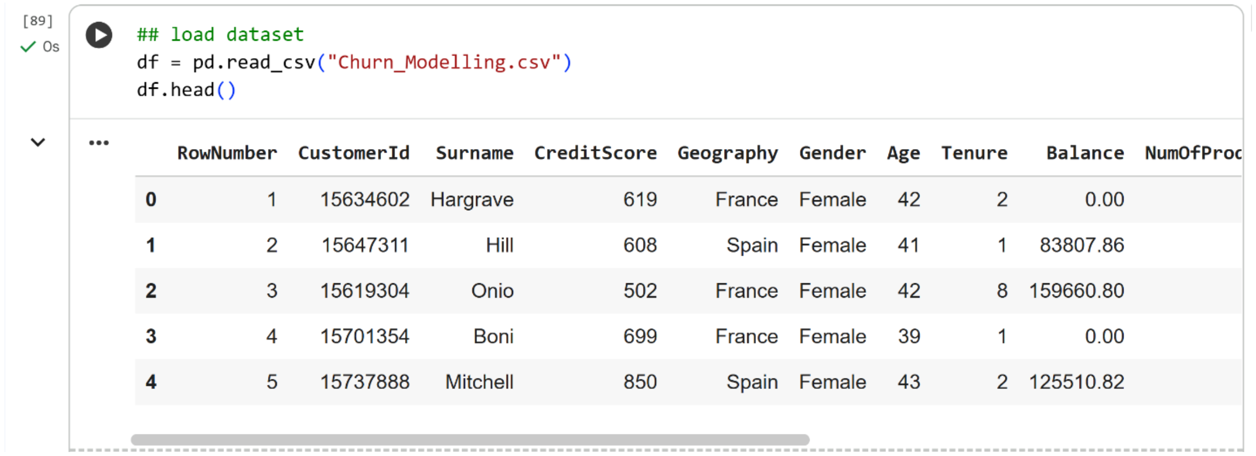
CODE

```
# ================================================
# importing the required libraries for the project
# ================================================

import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns

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

from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report

import warnings
warnings.filterwarnings('ignore')
```

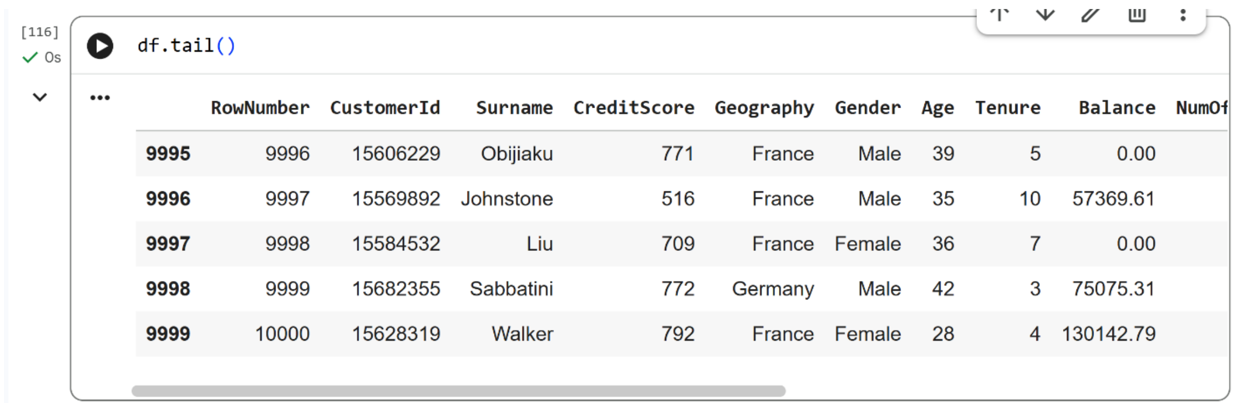
```
## load dataset
df = pd.read_csv("Churn_Modelling.csv")
df.head()
df.tail()
df.info()
df.describe()
```

This code loads the churn dataset and provides an overview of feature types, basic statistics, and the presence of the target variable.

Screenshot solutions // loading the churn modelling dataset and displaying values first 5 rows of the dataset


```
[89]
## load dataset
df = pd.read_csv("Churn_Modelling.csv")
df.head()
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProc
0	1	15634602	Hargrave	619	France	Female	42	2	0.00	
1	2	15647311	Hill	608	Spain	Female	41	1	83807.86	
2	3	15619304	Onio	502	France	Female	42	8	159660.80	
3	4	15701354	Boni	699	France	Female	39	1	0.00	
4	5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	


```
[116]
df.tail()
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOf
9995	9996	15606229	Obijiaku	771	France	Male	39	5	0.00	
9996	9997	15569892	Johnstone	516	France	Male	35	10	57369.61	
9997	9998	15584532	Liu	709	France	Female	36	7	0.00	
9998	9999	15682355	Sabbatini	772	Germany	Male	42	3	75075.31	
9999	10000	15628319	Walker	792	France	Female	28	4	130142.79	

Screenshot solutions // displaying the last 5 values rows of the dataset

1.2 Missing values and outliers

Data quality was evaluated for missing data, as well as reviewing numeric distributions to determine if there were any observations that could be considered outliers in CreditScore, Balance, or Estimated Salary features, and these outliers would either be accepted as viable or trimmed at extremes as necessary. Missing data was not a factor for any of the columns, so no data had to be imputed.

Code

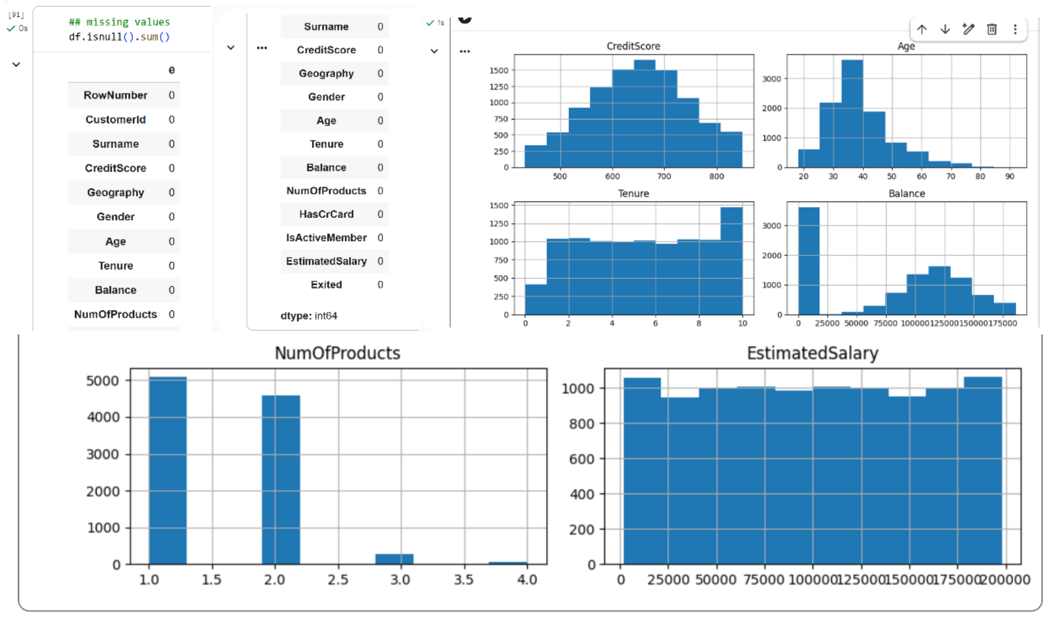
These preprocessing steps assist in ensuring that neither the model is biased by missing data, nor is it dominated by a small number of observations with a particular range of values.

```
## missing values
df.isnull().sum()

import seaborn as sns
import matplotlib.pyplot as plt

# Define list of numeric columns for visualization
```

```
numeric_cols = ['CreditScore', 'Age', 'Tenure', 'Balance',
                'NumOfProducts', 'EstimatedSalary']

# Create histograms for all numeric features to visualize their distributions
df[numeric_cols].hist(figsize=(10,8))
plt.tight_layout()
plt.show()

# Remove outliers by capping extreme values at 1% and 99% thresholds
for col in ['CreditScore', 'Balance', 'EstimatedSalary']:
    q1 = df[col].quantile(0.01)
    q99 = df[col].quantile(0.99)
    df[col] = df[col].clip(q1, q99)
```

These preprocessing steps help ensure the model is not biased by missing data or dominated by a small number of extreme values, which can distort learning.

Screenshot solutions



```
## missing values
df.isnull().sum()
```

```
                0
RowNumber       0
CustomerId      0
Surname         0
CreditScore     0
Geography       0
Gender          0
Age             0
Tenure          0
Balance         0
NumOfProducts   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
```

1.3 Encoding Categorically Variables

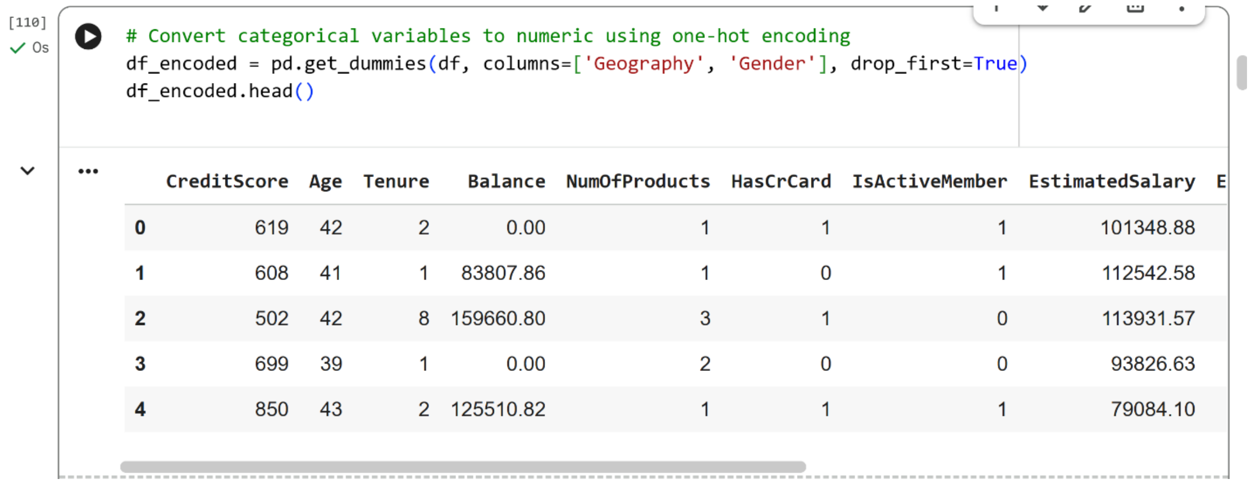
Machine learning models require numerical inputs, so categorical variables were encoded. **Geography** and **Gender** were converted using one-hot (dummy) encoding, with one category dropped to avoid redundancy.

Code

```
# Convert categorical variables to numeric using one-hot encoding
df_encoded = pd.get_dummies(df, columns=['Geography', 'Gender'], drop_first=True)
df_encoded.head()
```

This transformation produces binary features such as **Geography_Germany**, **Geography_Spain** and, **Gender_Male**, making the categorical information usable for both linear and tree-based models.

Screenshot solutions



```
[110]
0s
# Convert categorical variables to numeric using one-hot encoding
df_encoded = pd.get_dummies(df, columns=['Geography', 'Gender'], drop_first=True)
df_encoded.head()
```

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	E
0	619	42	2	0.00	1	1	1	101348.88	
1	608	41	1	83807.86	1	0	1	112542.58	
2	502	42	8	159660.80	3	1	0	113931.57	
3	699	39	1	0.00	2	0	0	93826.63	
4	850	43	2	125510.82	1	1	1	79084.10	

Scaling

```
# Split data into features (X) and target variable (y)
X = df_encoded.drop('Exited', axis=1)
y = df_encoded['Exited']

from sklearn.preprocessing import StandardScaler

# Scale numeric features to a standard range
scaler = StandardScaler()
num_cols = ["CreditScore", "Age", "Tenure", "Balance", "EstimatedSalary"]

df_encoded[num_cols] = scaler.fit_transform(df_encoded[num_cols])
```

The dataset was split into training and test sets with a 70/30 ratio, using stratification on **Exited** to preserve the churn proportion. Numeric features were scaled using **StandardScaler** for models sensitive to feature magnitude (e.g. Logistic Regression), while the Random Forest uses the unscaled version.

Spliting the data into training and testing sets

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

# Split data into training (70%) and testing (30%) sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)

# Scale features to standardize the data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Splitting the data makes it possible to conduct unbiased testing on new data, while scaling helps in improving the speed of convergence of algorithms based on distances and gradients.

Section 2 – Model selection and Training

Choice of algorithm and tasks

Because the target **Exited** is binary, both models address a **classification** problem. We chose two models for training:

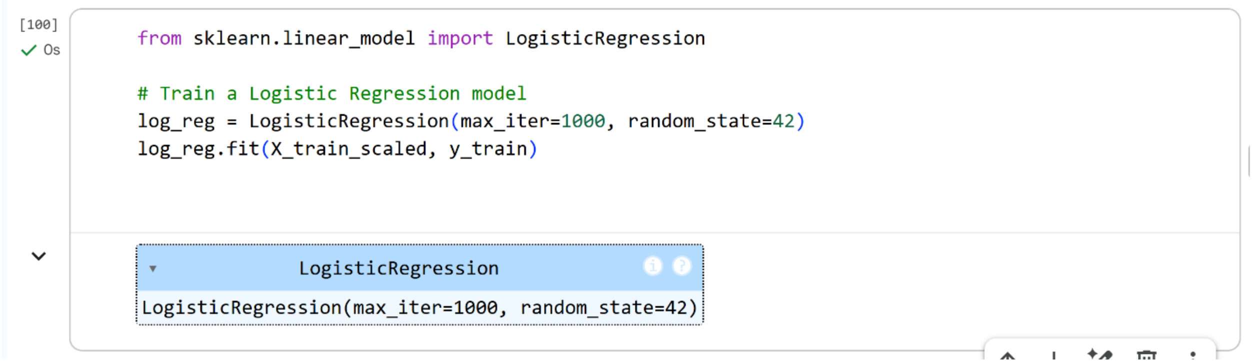
•Logistic Regression Model: A straightforward linear classifier that yields probabilities and is a good starting point.

•Random Forest Classifier Model: This is a combination of decision trees, which has the ability to identify non-linear relationships. It is common in handling data with columns.

This enables an analysis of a linear versus a non-linear approach on the same problem of churning. **Logistic Regression Model**

```
from sklearn.linear_model import LogisticRegression

# Train a Logistic Regression model
log_reg = LogisticRegression(max_iter=1000, random_state=42)
log_reg.fit(X_train_scaled, y_train)
```

Screenshot solutions



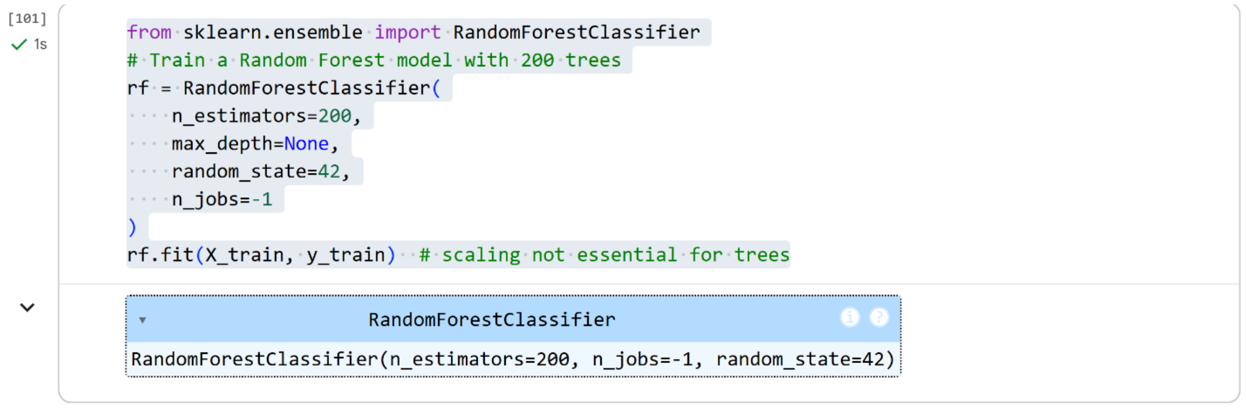
```
[100]
✓ 0s
from sklearn.linear_model import LogisticRegression

# Train a Logistic Regression model
log_reg = LogisticRegression(max_iter=1000, random_state=42)
log_reg.fit(X_train_scaled, y_train)
```

```
LogisticRegression
LogisticRegression(max_iter=1000, random_state=42)
```

Random Forest Model

```
from sklearn.ensemble import RandomForestClassifier
# Train a Random Forest model with 200 trees
rf = RandomForestClassifier(
    n_estimators=200,
    max_depth=None,
    random_state=42,
    n_jobs=-1
)
rf.fit(X_train, y_train)  # scaling not essential for trees
```

Screenshot solutions

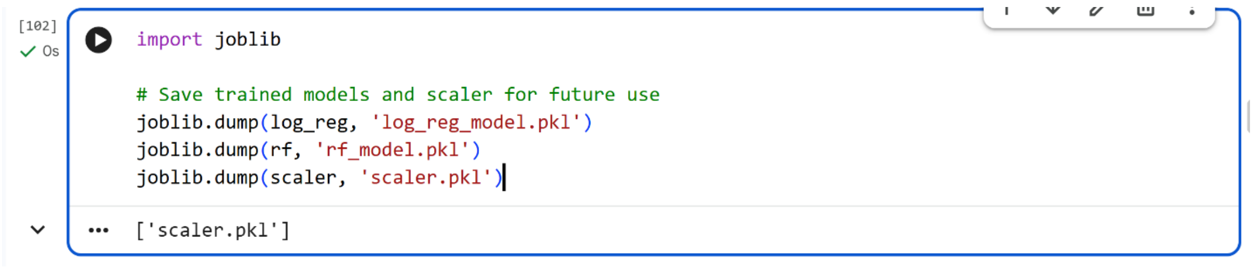


```
[101]
✓ 1s
from sklearn.ensemble import RandomForestClassifier
# Train a Random Forest model with 200 trees
rf = RandomForestClassifier(
    n_estimators=200,
    max_depth=None,
    random_state=42,
    n_jobs=-1
)
rf.fit(X_train, y_train)  # scaling not essential for trees
```

```
RandomForestClassifier
RandomForestClassifier(n_estimators=200, n_jobs=-1, random_state=42)
```

Saving trained models

```
import joblib

# Save trained models and scaler for future use
joblib.dump(log_reg, 'log_reg_model.pkl')
joblib.dump(rf, 'rf_model.pkl')
joblib.dump(scaler, 'scaler.pkl')
```



```
[102]
0s
import joblib

# Save trained models and scaler for future use
joblib.dump(log_reg, 'log_reg_model.pkl')
joblib.dump(rf, 'rf_model.pkl')
joblib.dump(scaler, 'scaler.pkl')
```

```
['scaler.pkl']
```

Section 3 – Prediction and Evaluation

3.1 Generating predictions for the test set

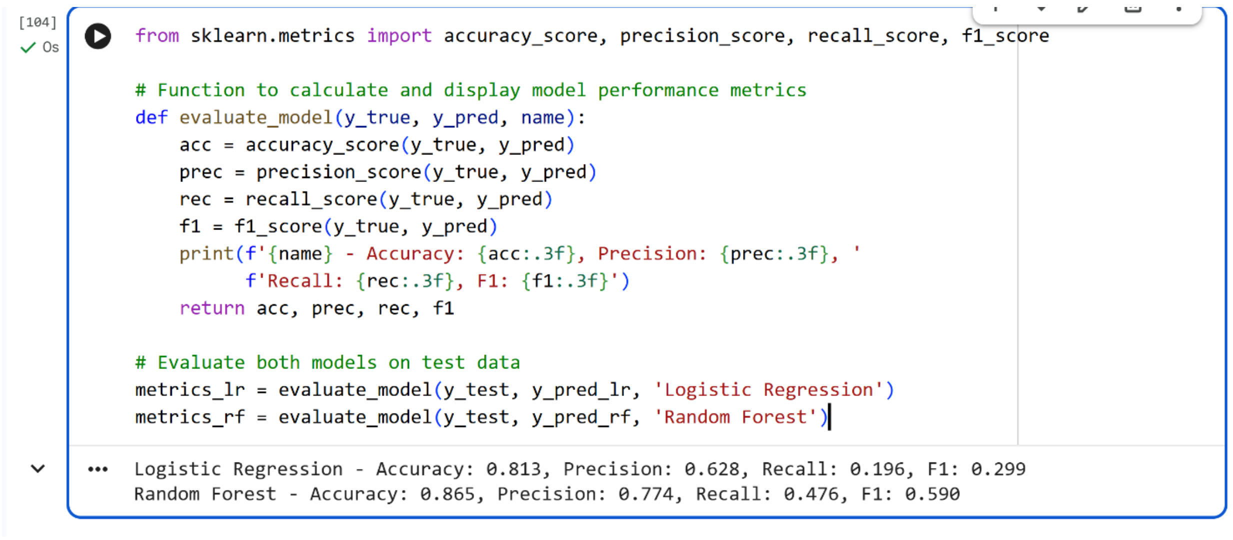
Predictions and probabilities were made for both models on the held-out test set.

```
# Make predictions using Logistic Regression model
y_pred_lr = log_reg.predict(X_test_scaled)
y_prob_lr = log_reg.predict_proba(X_test_scaled)[:, 1]

# Make predictions using Random Forest model
y_pred_rf = rf.predict(X_test)
y_prob_rf = rf.predict_proba(X_test)[:, 1]
```

Class labels consist of predicted churn or non churn while probability scores are used for evaluation with threshold measures like ROC and AUC.

3.2 Evaluation Metrics

```
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score

# Function to calculate and display model performance metrics
def evaluate_model(y_true, y_pred, name):
    acc = accuracy_score(y_true, y_pred)
    prec = precision_score(y_true, y_pred)
    rec = recall_score(y_true, y_pred)
    f1 = f1_score(y_true, y_pred)
    print(f'{name} - Accuracy: {acc:.3f}, Precision: {prec:.3f}, '
          f'Recall: {rec:.3f}, F1: {f1:.3f}')
    return acc, prec, rec, f1

# Evaluate both models on test data
metrics_lr = evaluate_model(y_test, y_pred_lr, 'Logistic Regression')
metrics_rf = evaluate_model(y_test, y_pred_rf, 'Random Forest')
```



```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Function to calculate and display model performance metrics
def evaluate_model(y_true, y_pred, name):
    acc = accuracy_score(y_true, y_pred)
    prec = precision_score(y_true, y_pred)
    rec = recall_score(y_true, y_pred)
    f1 = f1_score(y_true, y_pred)
    print(f'{name} - Accuracy: {acc:.3f}, Precision: {prec:.3f}, '
          f'Recall: {rec:.3f}, F1: {f1:.3f}')
    return acc, prec, rec, f1

# Evaluate both models on test data
metrics_lr = evaluate_model(y_test, y_pred_lr, 'Logistic Regression')
metrics_rf = evaluate_model(y_test, y_pred_rf, 'Random Forest')
```

```
Logistic Regression - Accuracy: 0.813, Precision: 0.628, Recall: 0.196, F1: 0.299
Random Forest - Accuracy: 0.865, Precision: 0.774, Recall: 0.476, F1: 0.590
```

In these metrics accuracy precision recall and f1 measure performance on classification models

These evaluate various aspects of performance: accuracy indicates overall prediction fairness, precision shows reliability of predictions ad recall shows the ability to correctly identify churning customers and f1 is a balance between precision and recall

Model	**Accuracy**	**Precision**	**Recall**	**F1**
Logistic Regression	**0.813**	**0.628**	**0.196**	**0.299**
Random Forest	**0.865**	**0.774**	**0.476**	**0.590**

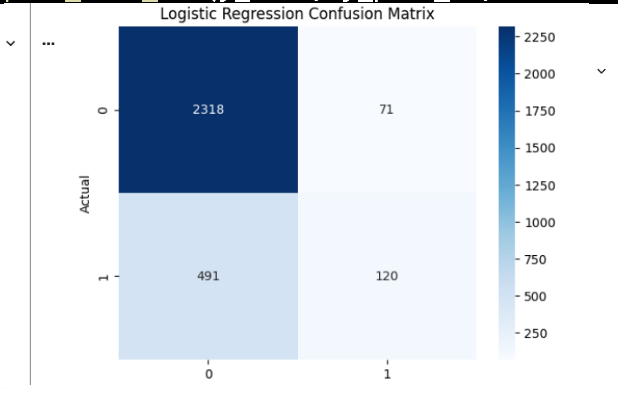
3.3 Comparison and Discussion

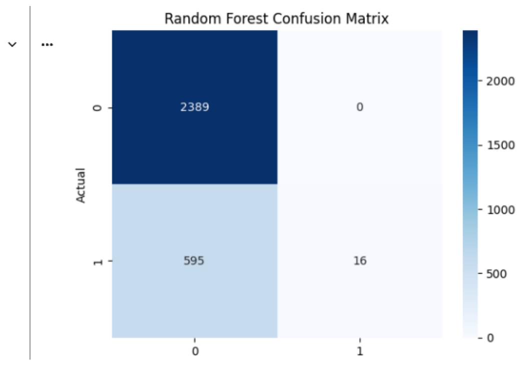
For this project, the Random Forest model was overall better at recall and F1 measure than the Logistics Regression model. This means that the Random Forest model is preferable when applied to the churn task. The reason for this preference is that the cost of not contacting a possible churner (false negative) might be higher than contacting someone who is not going to churn (false positive).

4.1 Confusion matrices

Confusion Matrices were drawn up to illustrate the correct and incorrect results for each model.

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Function to create and display confusion matrix
def plot_conf_mat(y_true, y_pred, title):
    cm = confusion_matrix(y_true, y_pred)
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title(title)
    plt.show()

# Plot confusion matrices for both models
plot_conf_mat(y_test, y_pred_lr, 'Logistic Regression Confusion Matrix')
plot_conf_mat(y_test, y_pred_rf, 'Random Forest Confusion Matrix')
```





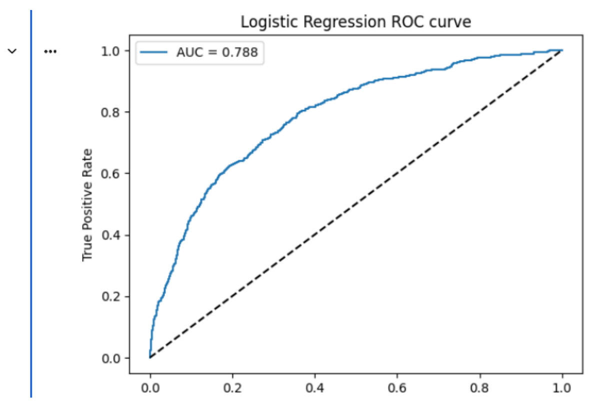
The plots below emphasize true negatives, false positives, false negatives, and true positives to show more clearly which model yields fewer missed churners:

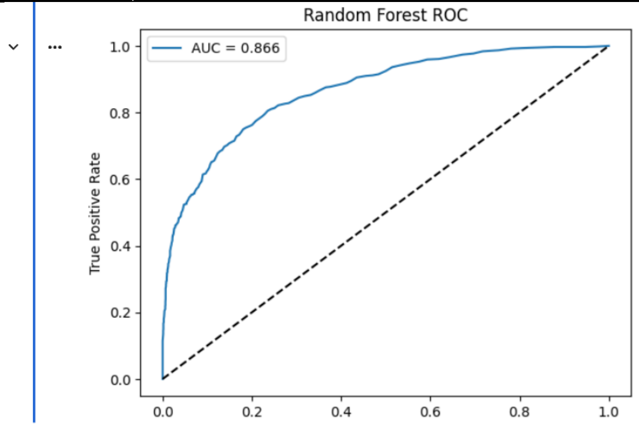
4.2 ROC curves and AUC

The performance evaluation was done across all classification thresholds with the use of ROC curves. AUC summarized the ranking quality.

```
from sklearn.metrics import roc_curve, roc_auc_score

# Function to plot ROC curve and calculate AUC score
def plot_roc(y_true, y_prob, title):
    fpr, tpr, _ = roc_curve(y_true, y_prob)
    auc = roc_auc_score(y_true, y_prob)
    plt.plot(fpr, tpr, label=f'AUC = {auc:.3f}')
    plt.plot([0,1], [0,1], 'k--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(title)
    plt.legend()
    plt.show()

# Plot ROC curves for both models
```

```
plot_roc(y_test, y_prob_lr, 'Logistic Regression ROC curve')
plot_roc(y_test, y_prob_rf, 'Random Forest ROC')
```



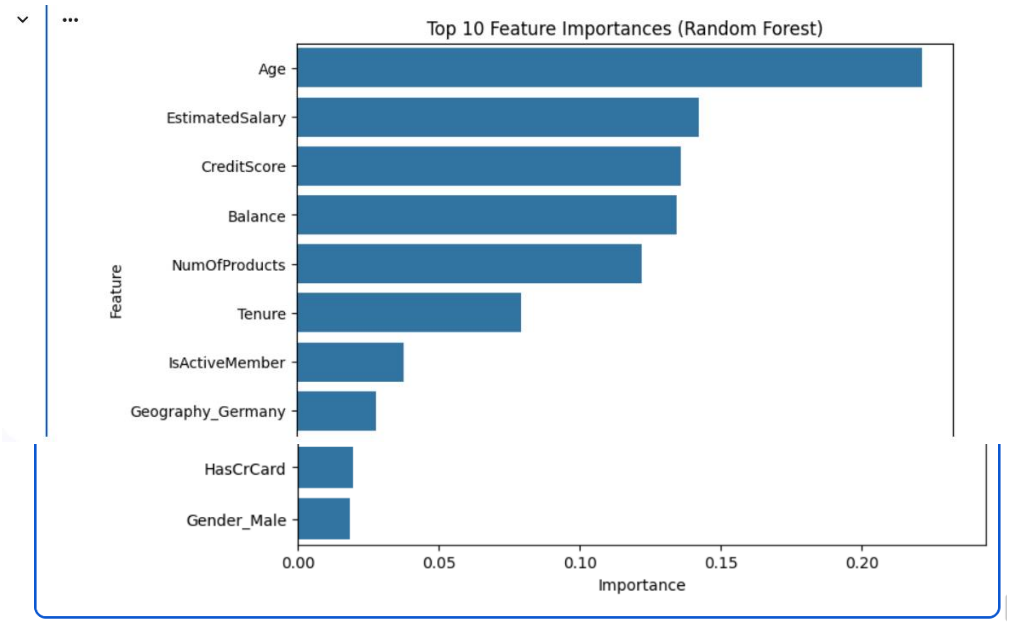


The higher AUC for Random Forest makes it better at ranking churners than Non-Churners, and this makes a strong case for selecting Random Forest as the best model compared to linear regression. The accuracy percentage of random forest is 86% whilst the one for linear regression shows 78%.

4.3 Feature importance

Feature importance obtained from the Random Forest model was also evaluated to identify the most relevant variables or features on which the churn prediction model output relied.

```
import numpy as np

# Extract feature importances from Random Forest model
importances = rf.feature_importances_
feature_names = X.columns

feat_imp = pd.Series(importances,
index=feature_names).sort_values(ascending=False)

# Plot top 10 most important feature
plt.figure(figsize=(8,6))
sns.barplot(x=feat_imp[:10], y=feat_imp.index[:10])
plt.title('Top 10 Feature Importances (Random Forest)')
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.show()
```



Main important features from random forest include:

Age, Balance, NumofProducts, Geography and lastly IsActiveMembers

This suggest that older customers with certain balance level and lower activity statuses are more likely to churn. This insights can help us to target retention strategies, such as focusing on campaigns on high-risk segments that is identified by the model