RICHO

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from google.colab import drive
drive.mount('/content/drive')
df = pd.read_csv('/content/drive/My Drive/Colab 
Notebooks/progresproject/posting/Customer_Chrun_Prediction.csv')
```

```
Drive already mounted at /content/drive; to attempt to forcibly 
remount, call drive.mount("/content/drive", force_remount=True).
```

```
df.head()
```

```
   RowNumber  CustomerId   Surname  CreditScore Geography  Gender  Age  
\
0          1    15634602  Hargrave          619    France  Female   42   

1          2    15647311      Hill          608     Spain  Female   41   

2          3    15619304      Onio          502    France  Female   42   

3          4    15701354      Boni          699    France  Female   39   

4          5    15737888  Mitchell          850     Spain  Female   43   


   Tenure    Balance  NumOfProducts  HasCrCard  IsActiveMember  \
0       2       0.00              1          1               1   
1       1   83807.86              1          0               1   
2       8  159660.80              3          1               0   
3       1       0.00              2          0               0   
4       2  125510.82              1          1               1   

   EstimatedSalary  Exited  
0        101348.88       1  
1        112542.58       0  
2        113931.57       1  
3         93826.63       0  
4         79084.10       0  
```

```
df.tail()
```

```
      RowNumber  CustomerId    Surname  CreditScore Geography  Gender 
Age  \
9995       9996    15606229   Obijiaku          771    France    Male   
```

```
39
9996        9997    15569892  Johnstone          516    France    Male
35
9997        9998    15584532        Liu          709    France  Female
36
9998        9999    15682355  Sabbatini          772   Germany    Male
42
9999       10000    15628319     Walker          792    France  Female
28

      Tenure    Balance  NumOfProducts  HasCrCard  IsActiveMember  \
9995       5       0.00              2          1               0
9996      10   57369.61              1          1               1
9997       7       0.00              1          0               1
9998       3   75075.31              2          1               0
9999       4  130142.79              1          1               0

      EstimatedSalary  Exited
9995         96270.64       0
9996        101699.77       0
9997         42085.58       1
9998         92888.52       1
9999         38190.78       0
```

```
df.shape
```

```
(10000, 14)
```

```
df = df.drop(['RowNumber','CustomerId','Surname'], axis=1)
```

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

```
CreditScore        0
Geography          0
Gender             0
Age                0
Tenure             0
Balance            0
NumOfProducts      0
HasCrCard          0
IsActiveMember     0
EstimatedSalary    0
Exited             0
dtype: int64
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype
```

```
---  ------            --------------  -----
 0   CreditScore      10000 non-null  int64
 1   Geography        10000 non-null  object
 2   Gender           10000 non-null  object
 3   Age              10000 non-null  int64
 4   Tenure           10000 non-null  int64
 5   Balance          10000 non-null  float64
 6   NumOfProducts    10000 non-null  int64
 7   HasCrCard        10000 non-null  int64
 8   IsActiveMember   10000 non-null  int64
 9   EstimatedSalary  10000 non-null  float64
 10  Exited           10000 non-null  int64
dtypes: float64(2), int64(7), object(2)
memory usage: 859.5+ KB
```

```
df.duplicated().sum()
```

```
0
```

```
df.describe()
```

```
        CreditScore           Age       Tenure        Balance
NumOfProducts  \
count  10000.000000  10000.000000  10000.000000   10000.000000
10000.000000
mean     650.528800     38.921800      5.012800   76485.889288
1.530200
std       96.653299     10.487806      2.892174   62397.405202
0.581654
min      350.000000     18.000000      0.000000       0.000000
1.000000
25%      584.000000     32.000000      3.000000       0.000000
1.000000
50%      652.000000     37.000000      5.000000   97198.540000
1.000000
75%      718.000000     44.000000      7.000000  127644.240000
2.000000
max      850.000000     92.000000     10.000000  250898.090000
4.000000

         HasCrCard  IsActiveMember  EstimatedSalary         Churn
count  10000.00000    10000.000000     10000.000000  10000.000000
mean       0.70550        0.515100    100090.239881      0.203700
std        0.45584        0.499797     57510.492818      0.402769
min        0.00000        0.000000        11.580000      0.000000
25%        0.00000        0.000000     51002.110000      0.000000
50%        1.00000        1.000000    100193.915000      0.000000
75%        1.00000        1.000000    149388.247500      0.000000
max        1.00000        1.000000    199992.480000      1.000000
```

```
df.rename(columns={'Exited':'Churn'}, inplace=True)
```

Explorative Data Analysis

Pie Chart for Customer Churn

```
plt.figure(figsize=(10,6))
plt.pie(df['Churn'].value_counts(),labels=['No','Yes'],autopct='%1.2f%
%')
plt.title('Churn Percentage')
plt.show()
```

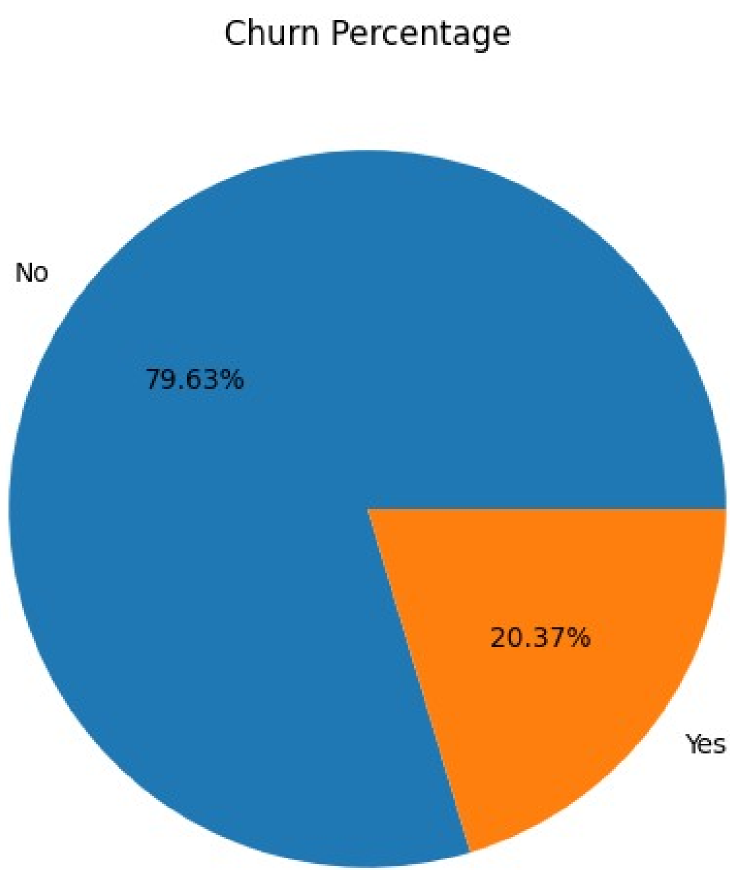
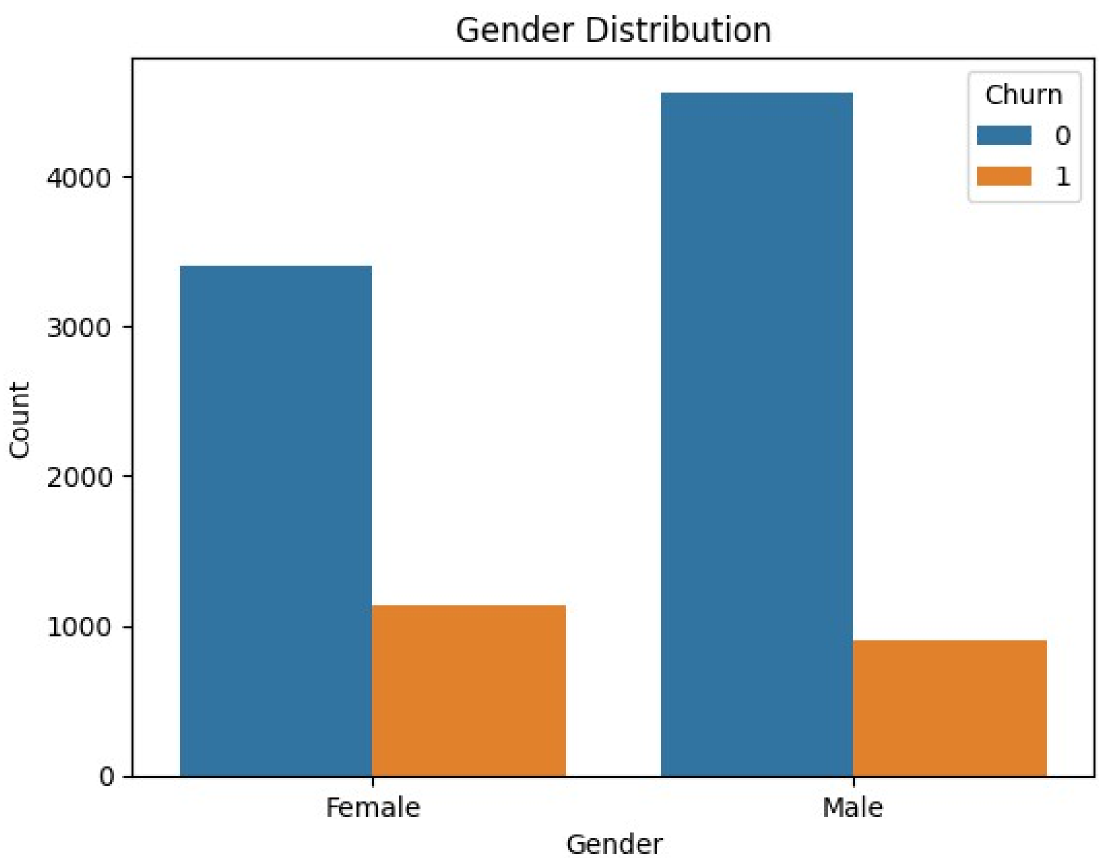


Diagram lingkaran dengan jelas memvisualisasikan churn pelanggan dalam kumpulan data. Mayoritas nasabah dalam kumpulan data terus menggunakan layanan bank (blue color), namun terdapat juga pemberhentian layanan dengan hanya 20,37% (orange color).

Gender

```
sns.countplot(x = 'Gender', data = df, hue = 'Churn')
plt.title('Gender Distribution')
plt.xlabel('Gender')
plt.ylabel('Count')
plt.show()
```

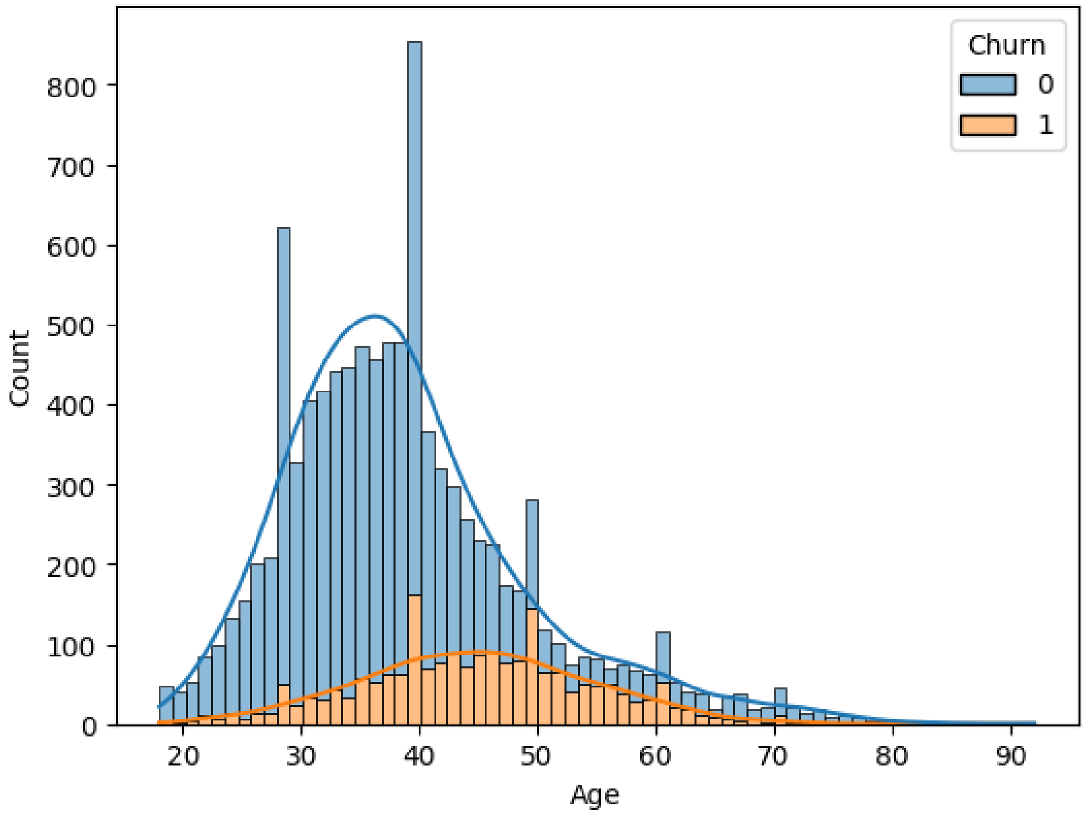


As seen in the graph, the majority of customers are men. However, if we look at customer churn, we can see that women have a greater tendency to churn than men. However, there is not much difference between the churn numbers of the two genders so we cannot make a hypothesis regarding customer churn based on customer gender.

Age Distribution

```
sns.histplot(data=df, x="Age", hue="Churn", multiple="stack",kde=True)
```

```
<Axes: xlabel='Age', ylabel='Count'>
```

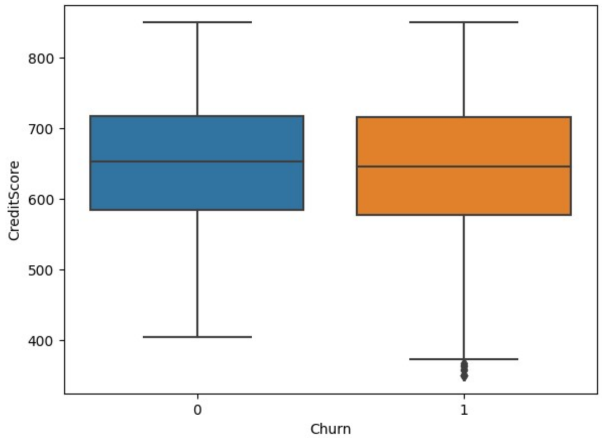


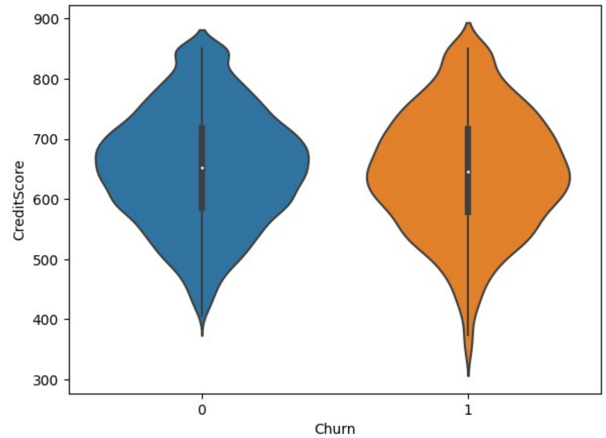
This histtogram visualizes the age distribution and the churn count of the customers. The majority of customers are from the age group 30-40 years old. However the customer churn count is highest for the customers of age 40 and 50. Therefore, age plays a significant role in customer churn, where late adults are more likely to churn as compared to young adults with minimal churn count.

Credit Score

```
fig, ax = plt.subplots(1,2,figsize=(15, 5))
sns.boxplot(x="Churn", y="CreditScore", data=df, ax=ax[0])
sns.violinplot(x="Churn", y="CreditScore", data=df, ax=ax[1])
```

```
<Axes: xlabel='Churn', ylabel='CreditScore'>
```

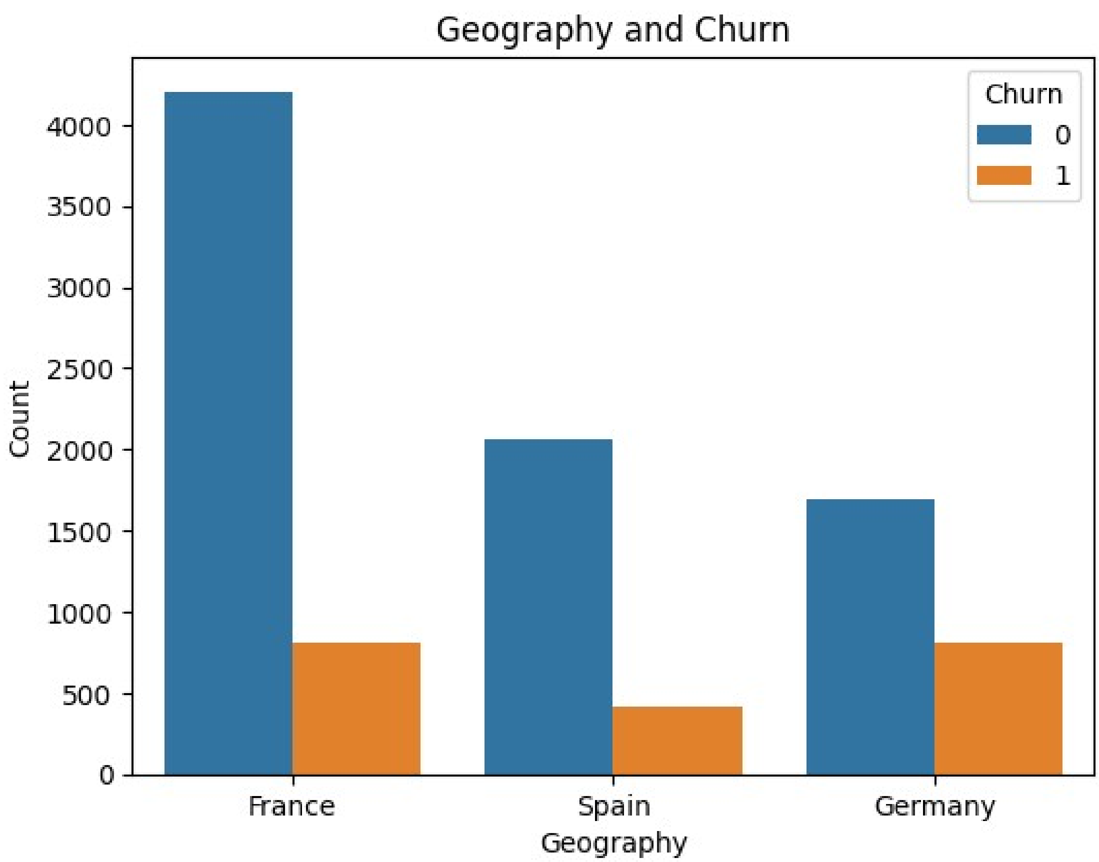




In the boxplot, the median of both the churn and non churn customers are almost same. In addition to that, the shape of violinplot is also similar for both the churn and non churn customers. However some churn customers have low credit score, but on the whole, the credit score is not a good indicator of churn.

Customer Location

```
sns.countplot(x = 'Geography', hue = 'Churn', data = df)
plt.title('Geography and Churn')
plt.xlabel('Geography')
plt.ylabel('Count')
plt.show()
```

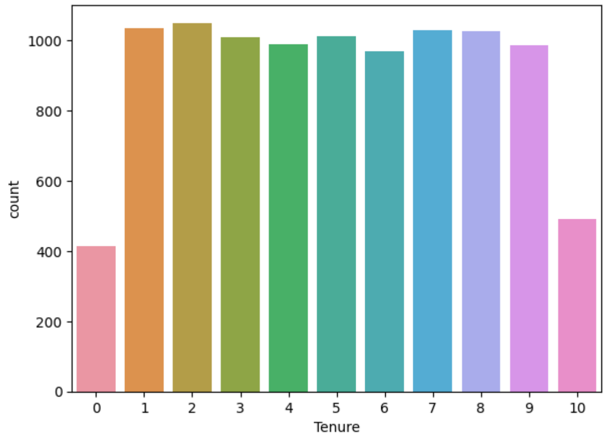


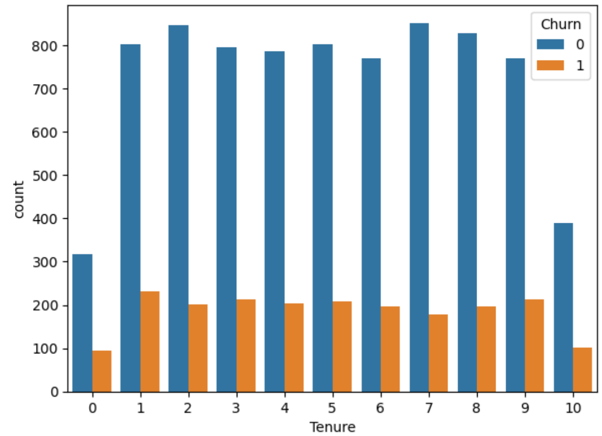
Majority of the customers are from France, followed by Spain and Germany. However in contrast to that France has the highest number of customer curn followed by Germany and Spain. From this we can infer that France customers are more likely to churn than the customers from other countries.

Tenure

```
fig,ax = plt.subplots(1,2,figsize=(15,5))
sns.countplot(x='Tenure', data=df,ax=ax[0])
sns.countplot(x='Tenure', hue='Churn', data=df,ax=ax[1])
```

```
<Axes: xlabel='Tenure', ylabel='count'>
```



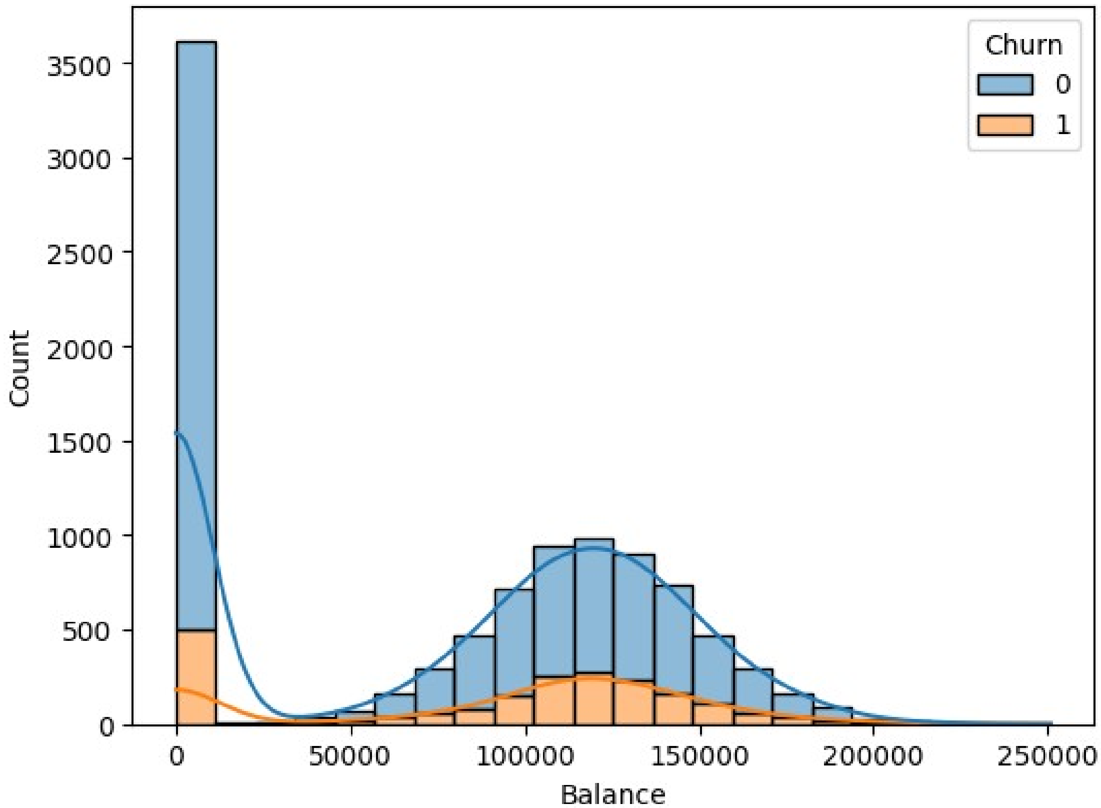


Looking at the churn of these customers based on their tenure, it can be observed that customers with tenure 1-9 years have higher churn count with maximum in customers with 1 year tenure followed those with 9 year tenure. However customers more than 9 years on tenure counts for the least churn. This is because the customers with higher tenure are more loyal to the bank and less likely to churn.

Bank Balance

```
sns.histplot(data=df, x="Balance", hue="Churn",
multiple="stack",kde=True)
```

```
<Axes: xlabel='Balance', ylabel='Count'>
```

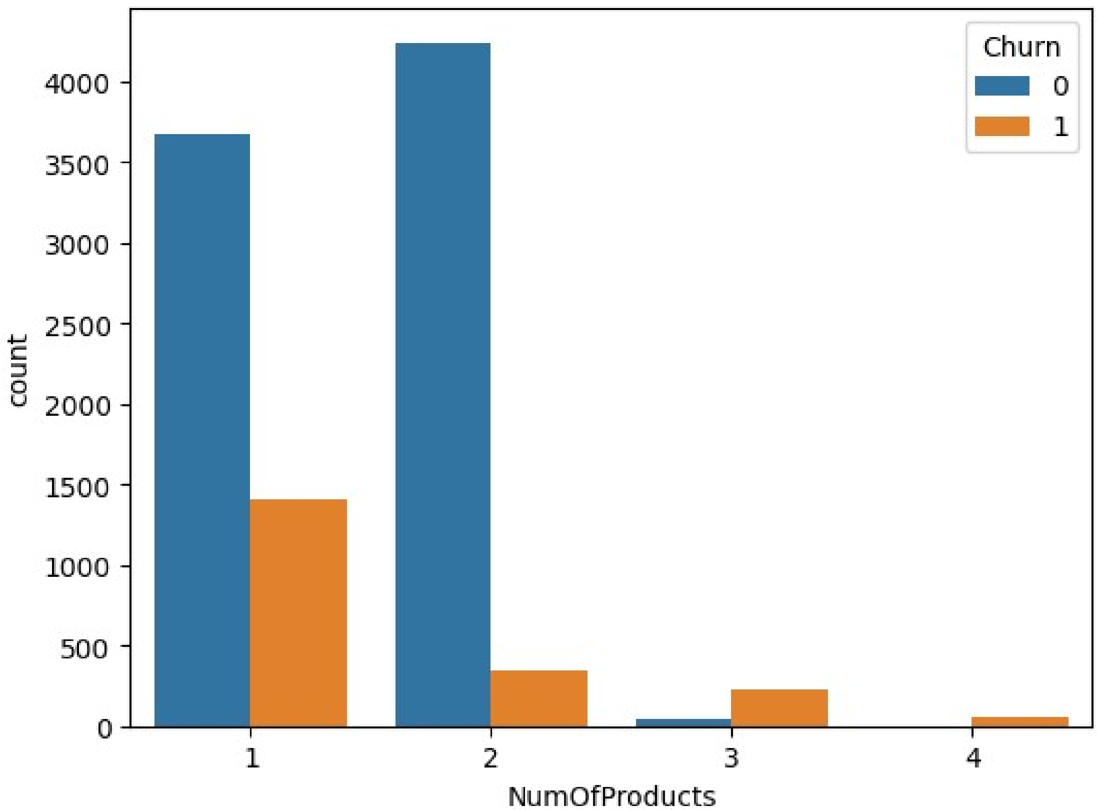


A huge number of customers have zero bank balance which also resulted in them leaving the bank. However, customer having bank balance between 100000 to 150000 are more likely to leave the bank after the customers with zero bank balance.

Number of Products Purchased

```
sns.countplot(x='NumOfProducts', hue='Churn', data=df)
```

```
<Axes: xlabel='NumOfProducts', ylabel='count'>
```

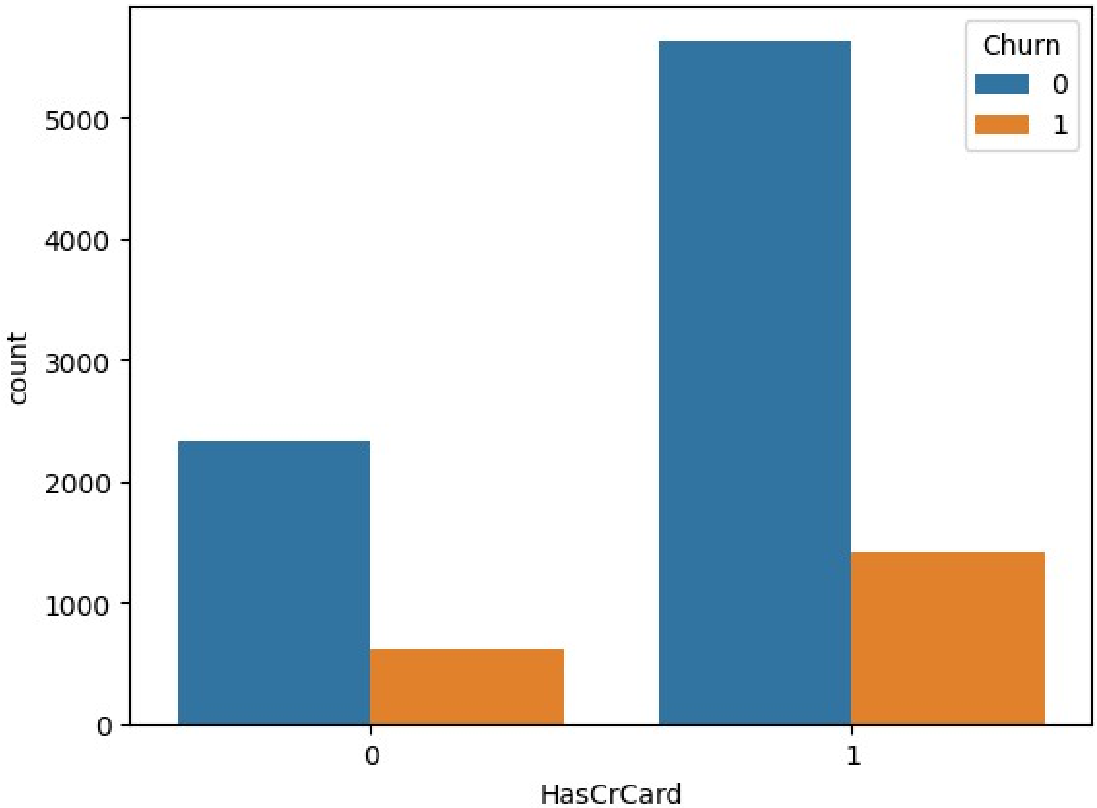


In the dataset, we have customers in four categories according to the number of products purchased. The customers with purchase or 1 or 2 products are highest in number and have low churn count in comparison to the non churn customers in the category. However, in the category where customers have purchased 3 or 4 products the number of leaving customers is much higher than the non leaving customers. Therefore, the number of product purchased is a good indicator of customer churn.

Customers with/without credit card

```
sns.countplot(x=df['HasCrCard'],hue=df['Churn'])
```

```
<Axes: xlabel='HasCrCard', ylabel='count'>
```

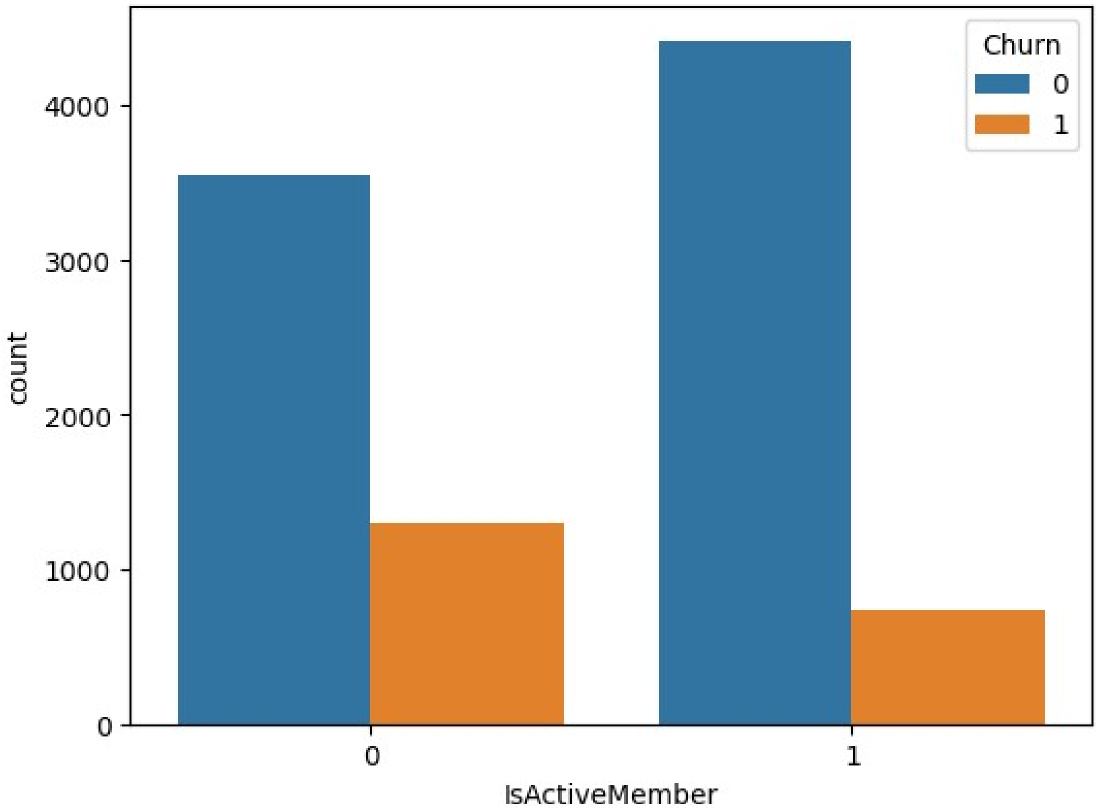


Majoity of the customers have credit cars i.e. nealy 70% of the customers have credit cards leaving 30% of the customers who do not have credit cards. Moreover, the number of customers leaving the bank are more whom have a credit card.

Active Members

```
sns.countplot(x='IsActiveMember', hue='Churn', data=df)
```

```
<Axes: xlabel='IsActiveMember', ylabel='count'>
```

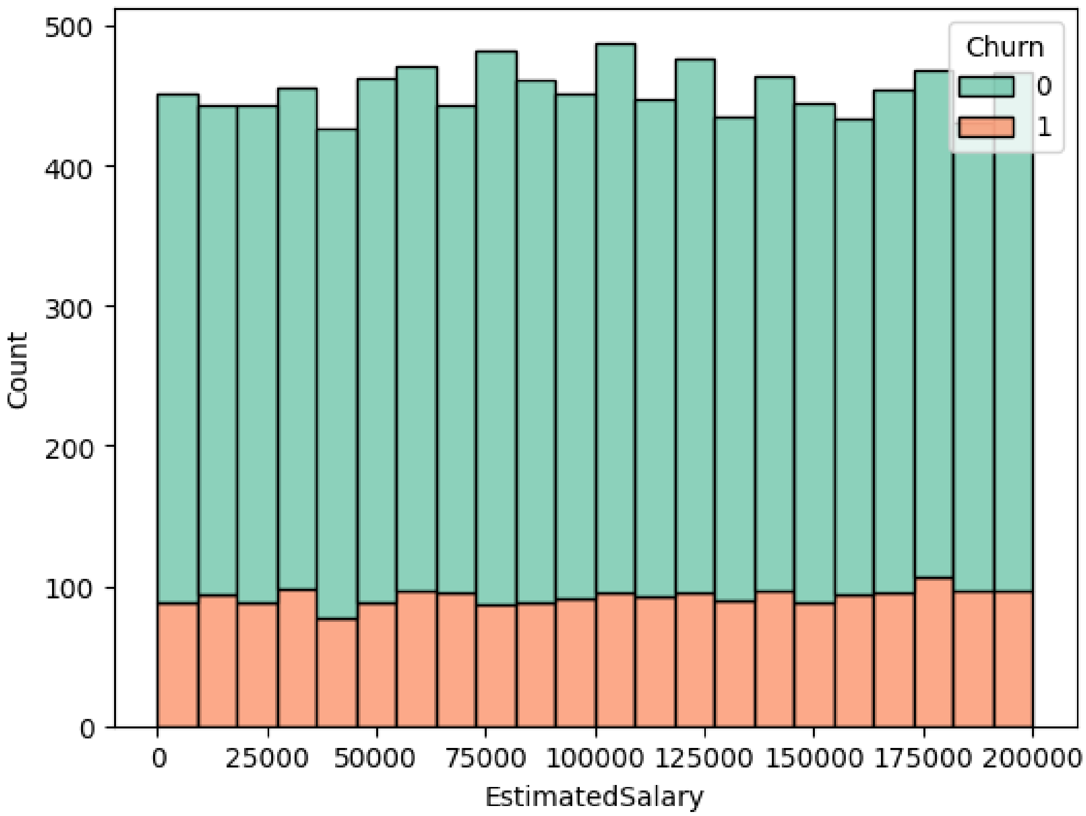


As expected, the churn count is higher for non active members as compared to the active members of the bank. This is because the active members are more satisfied with the services of the bank and hence they are less likely to leave the bank. Therefore, the bank should focus on the non active members and try to improve their services to retain them.

Estimated Salary

```
sns.histplot(data=df,x='EstimatedSalary',hue='Churn',multiple='stack',
palette='Set2')
```

```
<Axes: xlabel='EstimatedSalary', ylabel='Count'>
```



This graph shows the distribution of the estimated salary of the customers along with the churn count. On the whole the there is no definite pattern in the salary distribution of the customers who churned and who didn't. Therefore estimated salary is not a good predictor of churn.

DATA PREPROCESSING 2

Label Encoding the variables

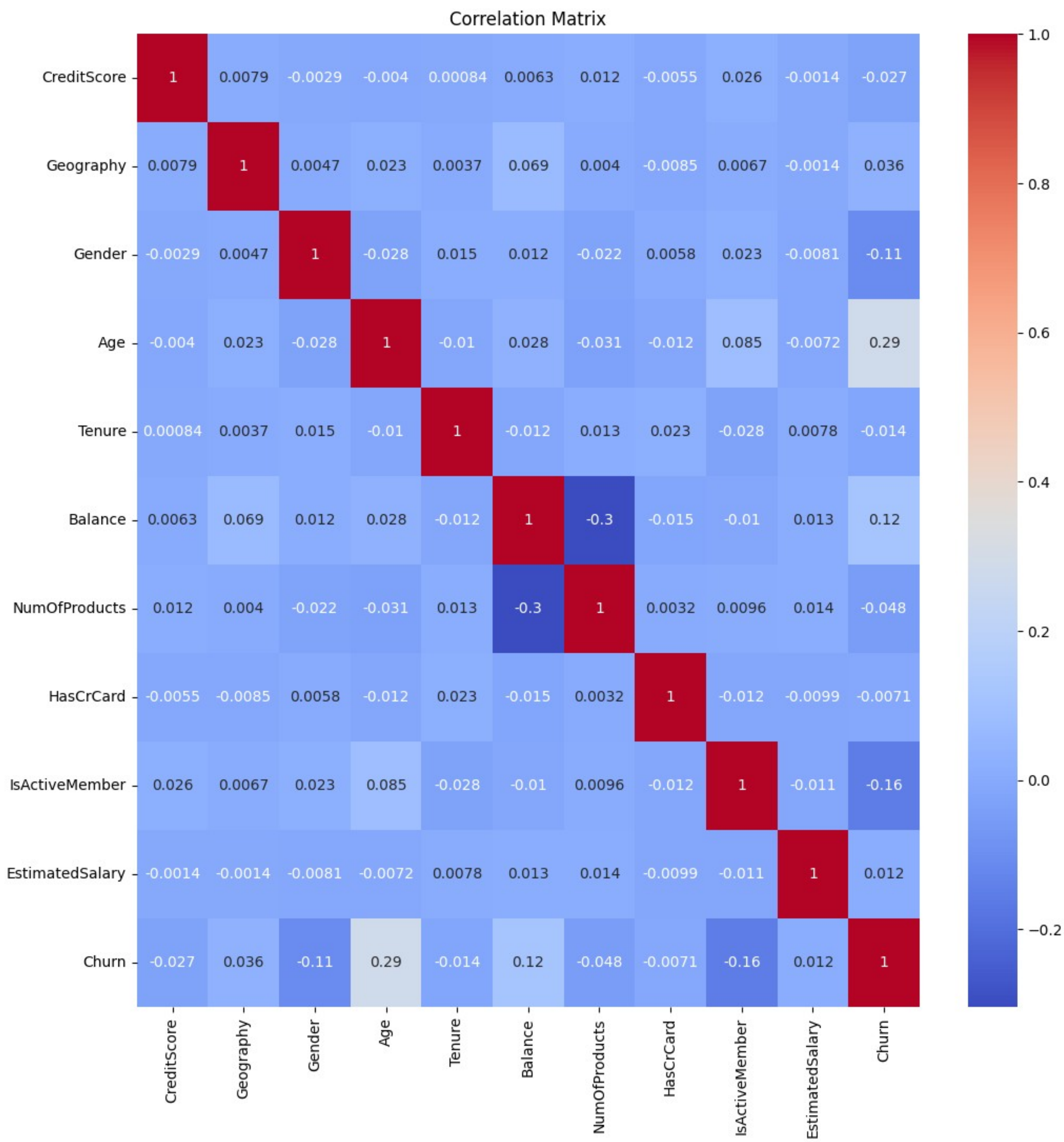
```
variables = ['Geography','Gender']
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
for i in variables:
    le.fit(df[i].unique())
    df[i]=le.transform(df[i])
    print(i,df[i].unique())

Geography [0 2 1]
Gender [0 1]
```

Normalization

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
df[['CreditScore','Balance','EstimatedSalary']] =
scaler.fit_transform(df[['CreditScore','Balance','EstimatedSalary']])
```

Coorelation Matrix Heatmap

```
plt.figure(figsize=(12,12))
sns.heatmap(df.corr(),annot=True,cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```



Train Test Split

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(df.drop('Churn',axis=1)
,df['Churn'],test_size=0.3,random_state=42)
```

Churn Prediction

Decision Tree

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
#creating Decision Tree Classifer object
dtree = DecisionTreeClassifier()

#defining parameter range
param_grid = {
    'max_depth': [2,4,6,8,10,12,14,16,18,20],
    'min_samples_leaf': [1,2,3,4,5,6,7,8,9,10],
    'criterion': ['gini', 'entropy'],
    'random_state': [0,42]
    }

#Creating grid search object
grid_dtree = GridSearchCV(dtree, param_grid, cv = 5, scoring =
'roc_auc', n_jobs = -1, verbose = 1)

#Fitting the grid search object to the training data
grid_dtree.fit(X_train, y_train)

#Printing the best parameters
print('Best parameters found: ', grid_dtree.best_params_)
```

```
Fitting 5 folds for each of 400 candidates, totalling 2000 fits
Best parameters found:  {'criterion': 'gini', 'max_depth': 6,
'min_samples_leaf': 10, 'random_state': 42}
```

```
dtree = DecisionTreeClassifier(criterion='gini', max_depth=6,
random_state=42, min_samples_leaf=10)
dtree
```

```
DecisionTreeClassifier(max_depth=6, min_samples_leaf=10,
random_state=42)
```

```
#training the model
dtree.fit(X_train,y_train)
#training accuracy
dtree.score(X_train,y_train)
```

```
0.8581428571428571
```

```
dtree_pred = dtree.predict(X_test)
```

Random Forest

```
from sklearn.ensemble import RandomForestClassifier
#creating Random Forest Classifer object
rfc = RandomForestClassifier()

#defining parameter range
param_grid = {
    'max_depth': [2,4,6,8,10],
    'min_samples_leaf': [2,4,6,8,10],
    'criterion': ['gini', 'entropy'],
    'random_state': [0,42]
    }

#Creating grid search object
grid_rfc = GridSearchCV(rfc, param_grid, cv = 5, scoring = 'roc_auc',
n_jobs = -1, verbose = 1)

#Fitting the grid search object to the training data
grid_rfc.fit(X_train, y_train)

#Printing the best parameters
print('Best parameters found: ', grid_rfc.best_params_)
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
Best parameters found:  {'criterion': 'entropy', 'max_depth': 10,
'min_samples_leaf': 8, 'random_state': 0}
```

```
rfc = RandomForestClassifier(min_samples_leaf=8, max_depth=10,
random_state=0, criterion='entropy')
rfc
```

```
RandomForestClassifier(criterion='entropy', max_depth=10,
min_samples_leaf=8,
                       random_state=0)
```

```
#training the model
rfc.fit(X_train, y_train)
#model accuracy
rfc.score(X_train, y_train)
```

```
0.8767142857142857
```

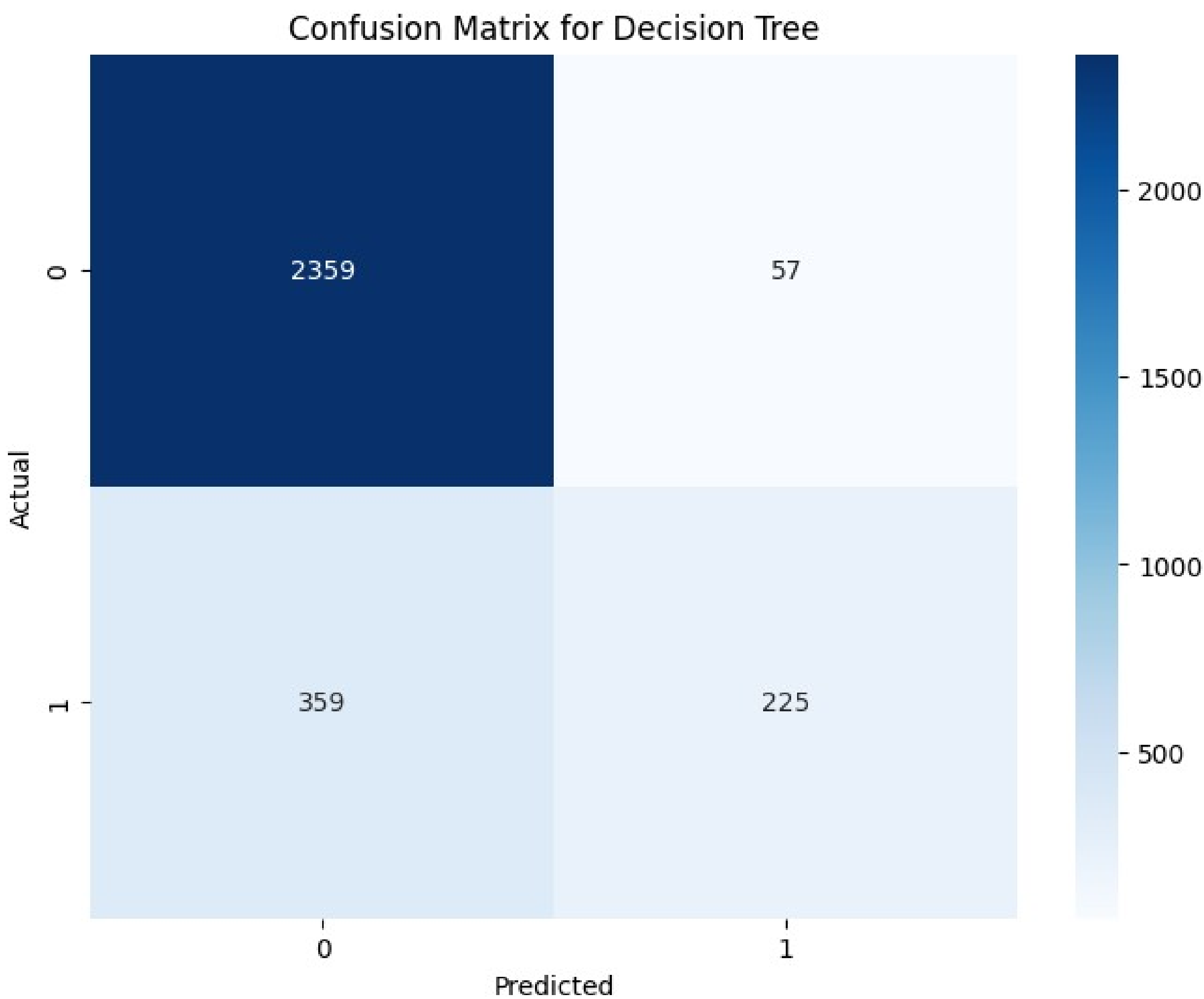
```
rfc_pred = rfc.predict(X_test)
```

Model Evaluation

Decison Tree

Confusion Matroks Headmap

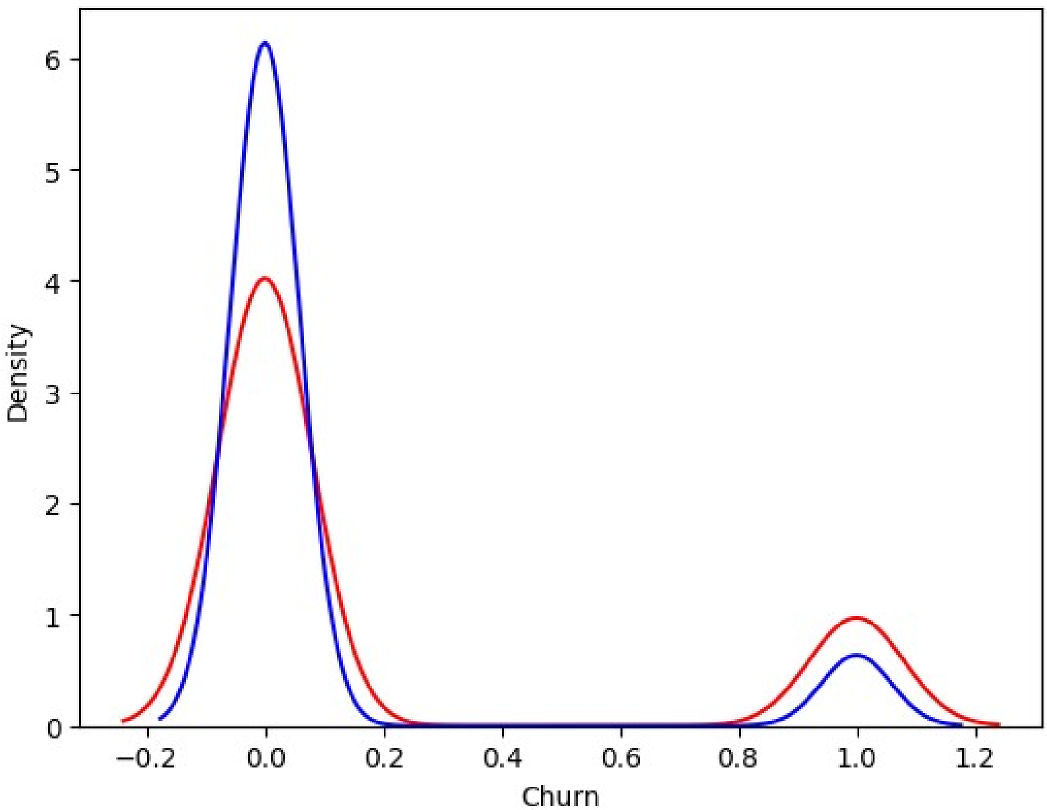
```
from sklearn.metrics import confusion_matrix
plt.figure(figsize=(8,6))
sns.heatmap(confusion_matrix(y_test,dtree_pred),annot=True,fmt='d',cma
p='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix for Decision Tree')
plt.show()
```



The True Positive shows the count of correctly classified data points whereas the False Positive elements are those that are misclassified by the model. The higher the True Positive values of the confusion matrix the better, indicating many correct predictions.

```
ax = sns.distplot(y_test, hist=False, color="r", label="Actual Value")
sns.distplot(dtree_pred, hist=False, color="b", label="Fitted
Values" , ax=ax)
```

```
<ipython-input-39-584003bfddf2>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

Please adapt your code to use either `displot` (a figure-level
function with
similar flexibility) or `kdeplot` (an axes-level function for kernel
density plots).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  ax = sns.distplot(y_test, hist=False, color="r", label="Actual
Value")
<ipython-input-39-584003bfddf2>:2: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

Please adapt your code to use either `displot` (a figure-level
function with
similar flexibility) or `kdeplot` (an axes-level function for kernel
density plots).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(dtree_pred, hist=False, color="b", label="Fitted
Values" , ax=ax)

<Axes: xlabel='Churn', ylabel='Density'>
```



Classification Report

```
from sklearn.metrics import classification_report
print(classification_report(y_test, dtree_pred))
```

```
              precision    recall  f1-score   support

           0       0.87      0.98      0.92      2416
           1       0.80      0.39      0.52       584

    accuracy                           0.86      3000
   macro avg       0.83      0.68      0.72      3000
weighted avg       0.85      0.86      0.84      3000
```

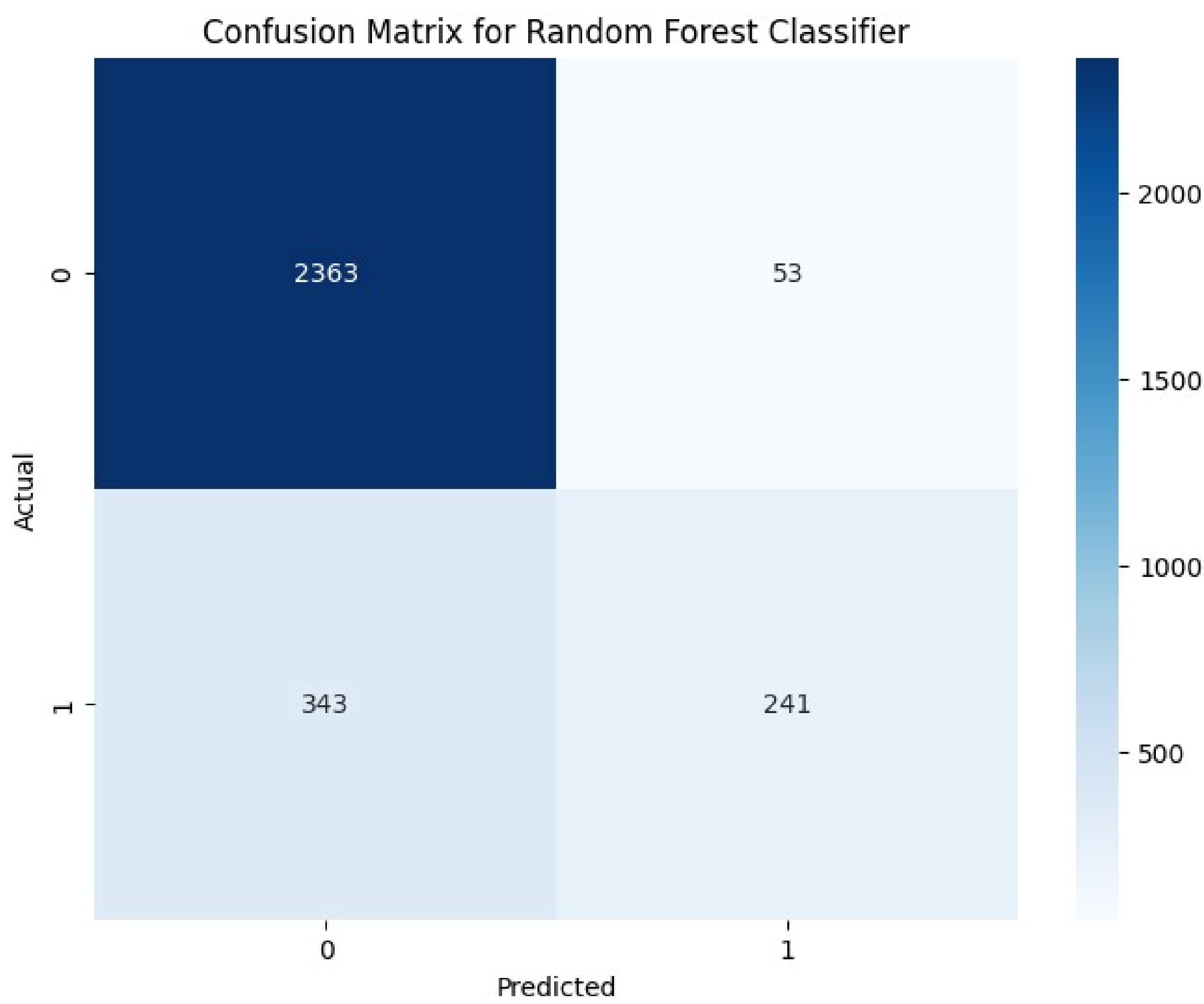
```
from sklearn.metrics import accuracy_score, mean_absolute_error,
r2_score
print("Accuracy Score: ", accuracy_score(y_test, dtree_pred))
print("Mean Absolute Error: ", mean_absolute_error(y_test,
dtree_pred))
print("R2 Score: ", r2_score(y_test, dtree_pred))
```

```
Accuracy Score:  0.8613333333333333
Mean Absolute Error:  0.13866666666666666
R2 Score:  0.11548580241313633
```

Random Forest Classifier

Confusion Matrix Heatmap

```
plt.figure(figsize=(8,6))
sns.heatmap(confusion_matrix(y_test,rfc_pred),annot=True,fmt='d',cmap=
'Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix for Random Forest Classifier')
plt.show()
```

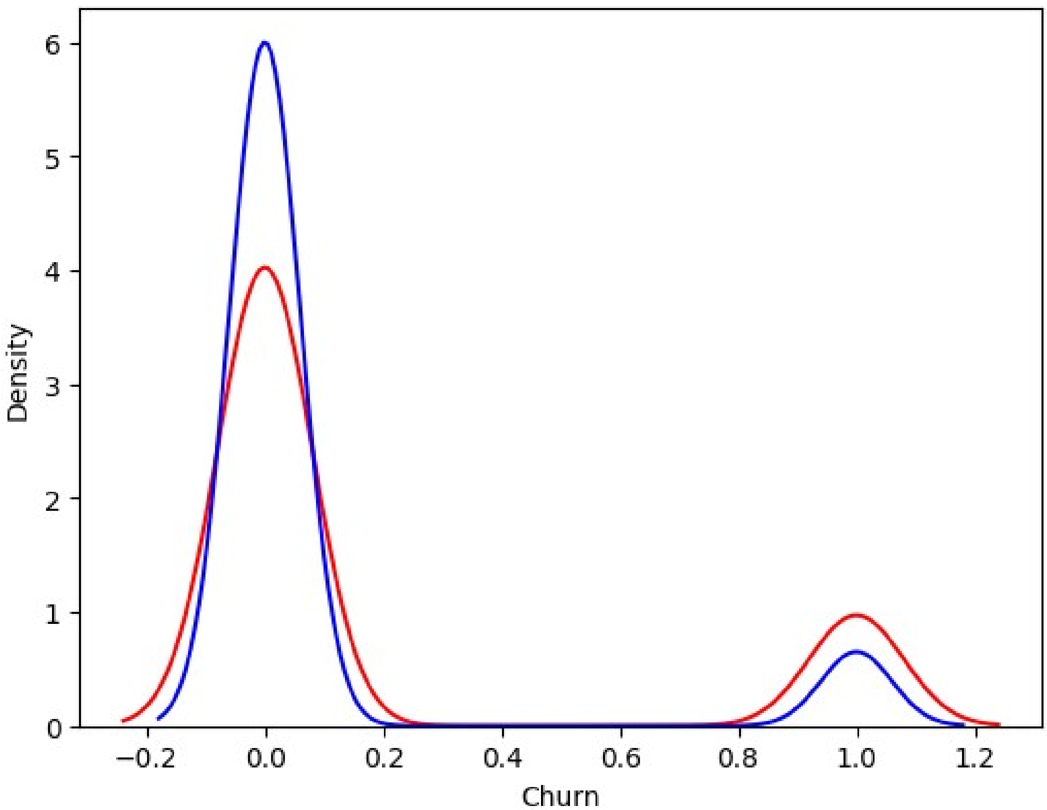


The True Positive shows the count of correctly classified data points whereas the False Positive elements are those that are misclassified by the model. The higher the True Positive values of the confusion matrix the better, indicating many correct predictions.

Distribution Plot

```
ax = sns.distplot(y_test, hist=False, color="r", label="Actual Value")
sns.distplot(rfc_pred, hist=False, color="b", label="Fitted Values" ,
ax=ax)
```

```
<ipython-input-43-333243e5e9cf>:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

Please adapt your code to use either `displot` (a figure-level
function with
similar flexibility) or `kdeplot` (an axes-level function for kernel
density plots).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  ax = sns.distplot(y_test, hist=False, color="r", label="Actual
Value")
<ipython-input-43-333243e5e9cf>:2: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

Please adapt your code to use either `displot` (a figure-level
function with
similar flexibility) or `kdeplot` (an axes-level function for kernel
density plots).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(rfc_pred, hist=False, color="b", label="Fitted
Values" , ax=ax)

<Axes: xlabel='Churn', ylabel='Density'>
```




```
from sklearn.metrics import classification_report
print(classification_report(y_test, rfc_pred))

              precision    recall  f1-score   support

           0       0.87      0.98      0.92      2416
           1       0.82      0.41      0.55       584

    accuracy                           0.87      3000
   macro avg       0.85      0.70      0.74      3000
weighted avg       0.86      0.87      0.85      3000


print("Accuracy Score: ", accuracy_score(y_test, rfc_pred))
print("Mean Absolute Error: ", mean_absolute_error(y_test, rfc_pred))
print("R2 Score: ", r2_score(y_test, rfc_pred))

Accuracy Score:  0.868
Mean Absolute Error:  0.132
R2 Score:  0.15801052345096633
```

Conclution

From the exploratory data analysis, I have concluded that the churn count of the customersdepends upon the following factors:

1. Age
2. Geography
3. Tenure
4. Balance
5. Number of Products
6. Has Credit Card
7. Is Active Member

Coming to the classification models, I have used the following models:

1. Decision Tree Classifier
2. Random Forest Classifier

Based on the two methods that have been tested, it was found that radom forest accuracy produced the greatest accuracy with a percentage of 87%. whereas the Decision Tree only produces an accuracy percentage of 86%.