ARVIO ANANDI

BANK CUSTOMER CHURN PREDICTION

TABLE OF CONTENTS

BUSINESS BACKGROUND

BUSINESS BACKGROUND

As a data scientist freelancer collaborating with a prominent bank in the EU, my mission revolves around predicting customers who are likely to churn. The bank entrusted me with the task of leveraging advanced analytics to identify this specific group. By doing so, the bank can precisely target this smaller audience group for marketing efforts, leading to significant savings in costs and time. This strategic approach does not only optimize resources but also ensures a focused and effective retention strategy, enhancing both customer satisfaction and the bank's overall efficiency.

BUSINESS OBJECTIVES

The primary business objective is to develop a highly accurate customer churn prediction model. This model will enable the bank to identify customers at risk of leaving, allowing for more actionable efforts to be made. By minimizing churn through strategic retention initiatives, the bank aims to enhance customer loyalty, optimize marketing costs, and improve overall customer satisfaction. Additionally, the project seeks to establish a data-driven culture within the bank, fostering innovation and ensuring long-term competitiveness in the financial market.

OUTPUT

The output of this project is a model to predict the customers who are at risk of leaving the bank given its features.

The features include Age, CreditScore, EstimatedSalary, etc.

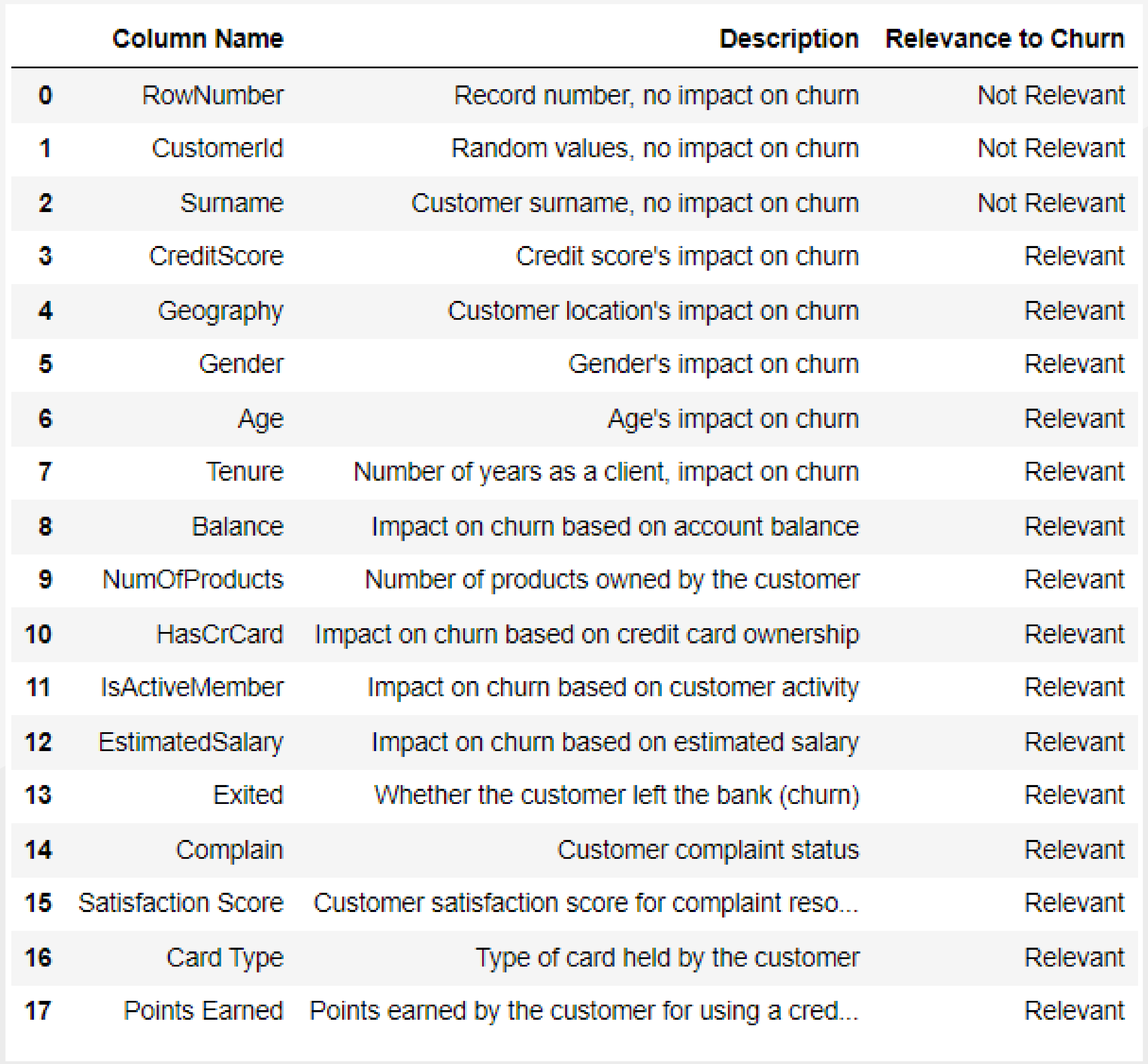
DATA UNDERSTANDING

DATA SOURCE

kaggle™

The dataset was acquired from Kaggle, with 10000 rows and 18 columns.

COLUMN EXPLANATION



	Column Name	Description	Relevance to Churn
0	RowNumber	Record number, no impact on churn	Not Relevant
1	CustomerId	Random values, no impact on churn	Not Relevant
2	Surname	Customer surname, no impact on churn	Not Relevant
3	CreditScore	Credit score's impact on churn	Relevant
4	Geography	Customer location's impact on churn	Relevant
5	Gender	Gender's impact on churn	Relevant
6	Age	Age's impact on churn	Relevant
7	Tenure	Number of years as a client, impact on churn	Relevant
8	Balance	Impact on churn based on account balance	Relevant
9	NumOfProducts	Number of products owned by the customer	Relevant
10	HasCrCard	Impact on churn based on credit card ownership	Relevant
11	IsActiveMember	Impact on churn based on customer activity	Relevant
12	EstimatedSalary	Impact on churn based on estimated salary	Relevant
13	Exited	Whether the customer left the bank (churn)	Relevant
14	Complain	Customer complaint status	Relevant
15	Satisfaction Score	Customer satisfaction score for complaint reso...	Relevant
16	Card Type	Type of card held by the customer	Relevant
17	Points Earned	Points earned by the customer for using a cred...	Relevant

ANALYTIC APPROACH

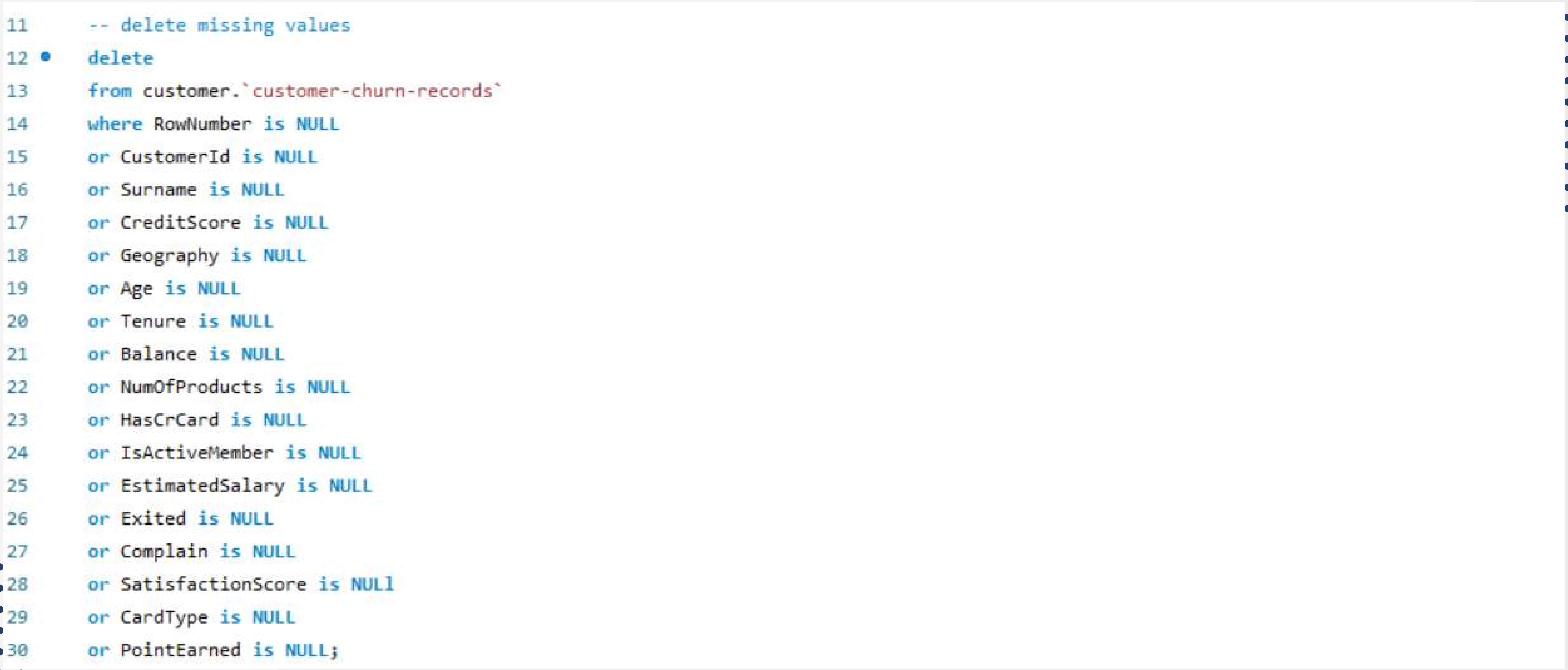
Supervised Learning (Binary Classification)
To classify customers into groups of churned and not churned.

Metrics
AUC_ROC, Precision, F1, Recall, and Accuracy as support for our prediction

DATA SPLIT RATIO

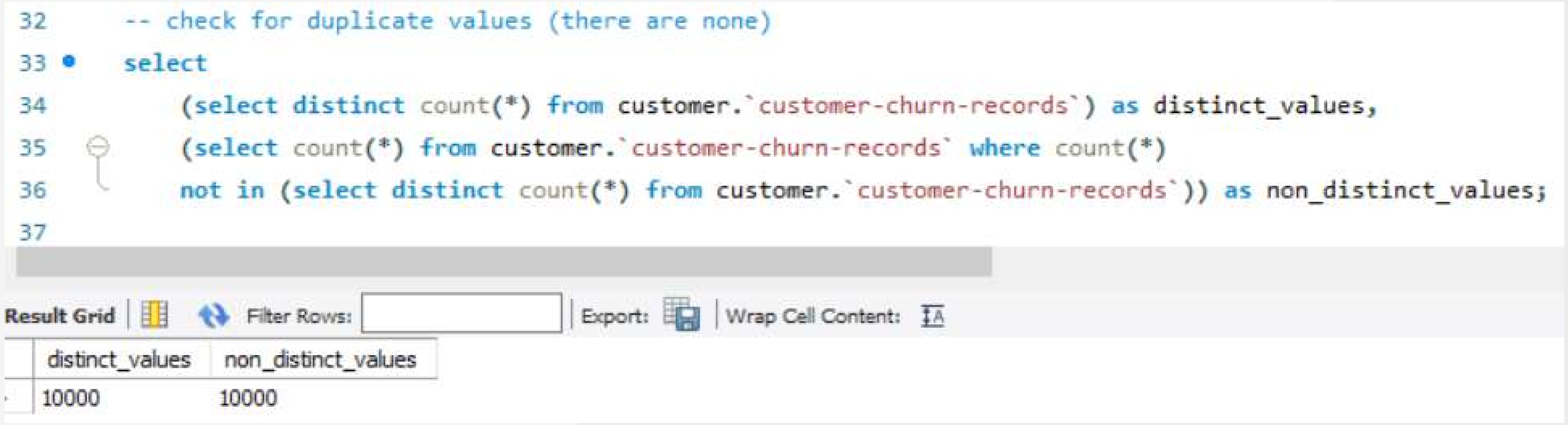
We split the data into the ratio of 80:20, where 80% is for training and 20% for testing and evaluating the model.

DATA WRANGLING



```sql
-- delete missing values
delete
from customer.`customer-churn-records`
where RowNumber is NULL
or CustomerId is NULL
or Surname is NULL
or CreditScore is NULL
or Geography is NULL
or Age is NULL
or Tenure is NULL
or Balance is NULL
or NumOfProducts is NULL
or HasCrCard is NULL
or IsActiveMember is NULL
or EstimatedSalary is NULL
or Exited is NULL
or Complain is NULL
or SatisfactionScore is NULl
or CardType is NULL
or PointEarned is NULL;
```

Removed missing values

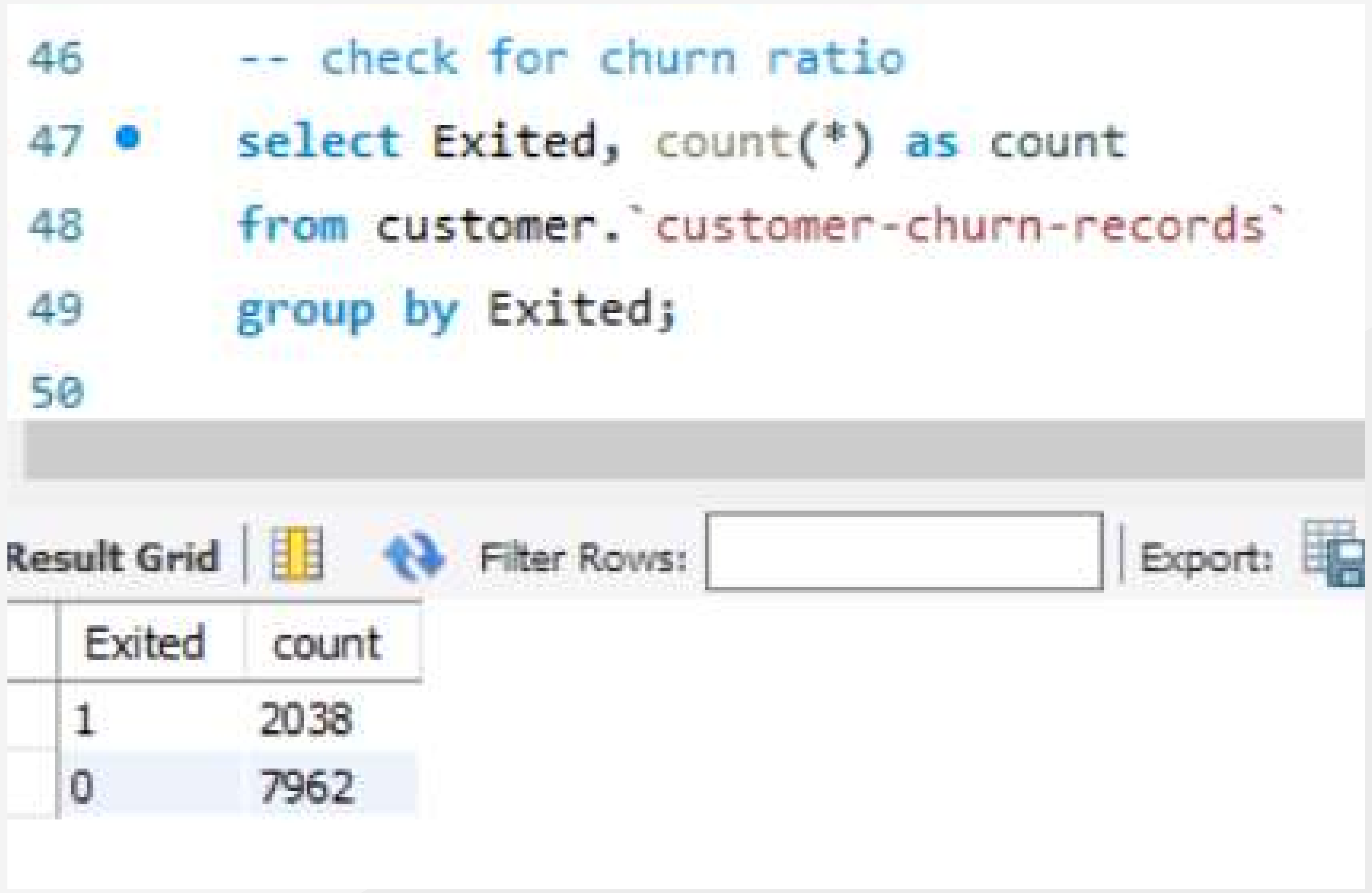


```sql
-- check for duplicate values (there are none)
select
    (select distinct count(*) from customer.`customer-churn-records`) as distinct_values,
    (select count(*) from customer.`customer-churn-records` where count(*)
    not in (select distinct count(*) from customer.`customer-churn-records`)) as non_distinct_values;
```

```
distinct_values   non_distinct_values
10000             10000
```

There seems to be no duplicates as the distinct values and non distinct value comparison is the same

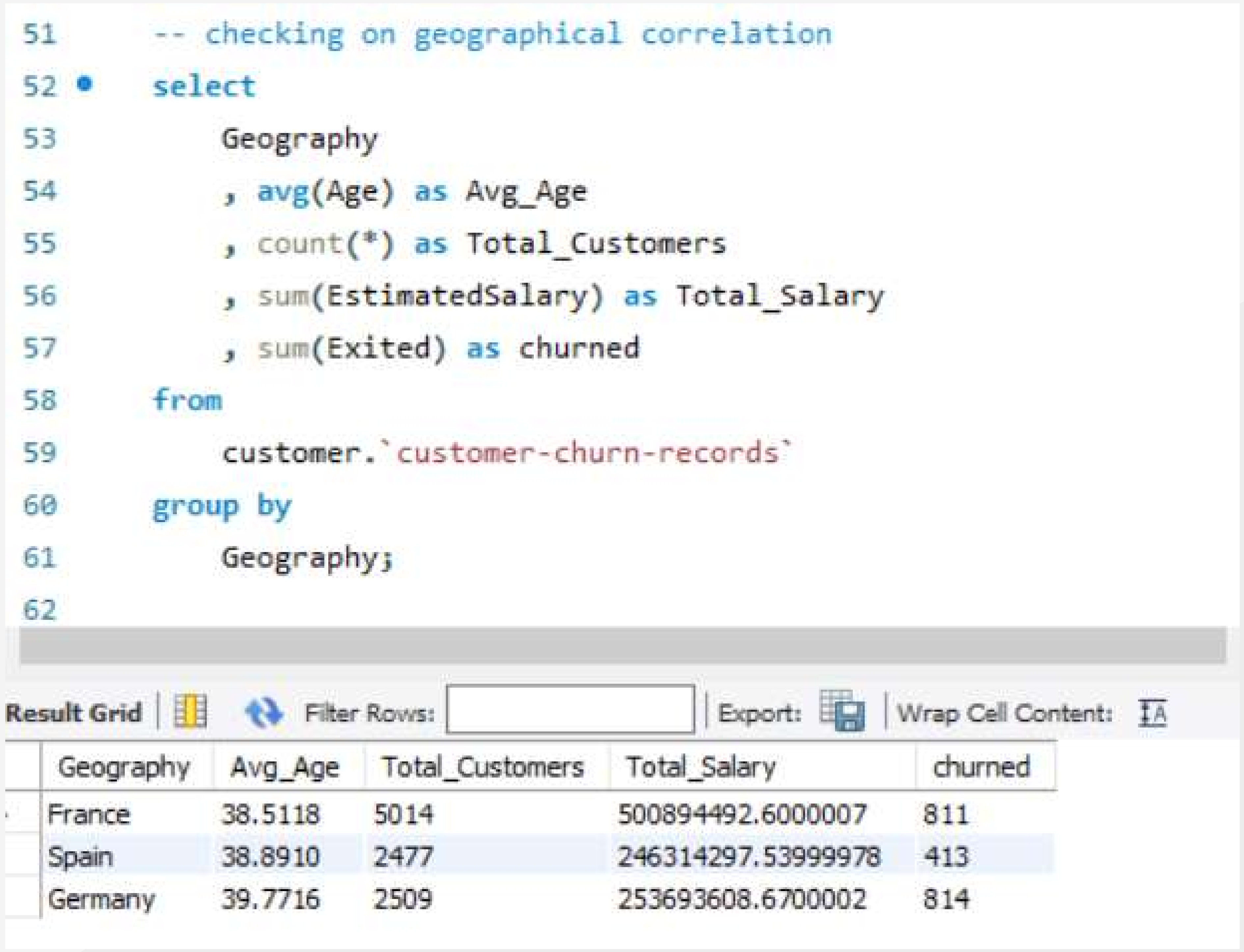
EXPLORATORY DATA ANALYSIS (SQL)



```
-- check for churn ratio
select Exited, count(*) as count
from customer.`customer-churn-records`
group by Exited;
```

Exited	count
1	2038
0	7962

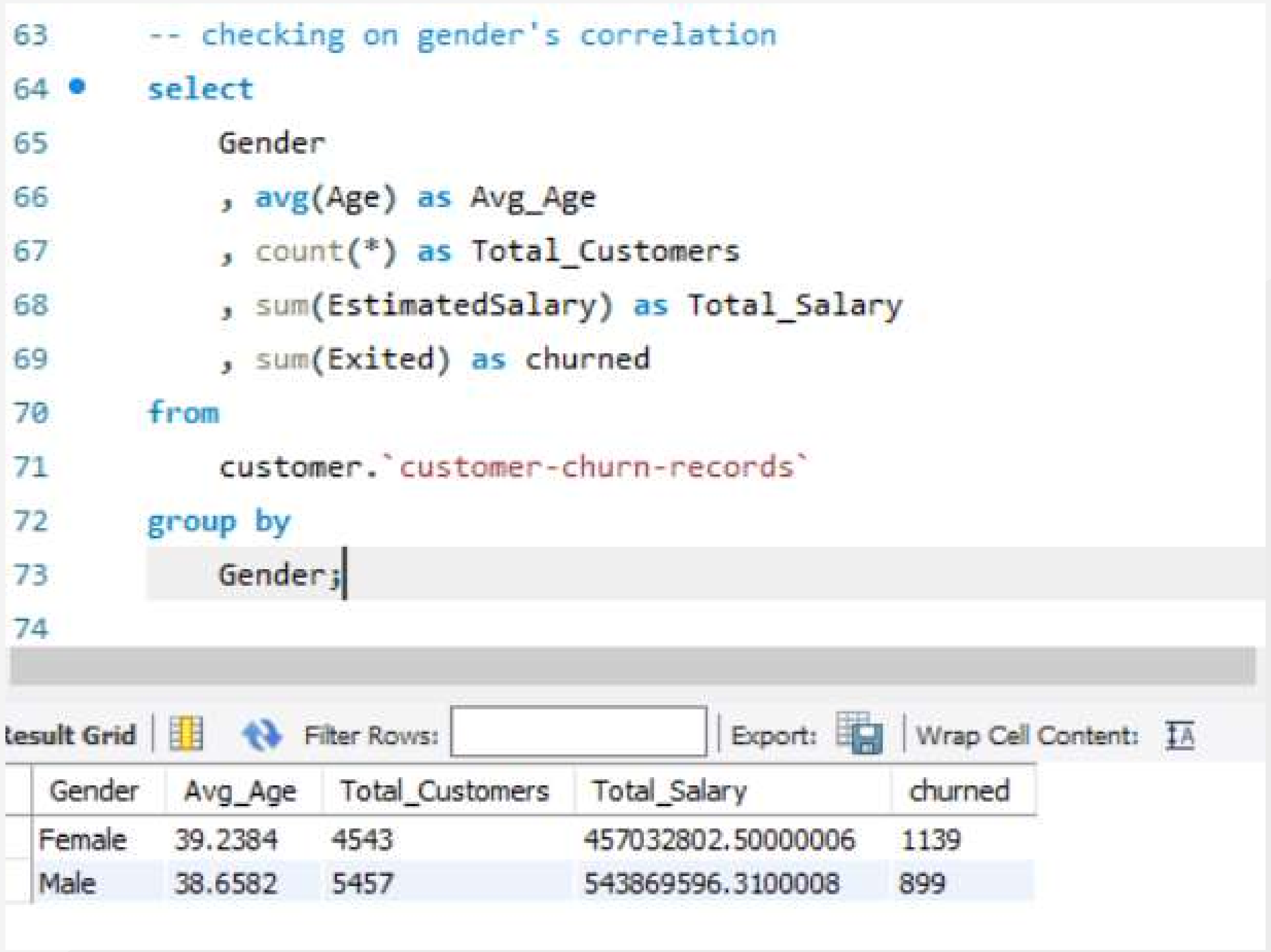
This table shows that there seems to be data imbalance for churned and not churned customers. This explains that roughly 80% of customers decided not to churn from the bank.



```
-- checking on geographical correlation
select
    Geography
    , avg(Age) as Avg_Age
    , count(*) as Total_Customers
    , sum(EstimatedSalary) as Total_Salary
    , sum(Exited) as churned
from
    customer.`customer-churn-records`
group by
    Geography;
```

Geography	Avg_Age	Total_Customers	Total_Salary	churned
France	38.5118	5014	500894492.6000007	811
Spain	38.8910	2477	246314297.53999978	413
Germany	39.7716	2509	253693608.6700002	814

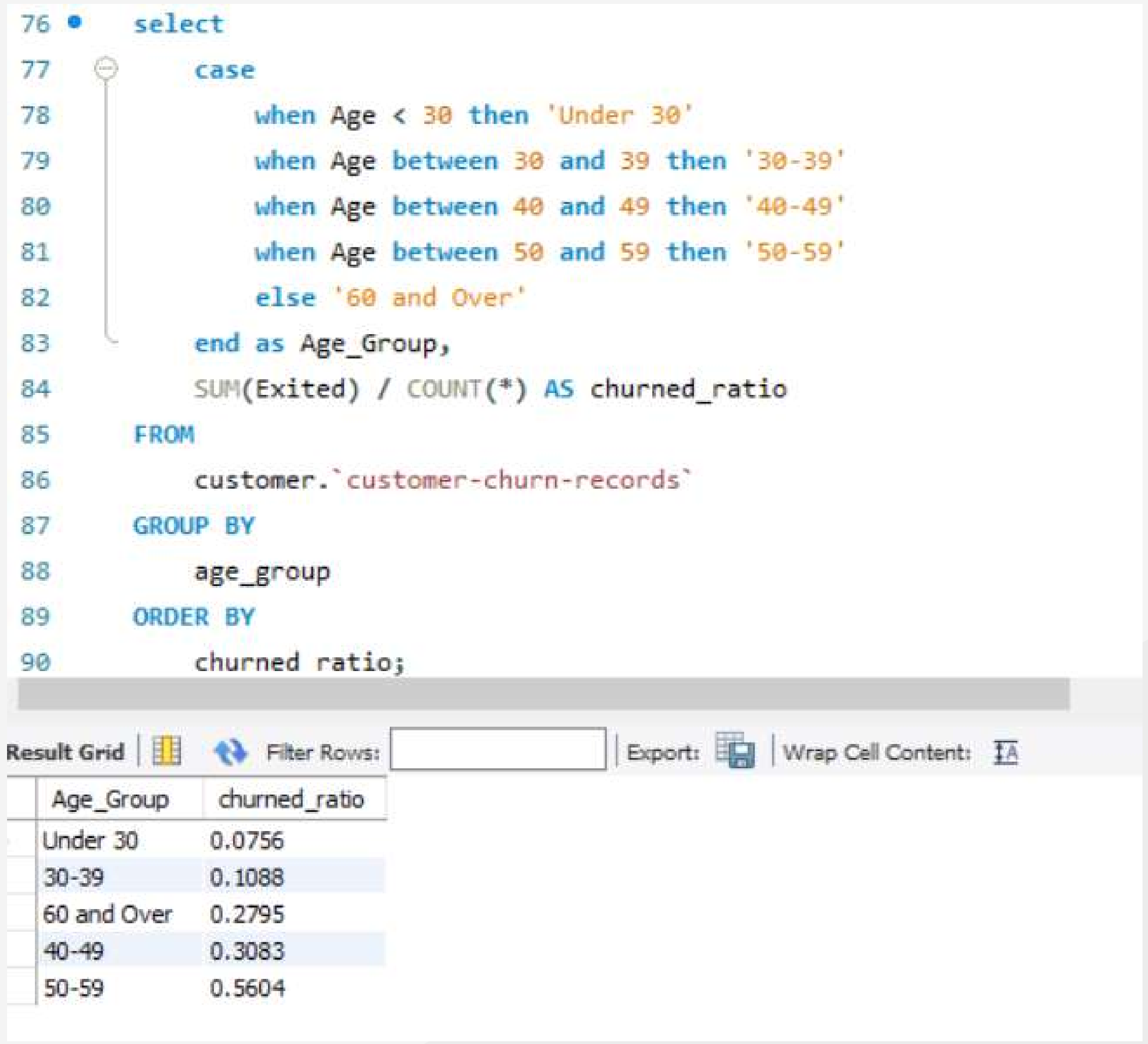
The table shows the distribution of churned customers from different countries. This tells us that people from France and Germany are more likely to churn than people from Spain.



```sql
-- checking on gender's correlation
select
    Gender
    , avg(Age) as Avg_Age
    , count(*) as Total_Customers
    , sum(EstimatedSalary) as Total_Salary
    , sum(Exited) as churned
from
    customer.`customer-churn-records`
group by
    Gender;
```

Gender	Avg_Age	Total_Customers	Total_Salary	churned
Female	39.2384	4543	457032802.50000006	1139
Male	38.6582	5457	543869596.3100008	899

From the given dataset, female customers tend to churn more than male customers. This could provide some insights and be explored further.



```
select
    case
        when Age < 30 then 'Under 30'
        when Age between 30 and 39 then '30-39'
        when Age between 40 and 49 then '40-49'
        when Age between 50 and 59 then '50-59'
        else '60 and Over'
    end as Age_Group,
    SUM(Exited) / COUNT(*) AS churned_ratio
FROM
    customer.`customer-churn-records`
GROUP BY
    age_group
ORDER BY
    churned_ratio;
```

Age_Group	churned_ratio
Under 30	0.0756
30-39	0.1088
60 and Over	0.2795
40-49	0.3083
50-59	0.5604

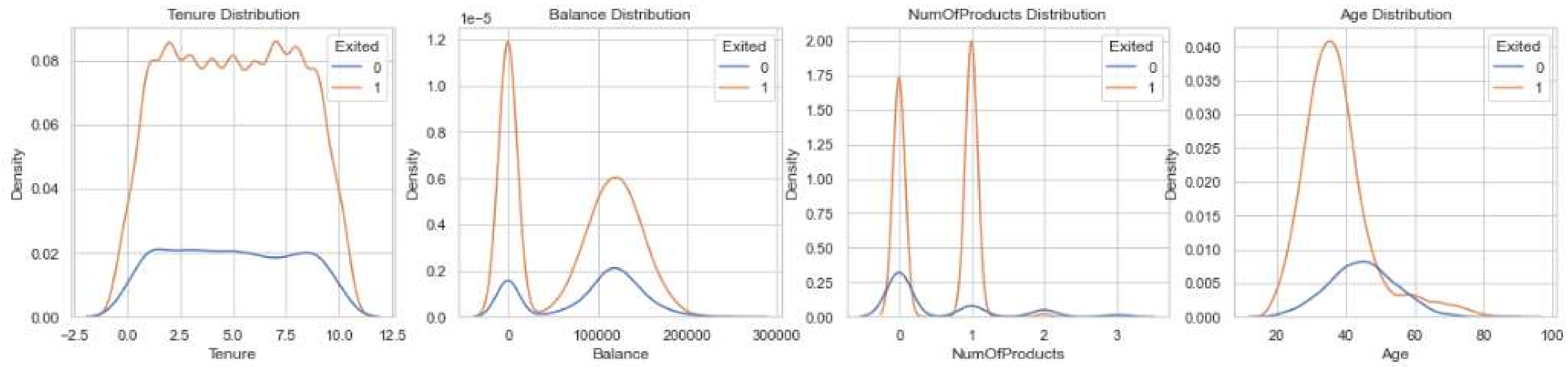
People under 39 and over 60 years of age have a fair share of not churning from the bank. For the young group, this could be explained by their digitalized environment where information are easily consumed which explains their financial literacy that helps to determine their long-term bank decisions whereas for the seniors, this could be explained by a more stable financial decisions, and their long-term relationships with banks which corresponds to them not switching banks

For a better grasp of exploration and understanding of underlying features , we perform Exploratory Data Analysis in python.

EXPLORATORY DATA ANALYSIS (PYTHON)

Numerical Columns Distributions

```
[77]: fig, ax = plt.subplots(1, 4, figsize=(20, 4))
sns.kdeplot(x=df['Tenure'], ax=ax[0], hue=df['Exited'])
ax[0].set_title("Tenure Distribution")
sns.kdeplot(x=df['Balance'], hue=df['Exited'], ax=ax[1])
ax[1].set_title("Balance Distribution")
sns.kdeplot(x=df['NumOfProducts'], hue=df['Exited'], ax=ax[2])
ax[2].set_title("NumOfProducts Distribution")
sns.kdeplot(x=df['Age'], hue=df['Exited'], ax=ax[3])
ax[3].set_title("Age Distribution")

plt.show()
```



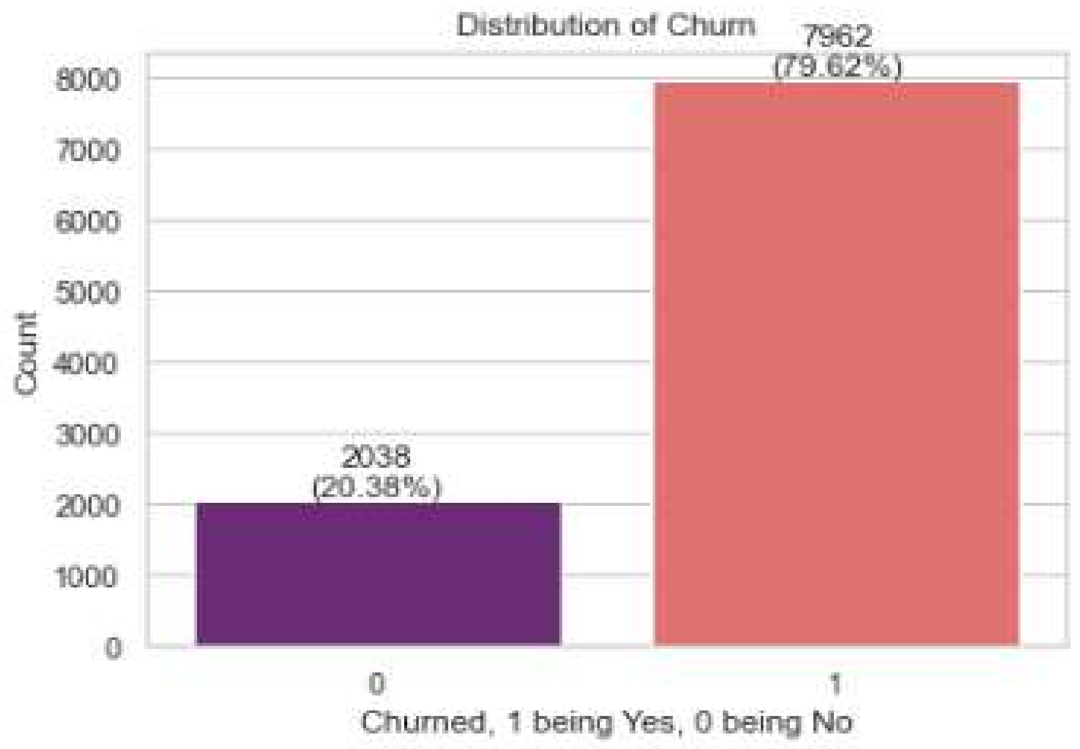
1. The Tenure density distribution is randomly distributed. We cannot infer much from this plot.
2. The Balance density distribution is concentrated on customers with balance from -25000 to 230000.
3. The NumOfProducts density distribution are mainly distributed on 1 and 2 products, with more churned customers condensed from 0.5 to 1.5 number of products.
4. Age distribution are right skewed, having more distribution around the age of 15-60.

Few takeaways:

- The highest number of customers who churned purchased 0 and 1 products from the bank.
- The highest number of customers who churned has balance in their credit card ranging between -30000-210000.
- The highest number of customers who churned are distributed around the age of 18-70.

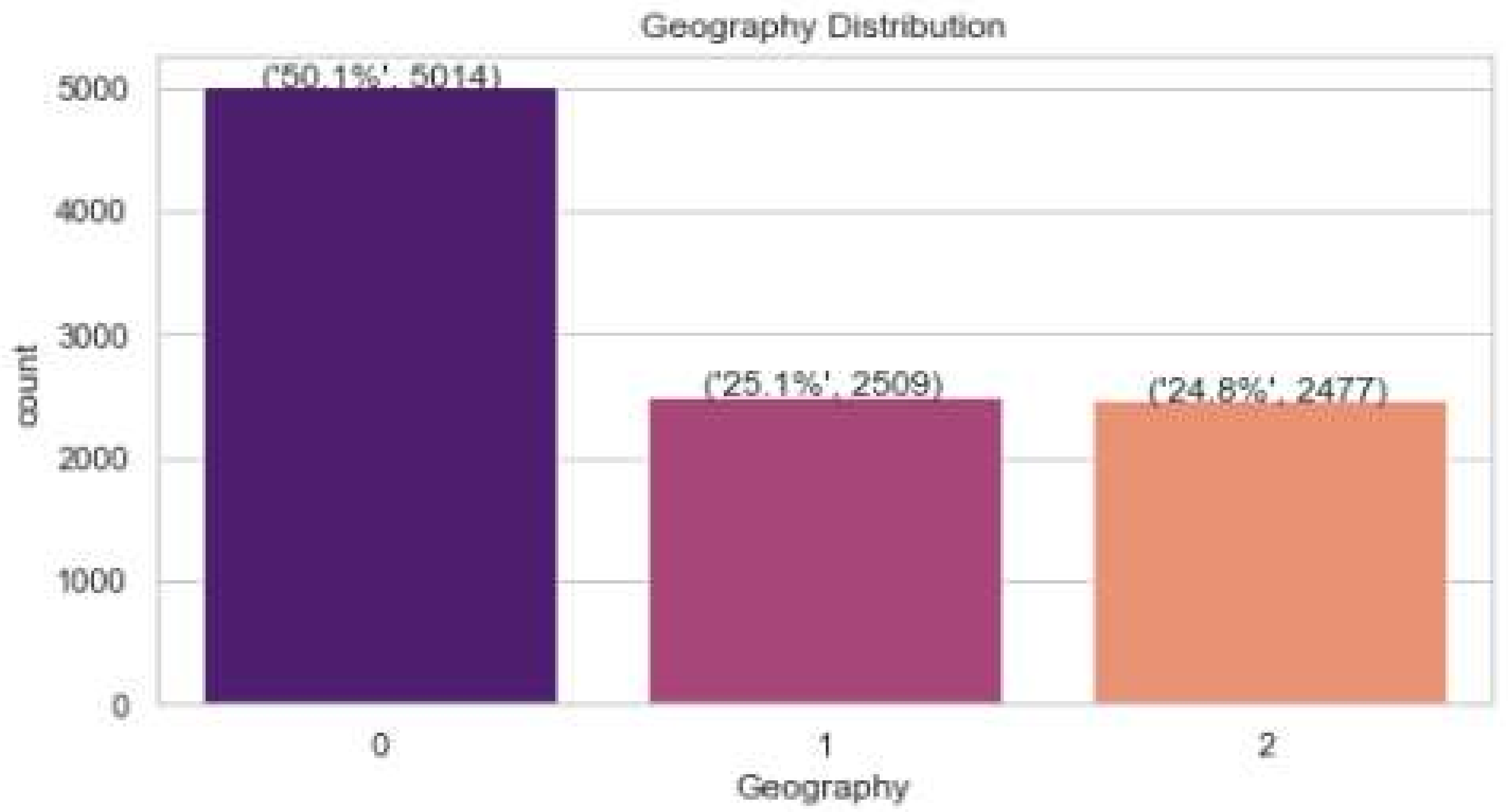
Categorical Features Distributions

```
[55]: # Create countplot to understand distribution
sns.set(style="whitegrid")
ax = sns.countplot(x=df['Exited'], palette="magma")
plt.xlabel('Churned, 1 being Yes, 0 being No')
plt.ylabel('Count')
plt.title('Distribution of Churn')
# Calculate churn count and percentage
total_count = len(df)
churn_count = df['Exited'].sum()
churn_percentage = (churn_count / total_count) * 100

# Add count and percentage labels on top of the bars
for p in ax.patches:
    height = p.get_height()
    ax.annotate(f'{height}\n({(height / total_count * 100):.2f}%)',
                (p.get_x() + p.get_width() / 2., height),
                ha='center', va='center',
                xytext=(0, 10),
                textcoords='offset points')
plt.show()
```



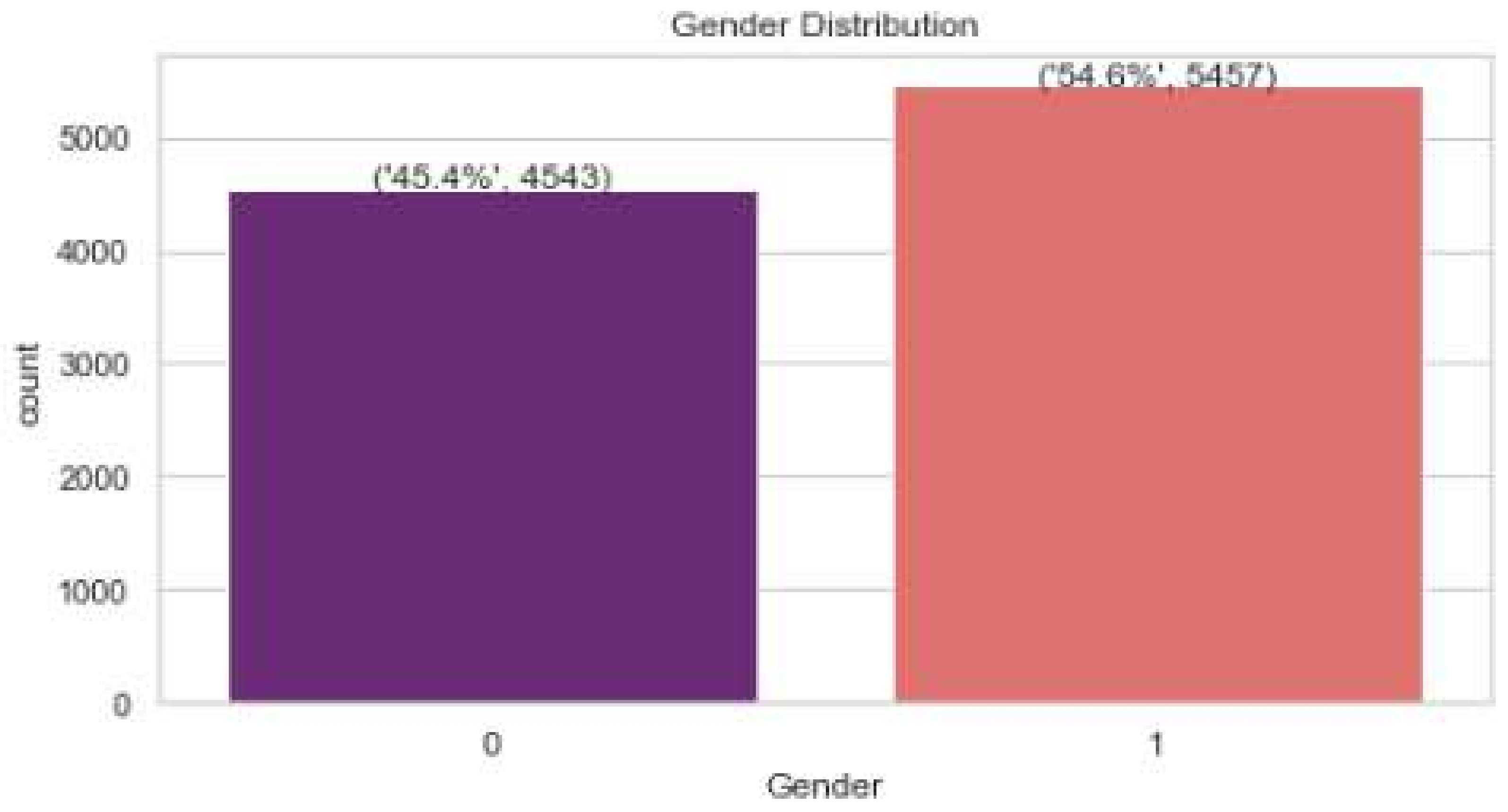
There are 20.38% of customers who churned from the Bank, This is quite a small amount.

```
plt.figure(figsize=(8,4))
ax = sns.countplot(x = df['Geography'],palette="magma")
add_labels(df, 'Geography', ax)
plt.title("Geography Distribution")
plt.show()
```



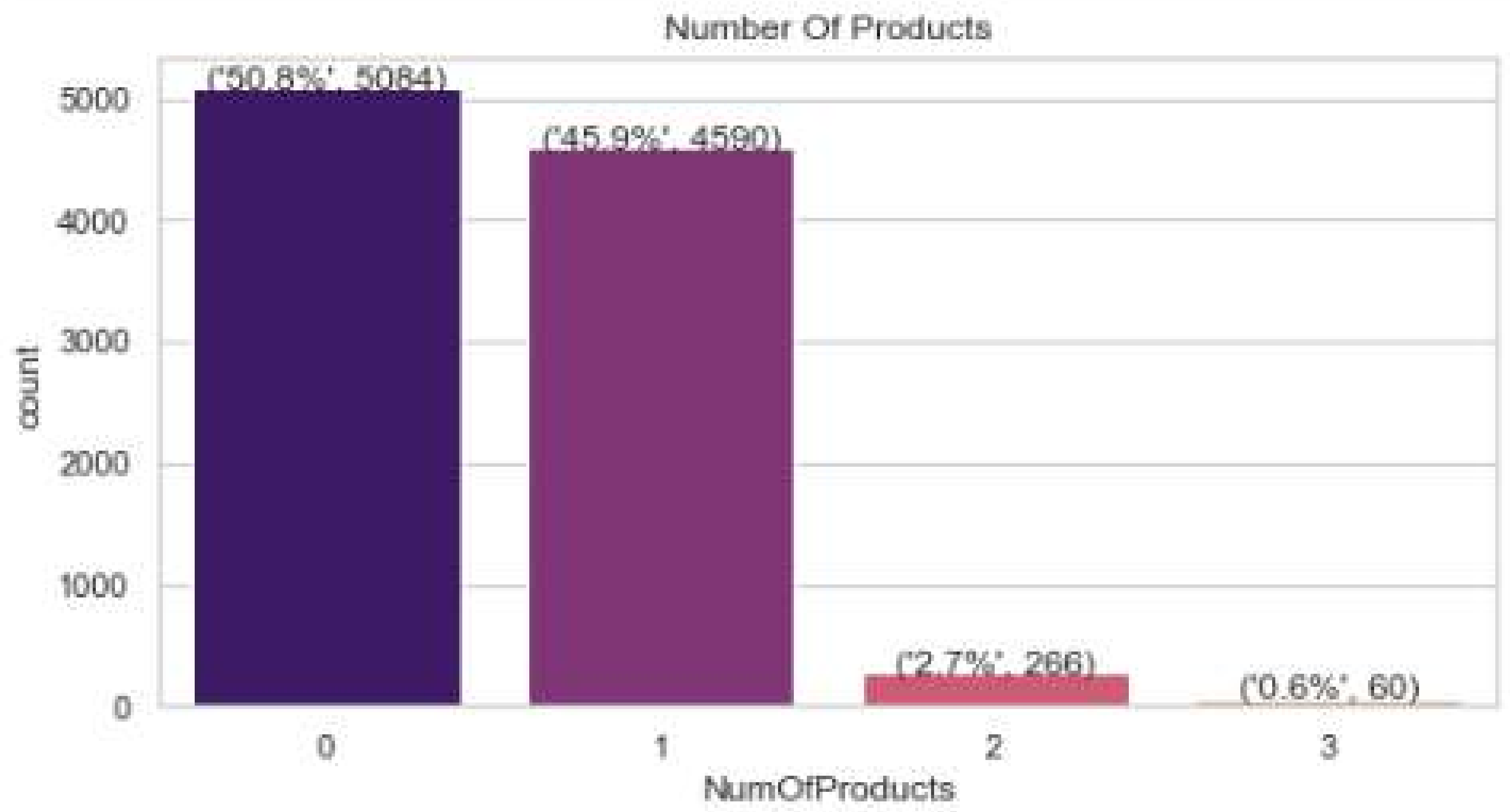
Most of the customers are from france which took half proportion of the distribution, followed by Spain and Germany with distributions around 25%.

```python
plt.figure(figsize=(8,4))
ax = sns.countplot(x = df['Gender'],palette="magma")
add_labels(df, 'Gender',ax)
plt.title("Gender Distribution")
plt.show()
```



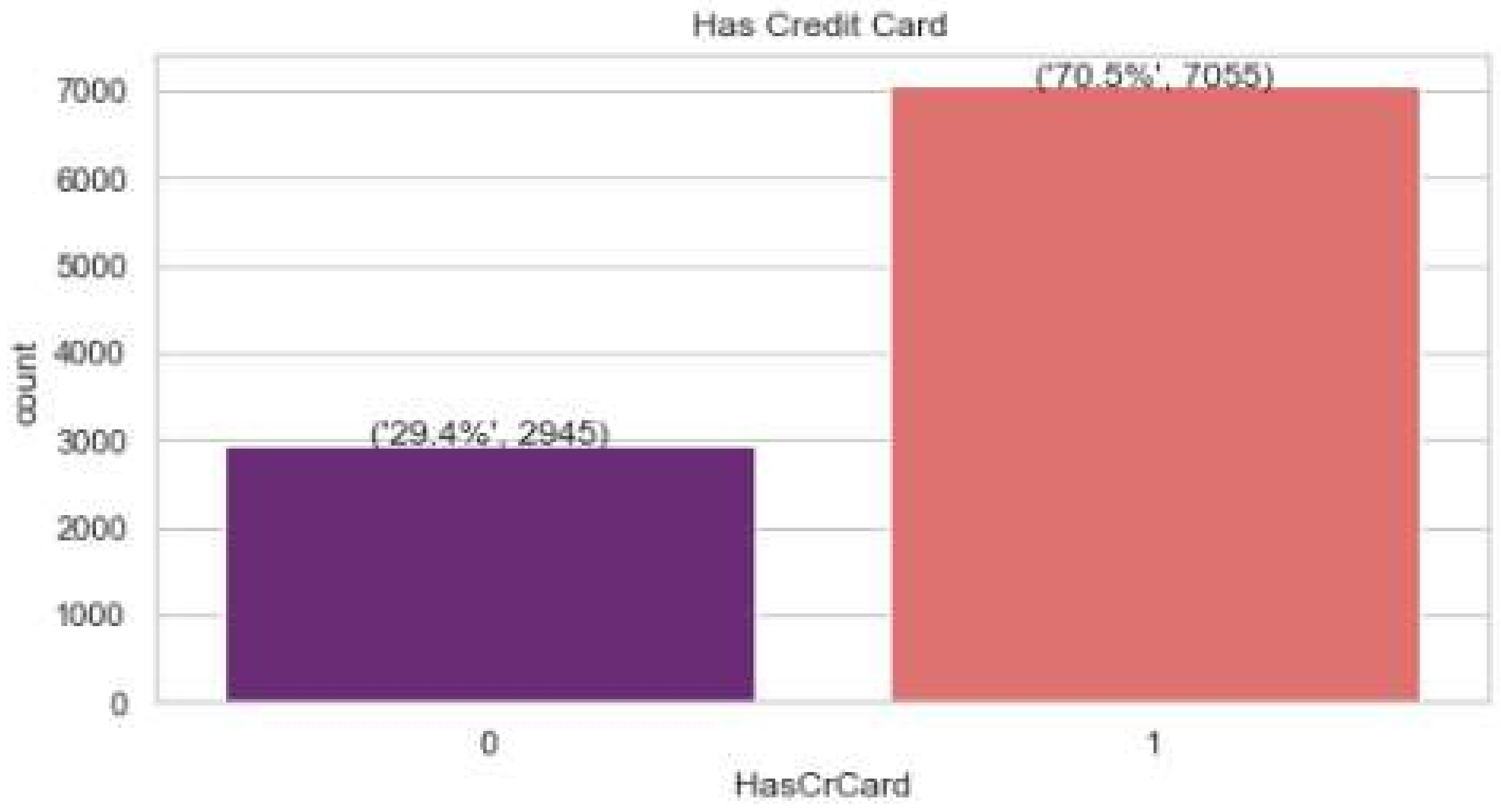
There are more portion of males compared to females in the customer gender distribution. Males are comprised of 54.6% of the population, while females taking 45.4% of the distribution.

```
plt.figure(figsize=(8,4))
ax = sns.countplot(x = df['NumOfProducts'],palette="magma")
add_labels(df, 'NumOfProducts',ax)
plt.title("Number Of Products")
plt.show()
```



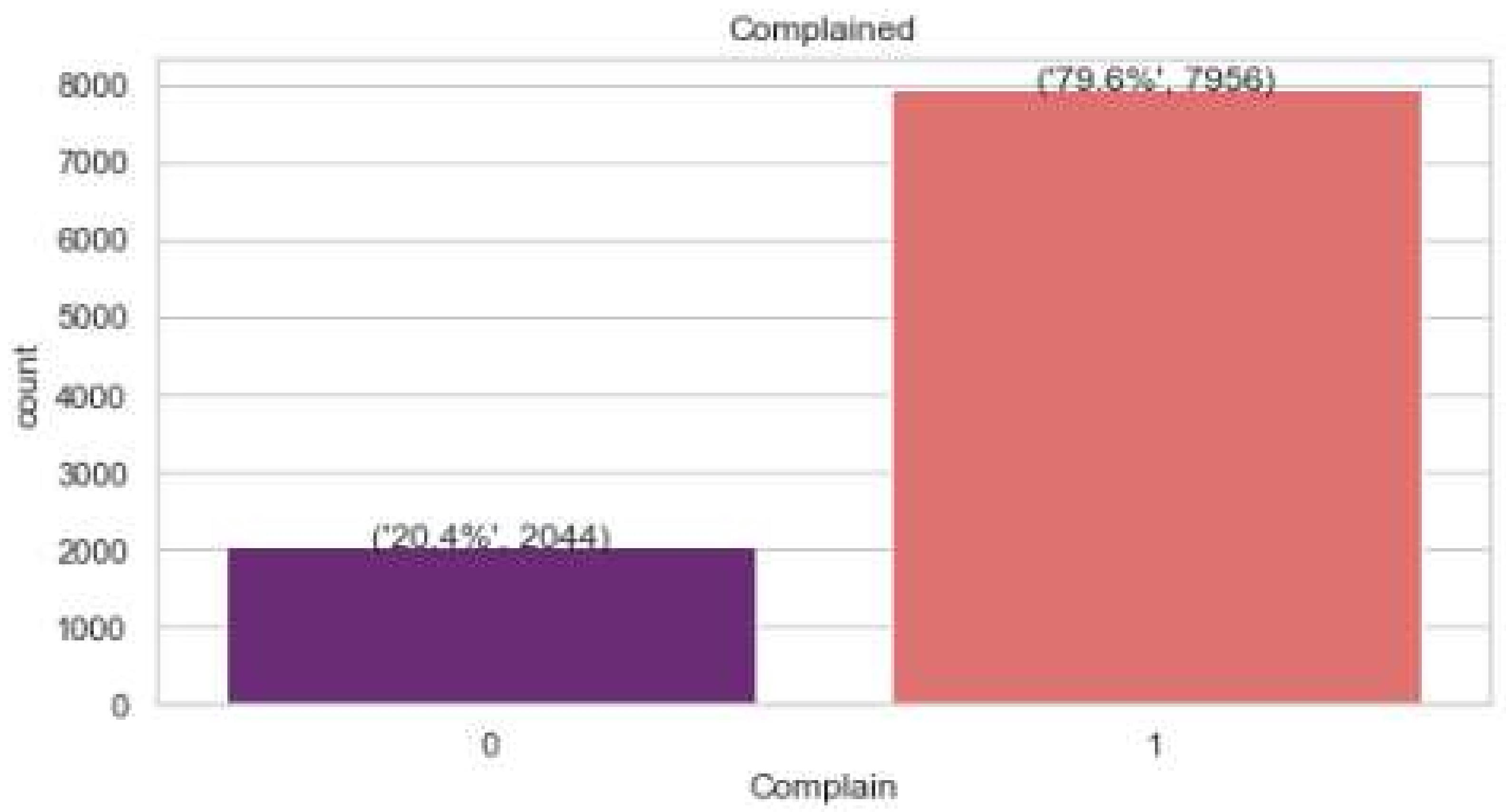
The plot suggests that the majority of customers purchased only 1 or 2 products from the bank.

```
plt.figure(figsize=(8,4))
ax = sns.countplot(x = df['HasCrCard'],palette="magma")
add_labels(df, 'HasCrCard',ax)
plt.title("Has Credit Card")
plt.show()
```



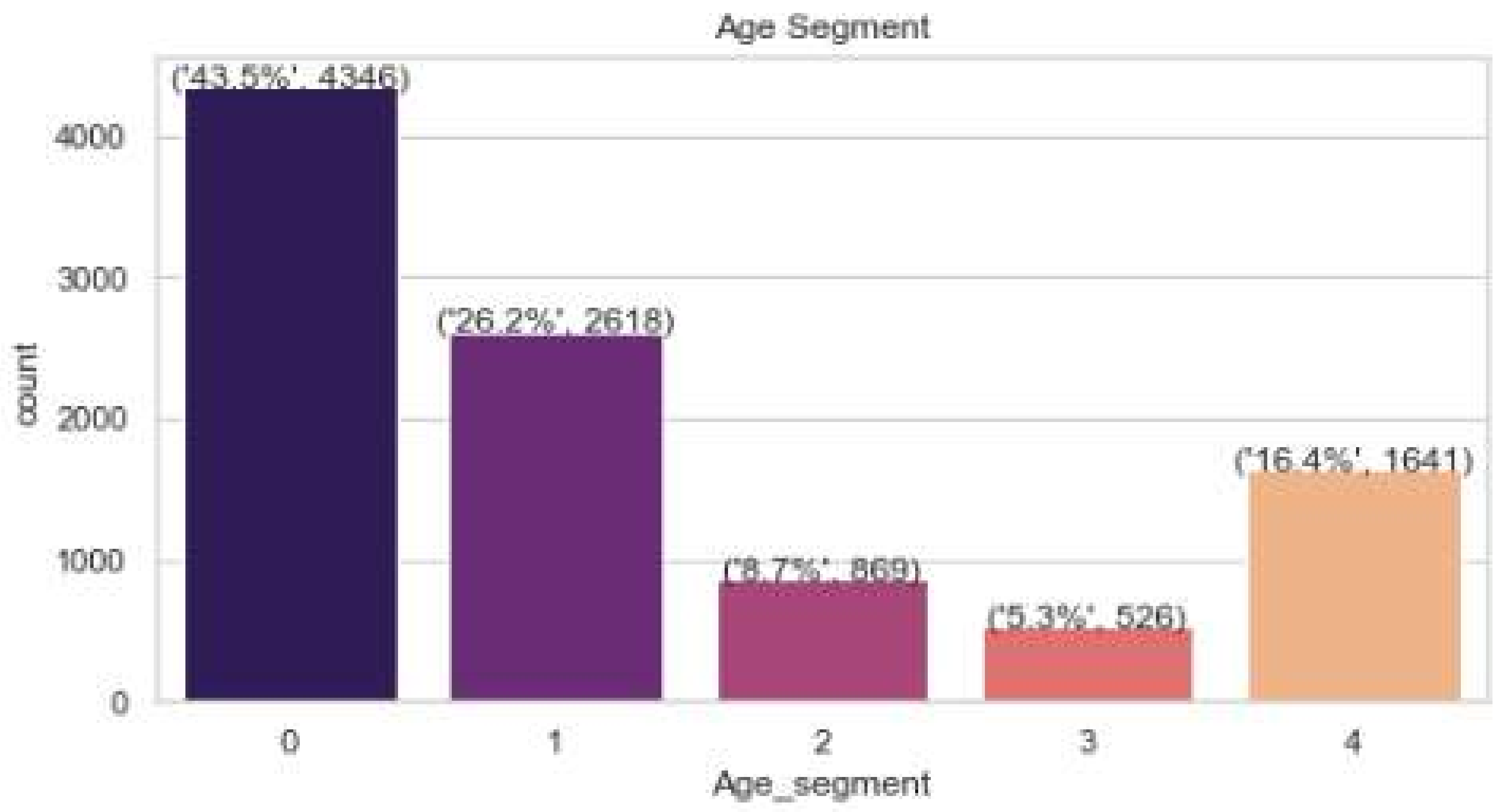
We can see that 70.5% of customers possess a credit card from the bank. This is a fairly big gap between the two classification.

```
plt.figure(figsize=(8,4))
ax = sns.countplot(x = df['Complain'],palette="magma")
add_labels(df, 'Complain',ax)
plt.title("Complained")
plt.show()
```

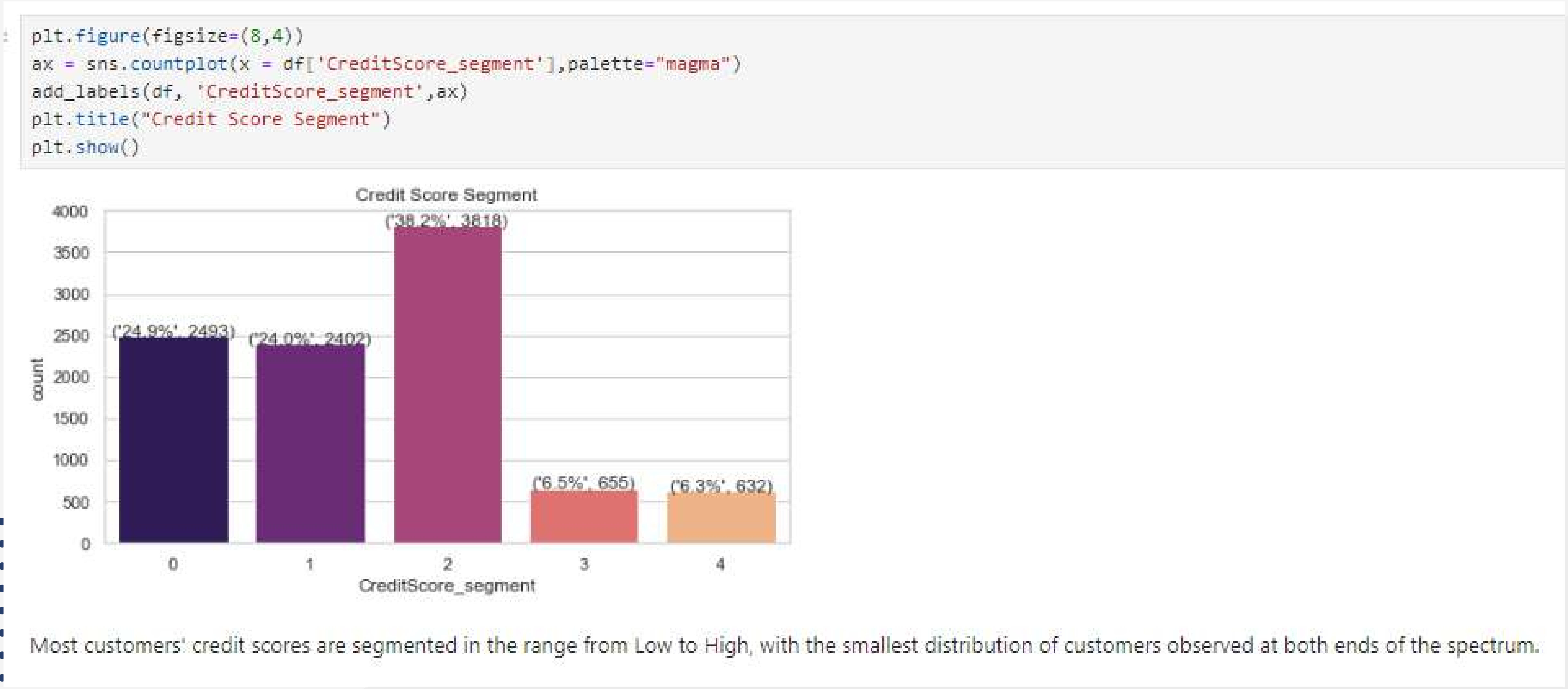


The number of complained customers has a smaller proportion of the distribution compared to customers who dont complain, indicating its relatively lower occurrence in the dataset. Despite of it, there could still be insights to be gathered from this column, so we would keep this feature moving forward.

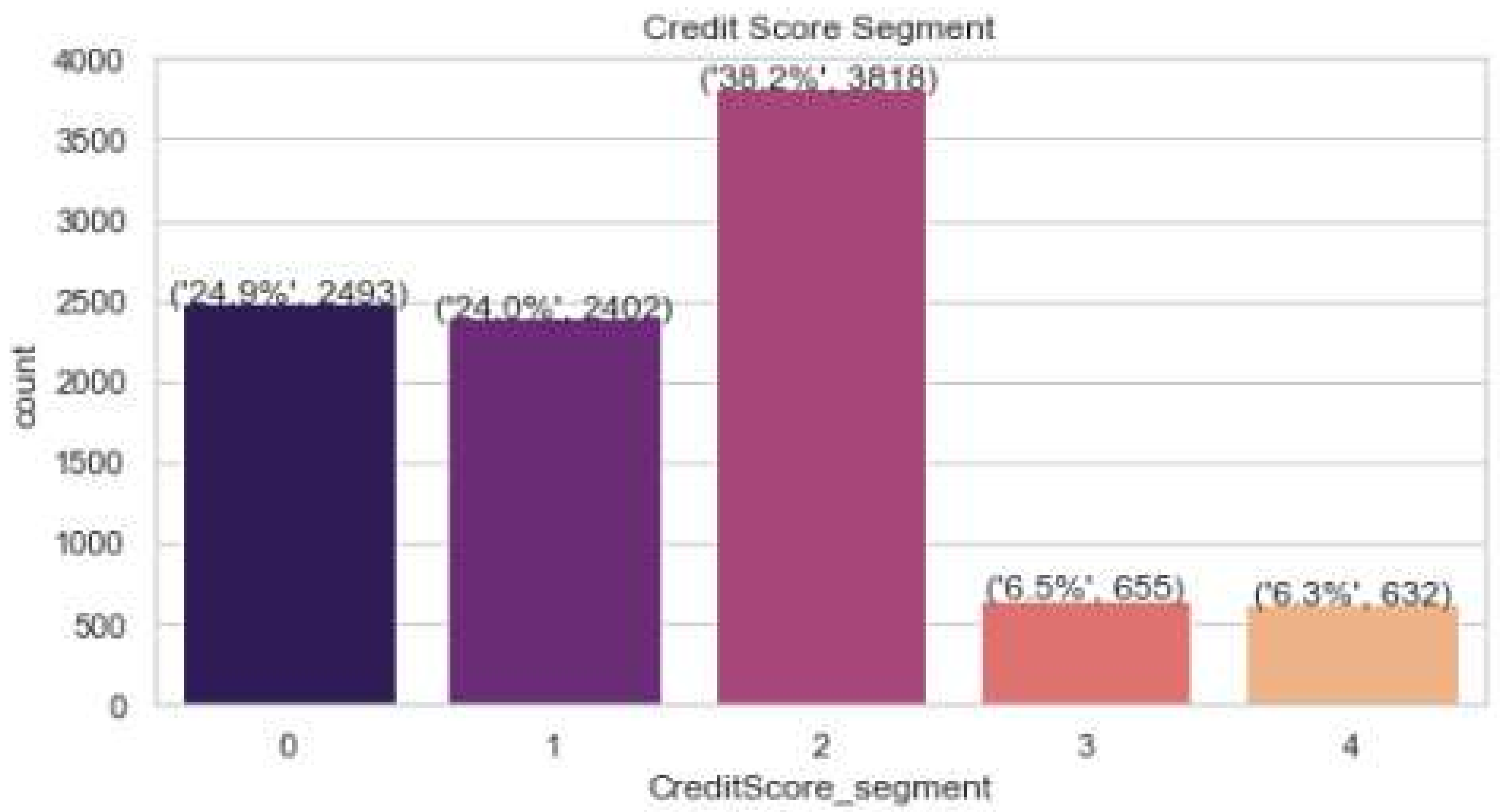
```
plt.figure(figsize=(8,4))
ax = sns.countplot(x = df['Age_segment'],palette="magma")
add_labels(df, 'Age_segment',ax)
plt.title("Age Segment")
plt.show()
```



From the countplot, the distribution of age is segmented around the age group of 30-49 years old.

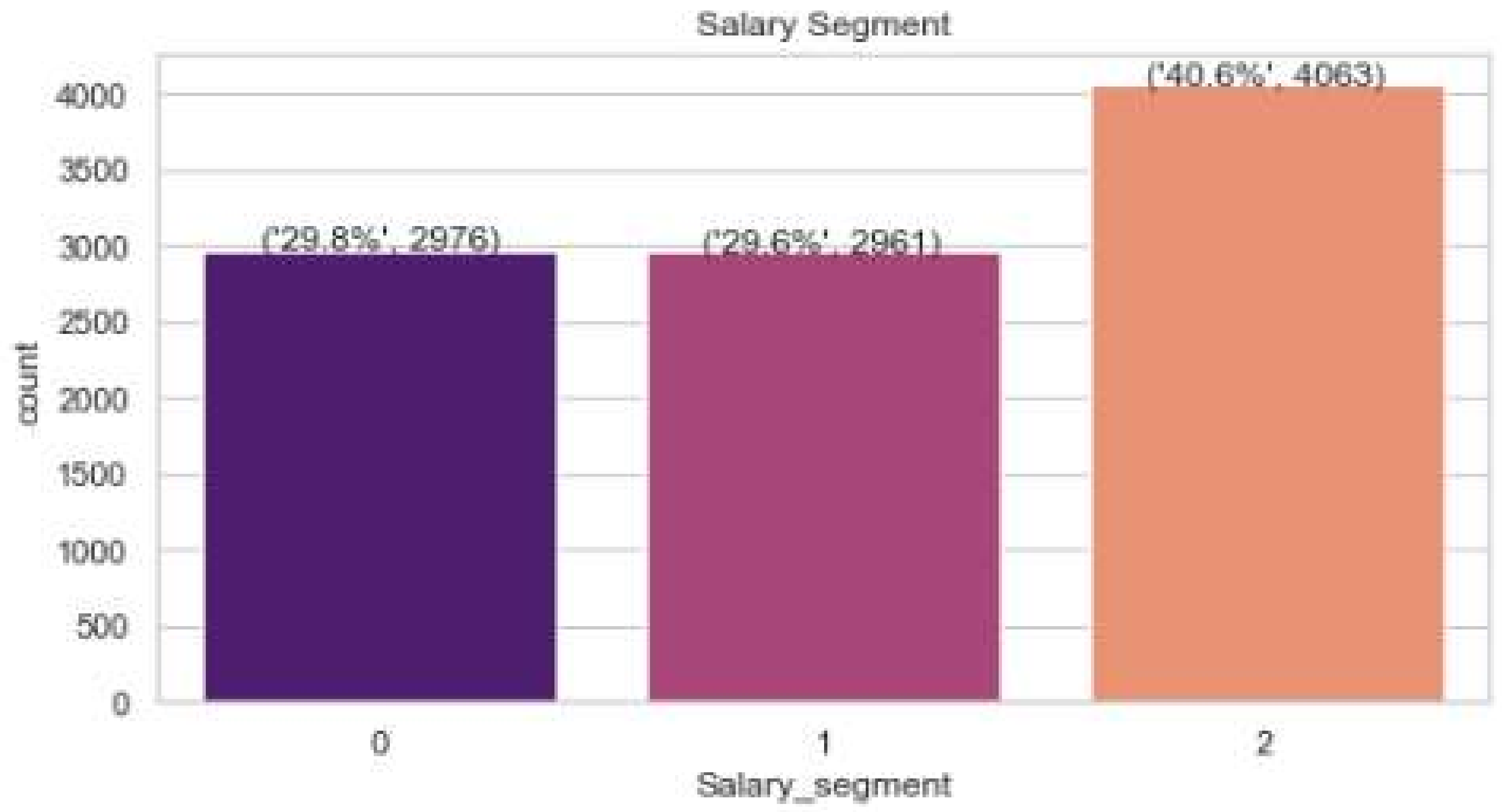


```
plt.figure(figsize=(8,4))
ax = sns.countplot(x = df['CreditScore_segment'],palette="magma")
add_labels(df, 'CreditScore_segment',ax)
plt.title("Credit Score Segment")
plt.show()
```



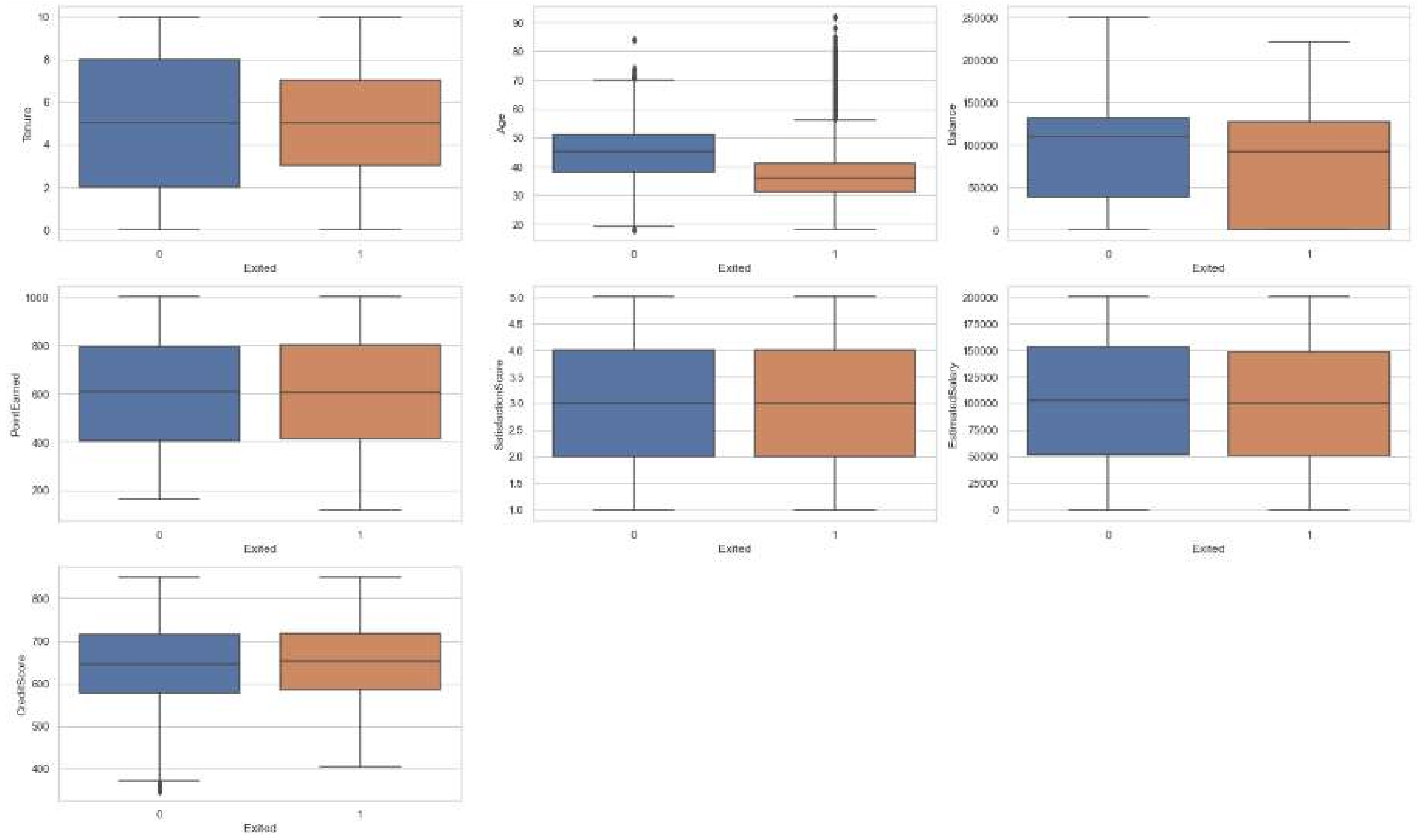
Most customers' credit scores are segmented in the range from Low to High, with the smallest distribution of customers observed at both ends of the spectrum.

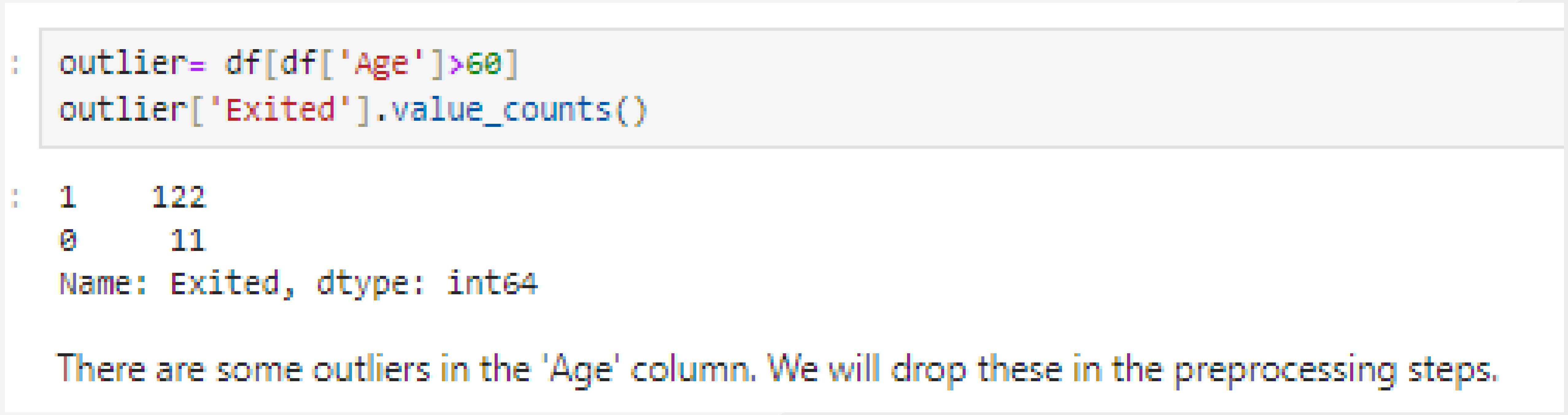
```
plt.figure(figsize=(8,4))
ax = sns.countplot(x = df['Salary_segment'],palette="magma")
add_labels(df, 'Salary_segment',ax)
plt.title("Salary Segment")
plt.show()
```



We can see that the majority of our data points belong to the middle income category, with the remaining data evenly divided between the other two categories.

```python
# Set up the figure and axes
plt.figure(figsize=(20, 12))

# Iterate through each column and create grouped box plots side by side
for i, col in enumerate(numerical_columns, start=1):
    plt.subplot(3, 3, i)  # 2 rows, 3 columns layout for 6 features
    sns.boxplot(x="Exited", y=col, data=df)
    plt.xlabel('Exited')
    plt.ylabel(col)

# Adjust layout and show the plot
plt.tight_layout()
plt.show()
```

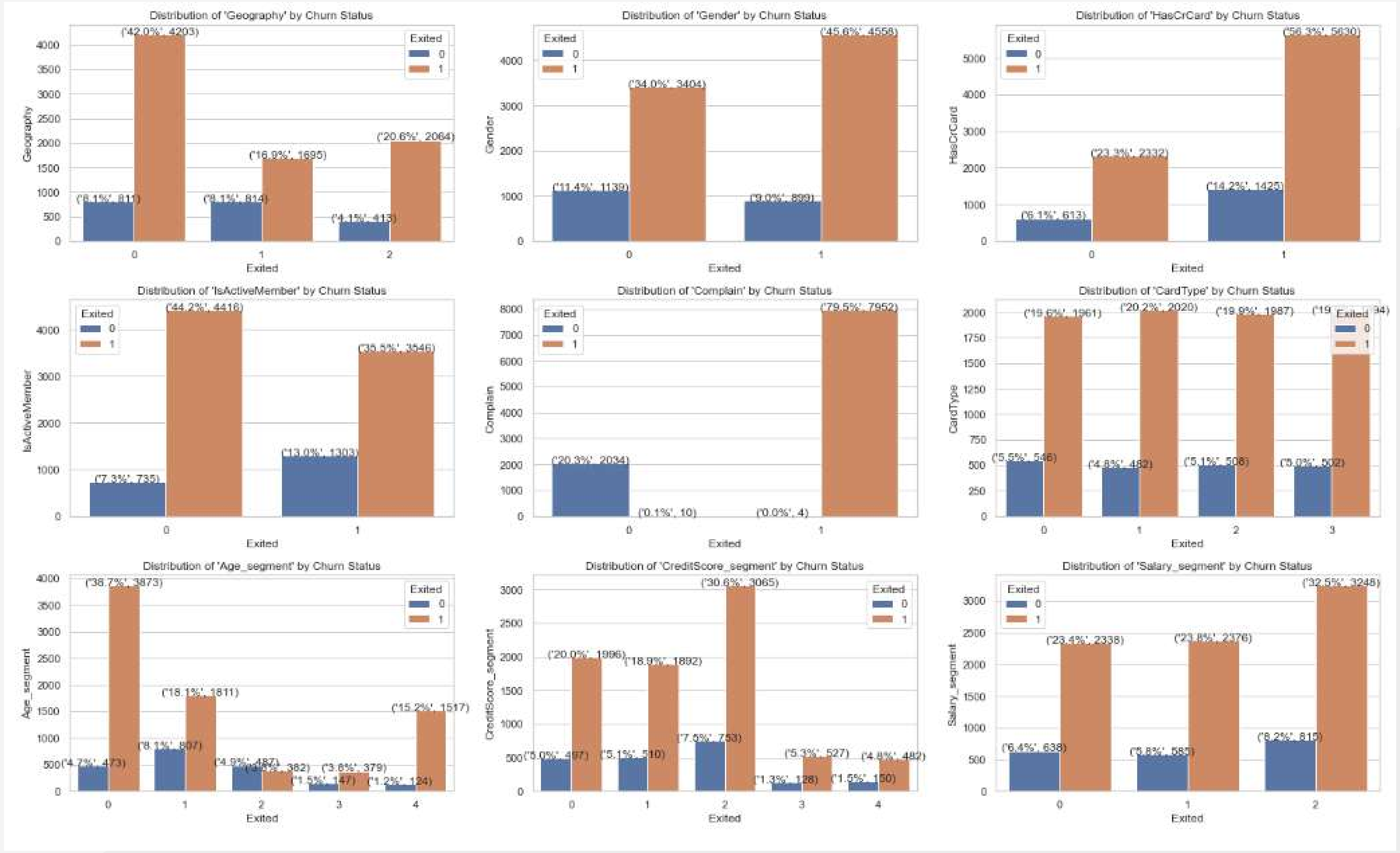




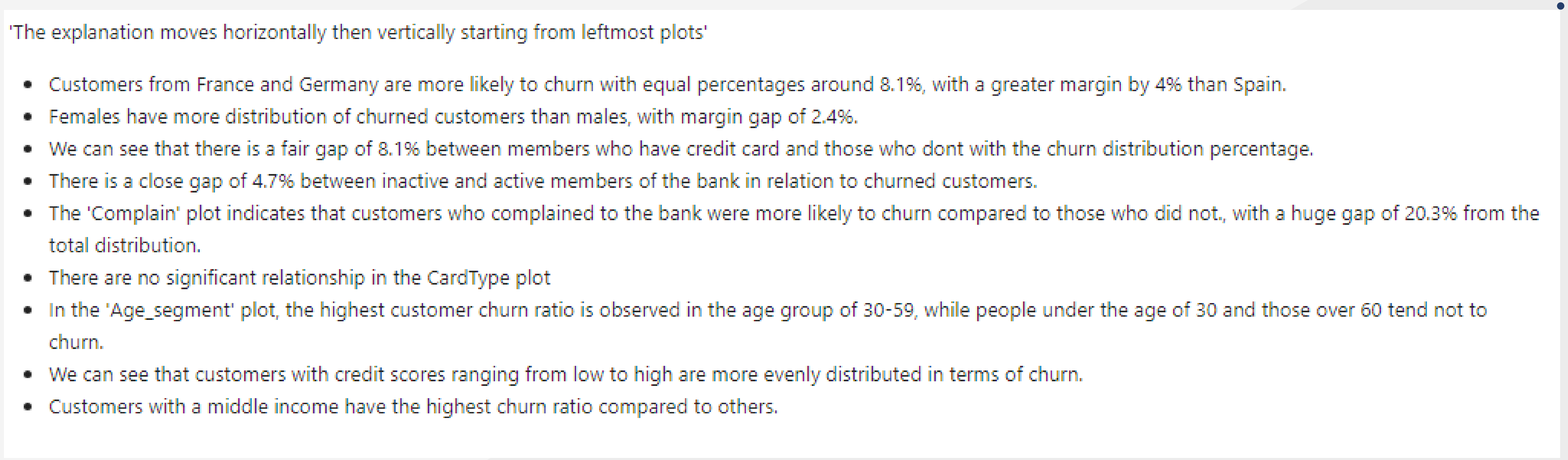
```
outlier= df[df['Age']>60]
outlier['Exited'].value_counts()
```

```
1    122
0     11
Name: Exited, dtype: int64
```

There are some outliers in the 'Age' column. We will drop these in the preprocessing steps.



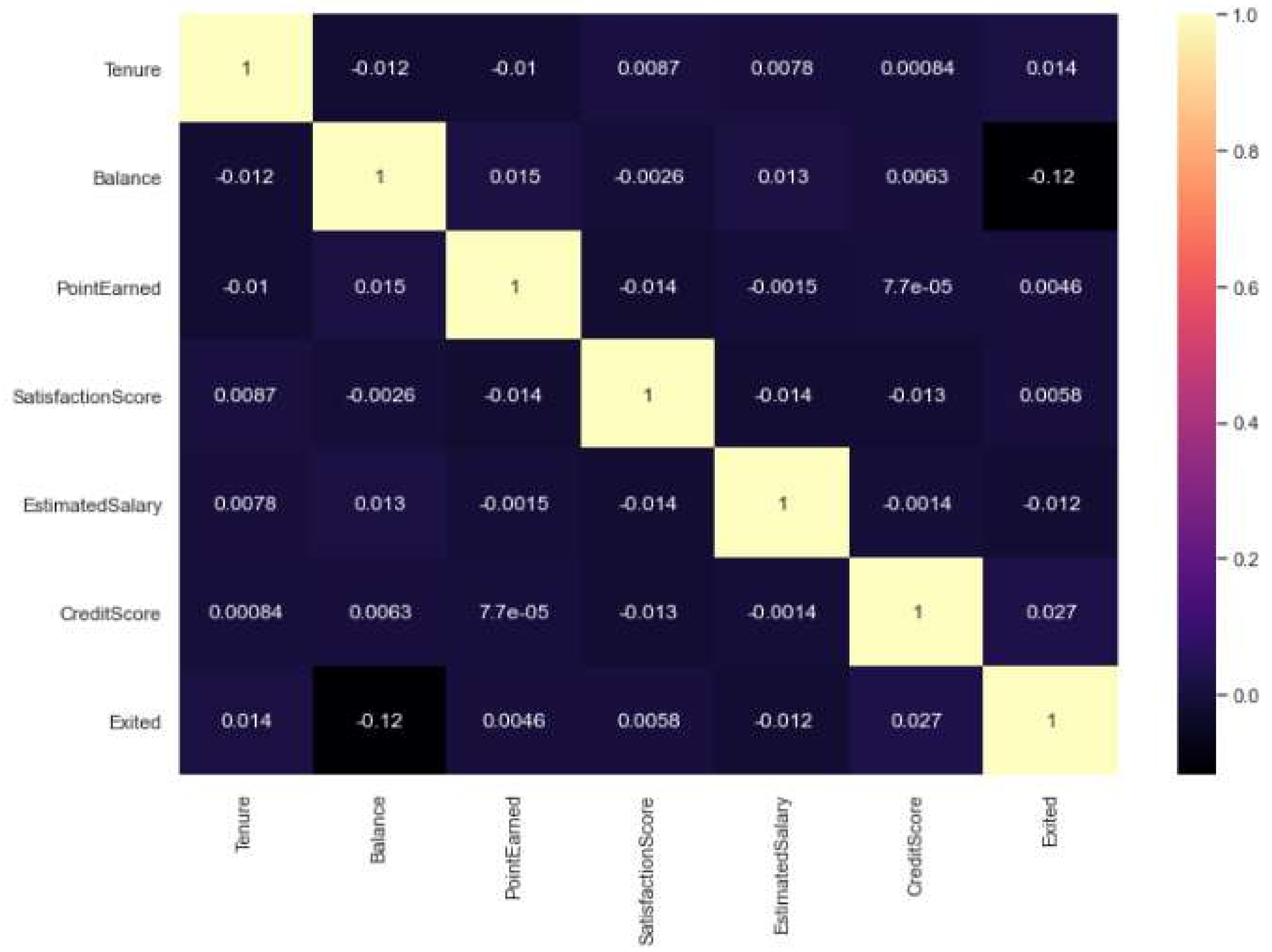
Distribution of 'Geography' by Churn Status
('42.0%', 4203)
('8.1%', 811)
('16.9%', 1695)
('8.1%', 814)
('20.6%', 2064)
('4.1%', 413)
Geography
Exited
Distribution of 'Gender' by Churn Status
('34.0%', 3404)
('11.4%', 1139)
('45.6%', 4558)
('9.0%', 899)
Gender
Exited
Distribution of 'HasCrCard' by Churn Status
('23.3%', 2332)
('6.1%', 613)
('56.3%', 5630)
('14.2%', 1425)
HasCrCard
Exited
Distribution of 'IsActiveMember' by Churn Status
('44.2%', 4416)
('7.3%', 735)
('35.5%', 3546)
('13.0%', 1303)
IsActiveMember
Exited
Distribution of 'Complain' by Churn Status
('20.3%', 2034)
('0.1%', 10)
('0.0%', 4)
('79.5%', 7952)
Complain
Exited
Distribution of 'CardType' by Churn Status
('19.6%', 1961)
('5.5%', 546)
('20.2%', 2020)
('4.8%', 482)
('19.9%', 1987)
('5.1%', 508)
('19
94)
('5.0%', 502)
CardType
Exited
Distribution of 'Age_segment' by Churn Status
('38.7%', 3873)
('4.7%', 473)
('18.1%', 1811)
('8.1%', 807)
('4.9%', 487)
382)
('3.8%', 379)
('1.5%', 147)
('15.2%', 1517)
('1.2%', 124)
Age_segment
Exited
Distribution of 'CreditScore_segment' by Churn Status
('20.0%', 1996)
('5.0%', 497)
('18.9%', 1892)
('5.1%', 510)
('30.6%', 3065)
('7.5%', 753)
('5.3%', 527)
('1.3%', 128)
('4.8%', 482)
('1.5%', 150)
CreditScore_segment
Exited
Distribution of 'Salary_segment' by Churn Status
('23.4%', 2338)
('6.4%', 638)
('23.8%', 2376)
('5.8%', 585)
('32.5%', 3248)
('8.2%', 815)
Salary_segment
Exited
Exited
0
1

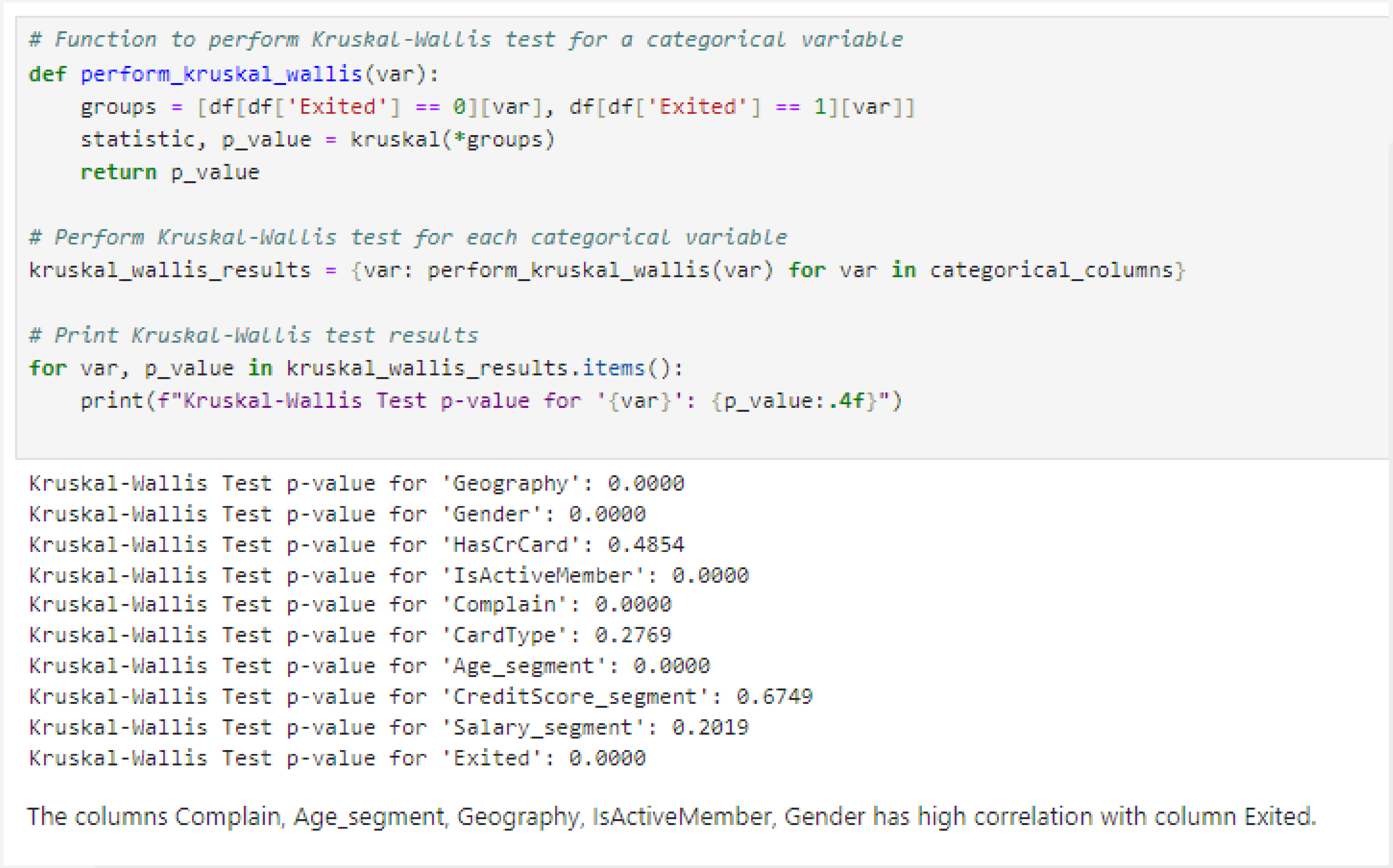

'The explanation moves horizontally then vertically starting from leftmost plots'

- Customers from France and Germany are more likely to churn with equal percentages around 8.1%, with a greater margin by 4% than Spain.
- Females have more distribution of churned customers than males, with margin gap of 2.4%.
- We can see that there is a fair gap of 8.1% between members who have credit card and those who dont with the churn distribution percentage.
- There is a close gap of 4.7% between inactive and active members of the bank in relation to churned customers.
- The 'Complain' plot indicates that customers who complained to the bank were more likely to churn compared to those who did not., with a huge gap of 20.3% from the total distribution.
- There are no significant relationship in the CardType plot
- In the 'Age_segment' plot, the highest customer churn ratio is observed in the age group of 30-59, while people under the age of 30 and those over 60 tend not to churn.
- We can see that customers with credit scores ranging from low to high are more evenly distributed in terms of churn.
- Customers with a middle income have the highest churn ratio compared to others.

```
numerical_columns = ['Tenure', 'Balance', 'PointEarned','SatisfactionScore','EstimatedSalary','CreditScore','Exited']
plt.figure(figsize=(12,8))
numeric_df = df[numerical_columns]
sns.heatmap(numeric_df.corr(), cmap = 'magma', annot = True)
```

```
<AxesSubplot:>
```

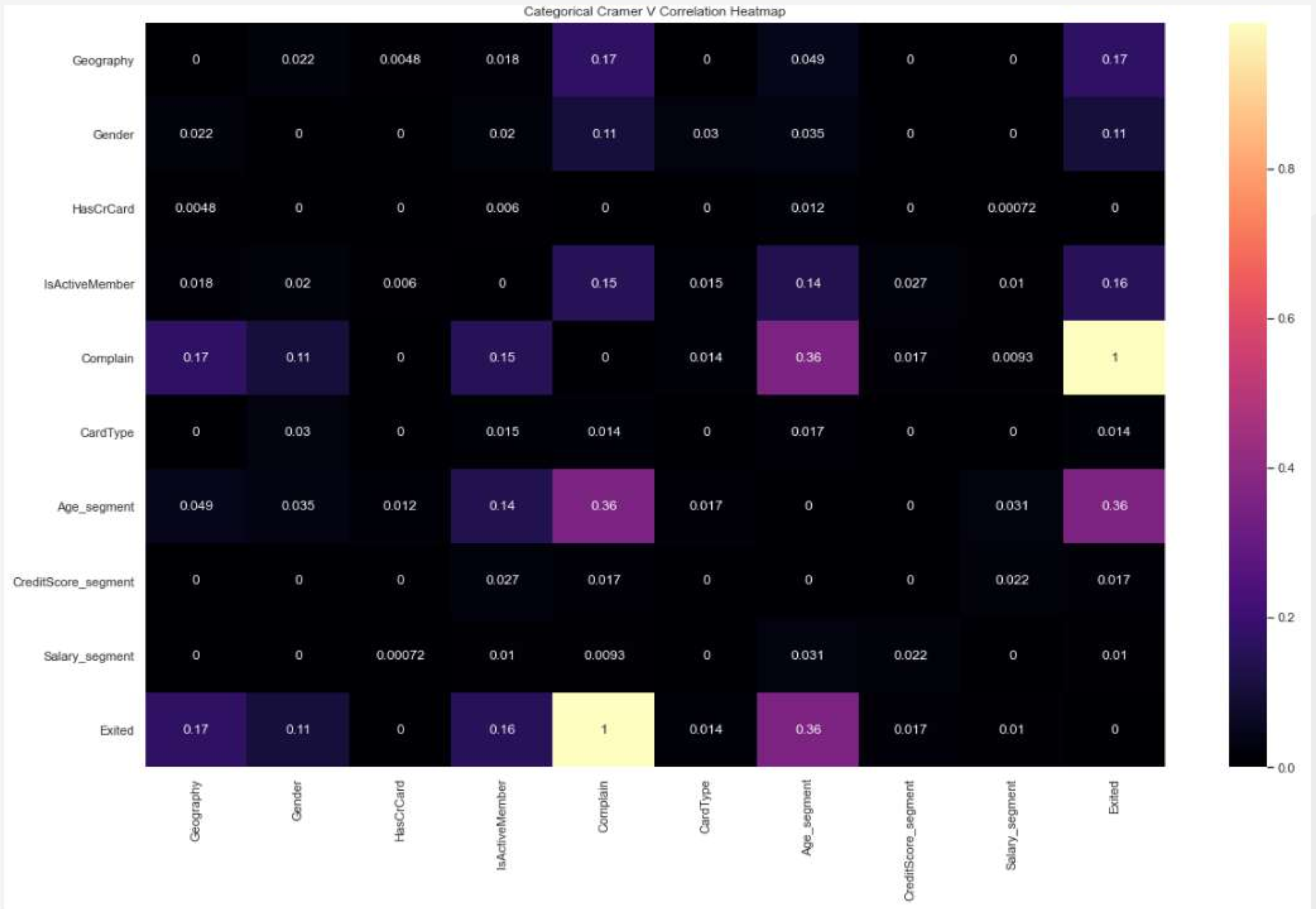




```python
# Function to perform Kruskal-Wallis test for a categorical variable
def perform_kruskal_wallis(var):
    groups = [df[df['Exited'] == 0][var], df[df['Exited'] == 1][var]]
    statistic, p_value = kruskal(*groups)
    return p_value

# Perform Kruskal-Wallis test for each categorical variable
kruskal_wallis_results = {var: perform_kruskal_wallis(var) for var in categorical_columns}

# Print Kruskal-Wallis test results
for var, p_value in kruskal_wallis_results.items():
    print(f"Kruskal-Wallis Test p-value for '{var}': {p_value:.4f}")
```

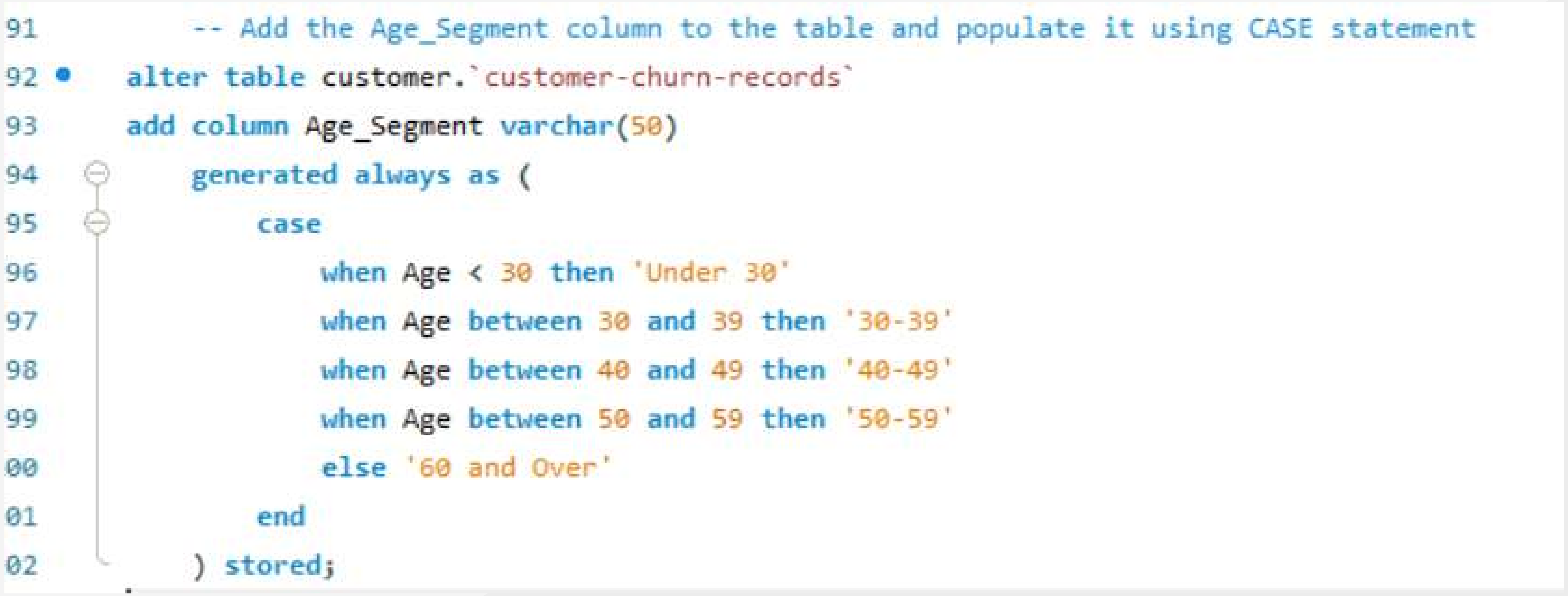
```
Kruskal-Wallis Test p-value for 'Geography': 0.0000
Kruskal-Wallis Test p-value for 'Gender': 0.0000
Kruskal-Wallis Test p-value for 'HasCrCard': 0.4854
Kruskal-Wallis Test p-value for 'IsActiveMember': 0.0000
Kruskal-Wallis Test p-value for 'Complain': 0.0000
Kruskal-Wallis Test p-value for 'CardType': 0.2769
Kruskal-Wallis Test p-value for 'Age_segment': 0.0000
Kruskal-Wallis Test p-value for 'CreditScore_segment': 0.6749
Kruskal-Wallis Test p-value for 'Salary_segment': 0.2019
Kruskal-Wallis Test p-value for 'Exited': 0.0000
```

The columns Complain, Age_segment, Geography, IsActiveMember, Gender has high correlation with column Exited.



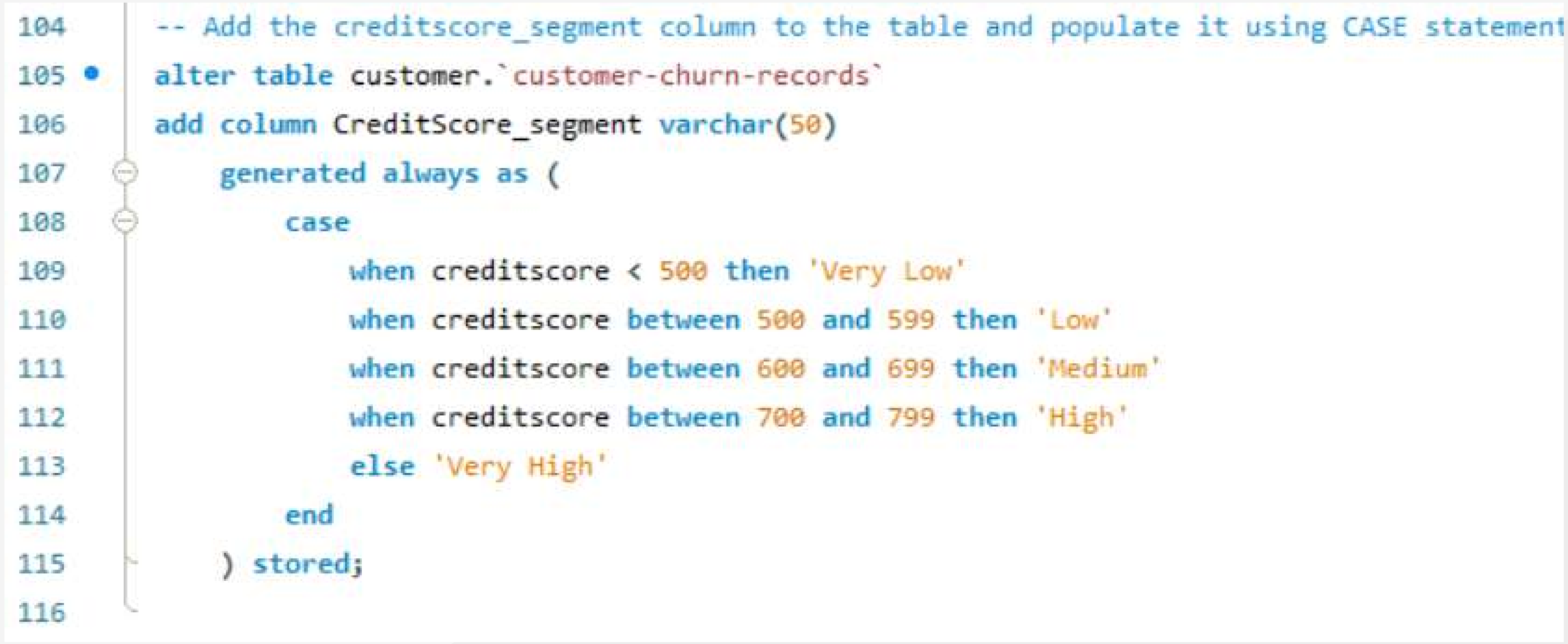
The correlation heatmap suggests that there are two columns that have moderately useful correlation with Customer churn behavior, Complain and Age_segment, with Complain having direct correlation and Age_segment having low correlation.

DATA PREPROCESSING



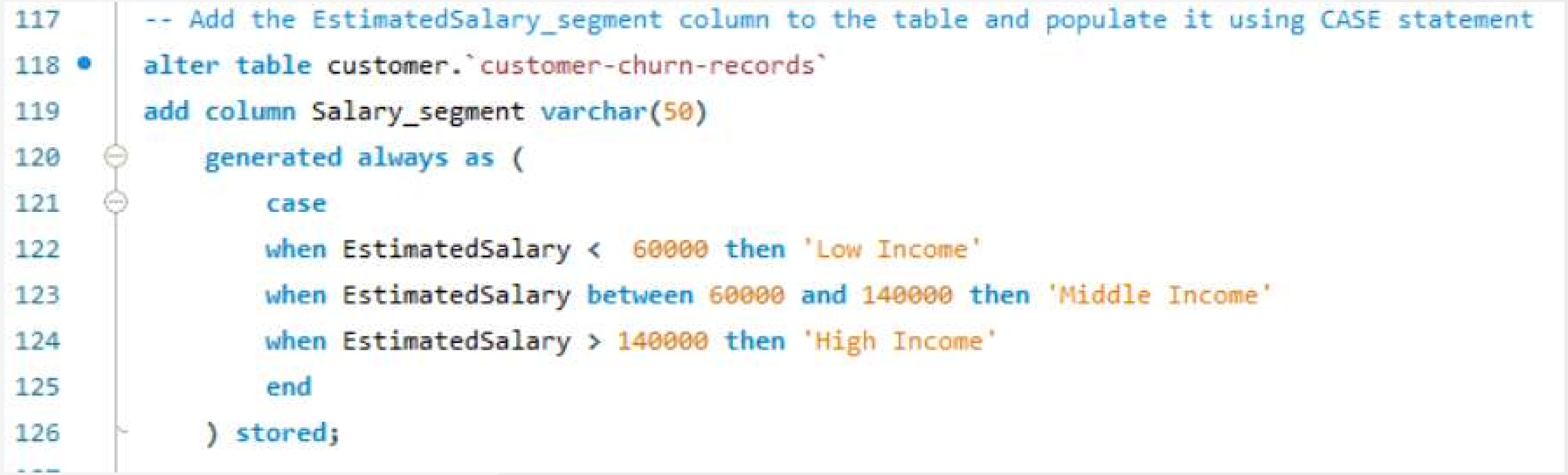
```sql
        -- Add the Age_Segment column to the table and populate it using CASE statement
alter table customer.`customer-churn-records`
add column Age_Segment varchar(50)
    generated always as (
        case
            when Age < 30 then 'Under 30'
            when Age between 30 and 39 then '30-39'
            when Age between 40 and 49 then '40-49'
            when Age between 50 and 59 then '50-59'
            else '60 and Over'
        end
    ) stored;
```

Feature engineer a new feature called Age_segment to segment the age groups to 5 groups (Under 30, 30-39, 40-49, 50-59, and 60 and Over).



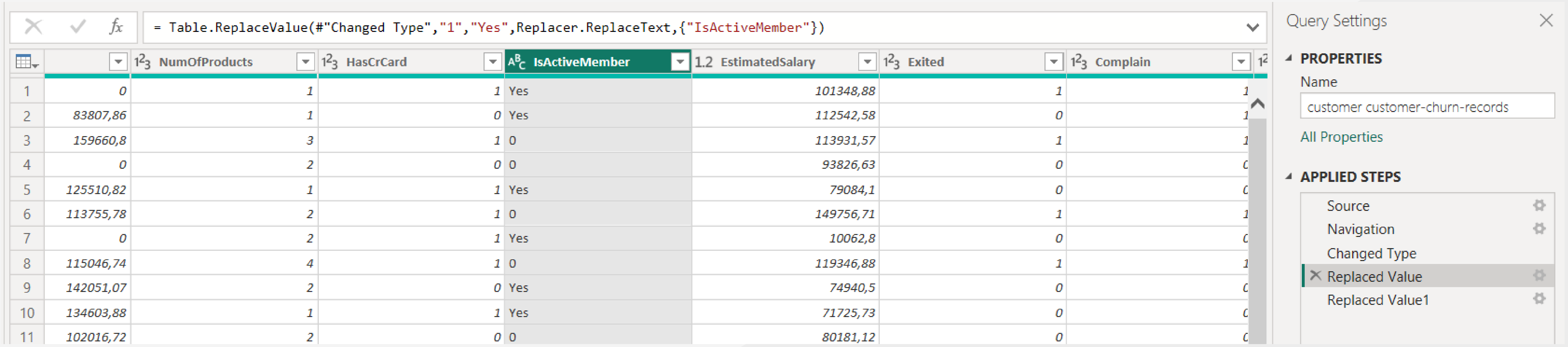
```sql
-- Add the creditscore_segment column to the table and populate it using CASE statement
alter table customer.`customer-churn-records`
add column CreditScore_segment varchar(50)
    generated always as (
        case
            when creditscore < 500 then 'Very Low'
            when creditscore between 500 and 599 then 'Low'
            when creditscore between 600 and 699 then 'Medium'
            when creditscore between 700 and 799 then 'High'
            else 'Very High'
        end
    ) stored;
```

Feature engineer a new feature called Creditscore_segment to segment the CreditScore to group of 5 categories (Very Low, Low, Medium, High, and Very High).



```sql
-- Add the EstimatedSalary_segment column to the table and populate it using CASE statement
alter table customer.`customer-churn-records`
add column Salary_segment varchar(50)
    generated always as (
        case
        when EstimatedSalary <  60000 then 'Low Income'
        when EstimatedSalary between 60000 and 140000 then 'Middle Income'
        when EstimatedSalary > 140000 then 'High Income'
        end
    ) stored;
```

Feature engineered a new feature called Salary_segment to segment the Salary to groups of 3 categories (Low Income, Middle Income, and High Income).



```
= Table.ReplaceValue(#"Changed Type","1","Yes",Replacer.ReplaceText,{"IsActiveMember"})
```

		NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited	Complain
1	0	1	1	Yes	101348,88	1	1
2	83807,86	1	0	Yes	112542,58	0	1
3	159660,8	3	1	0	113931,57	1	1
4	0	2	0	0	93826,63	0	0
5	125510,82	1	1	Yes	79084,1	0	0
6	113755,78	2	1	0	149756,71	1	1
7	0	2	1	Yes	10062,8	0	0
8	115046,74	4	1	0	119346,88	1	1
9	142051,07	2	0	Yes	74940,5	0	0
10	134603,88	1	1	Yes	71725,73	0	0
11	102016,72	2	0	0	80181,12	0	0

Query Settings

PROPERTIES

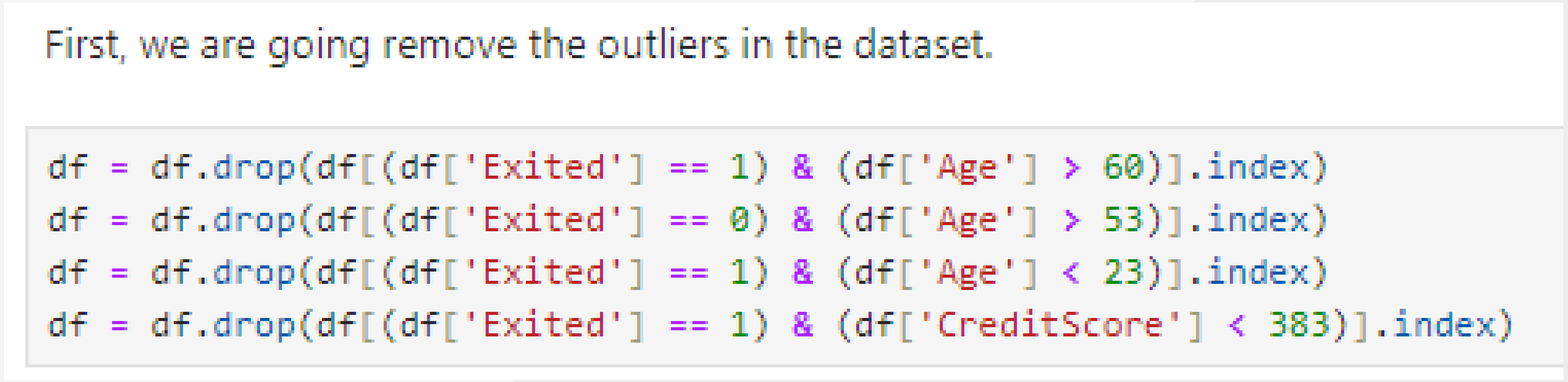
Name: customer customer-churn-records

All Properties

APPLIED STEPS

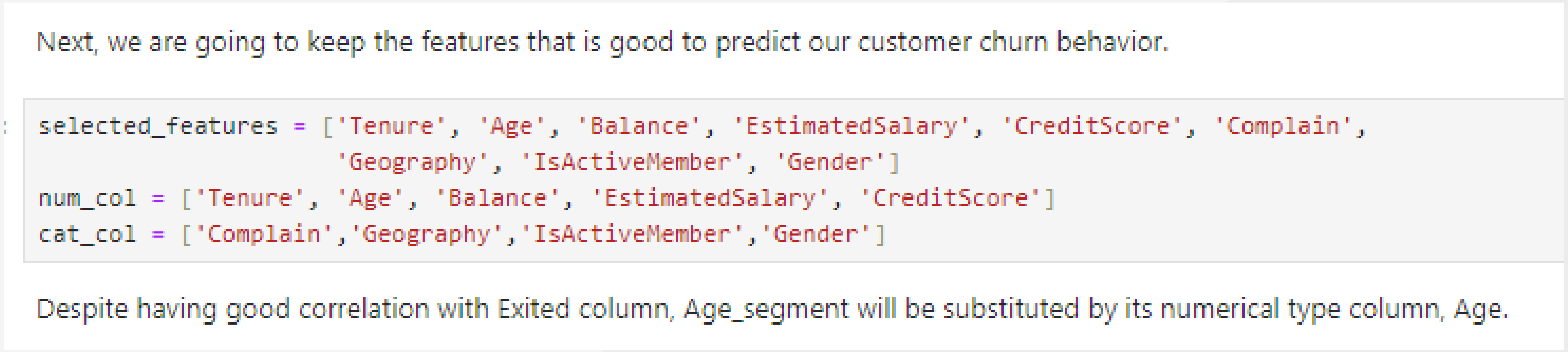
- Source
- Navigation
- Changed Type
- Replaced Value
- Replaced Value1

In Power BI, using the Power Query Transform, i changed the values of column IsActiveMember from 1 and 0s to Yes and No. This was done to create clearer labels for visualization later.



First, we are going remove the outliers in the dataset.

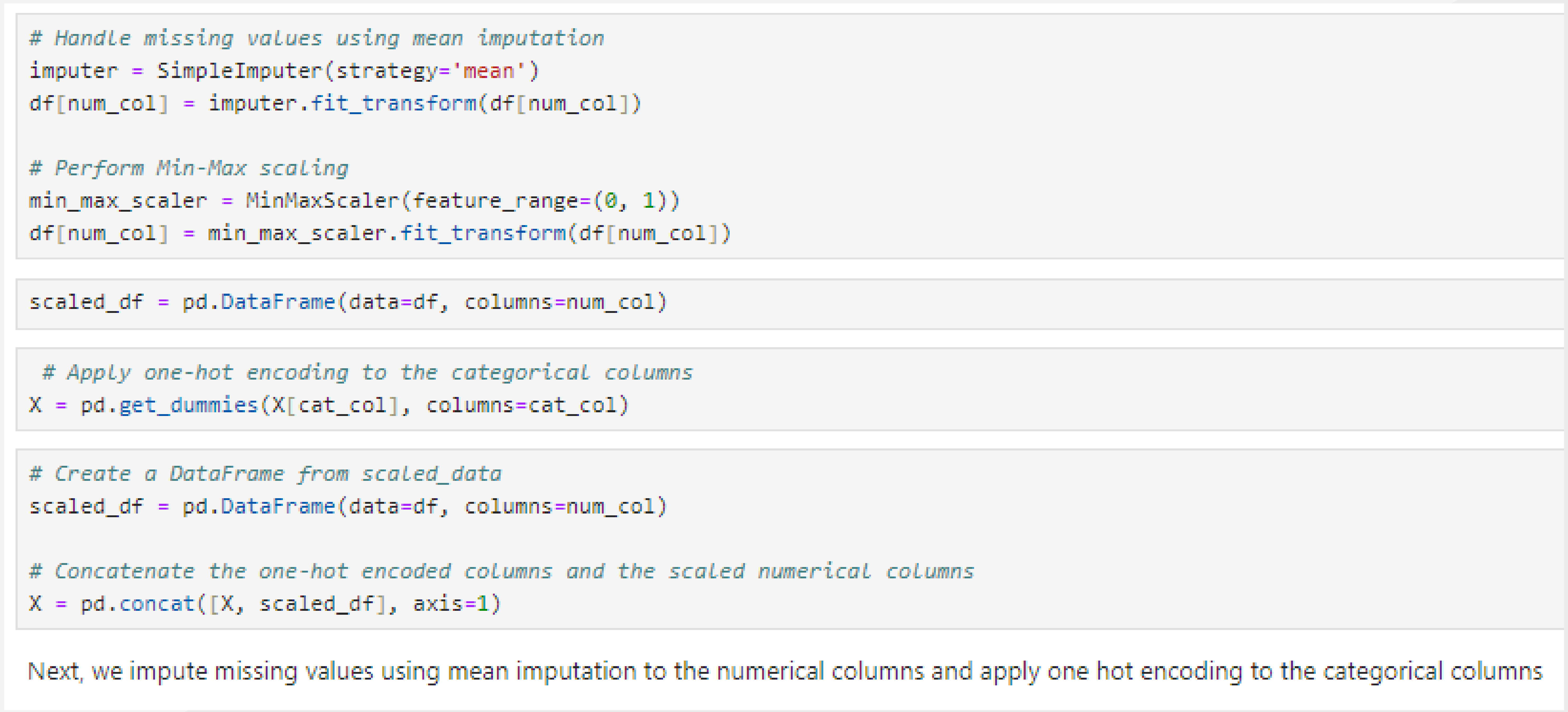
```
df = df.drop(df[(df['Exited'] == 1) & (df['Age'] > 60)].index)
df = df.drop(df[(df['Exited'] == 0) & (df['Age'] > 53)].index)
df = df.drop(df[(df['Exited'] == 1) & (df['Age'] < 23)].index)
df = df.drop(df[(df['Exited'] == 1) & (df['CreditScore'] < 383)].index)
```



Next, we are going to keep the features that is good to predict our customer churn behavior.

```
selected_features = ['Tenure', 'Age', 'Balance', 'EstimatedSalary', 'CreditScore', 'Complain',
                     'Geography', 'IsActiveMember', 'Gender']
num_col = ['Tenure', 'Age', 'Balance', 'EstimatedSalary', 'CreditScore']
cat_col = ['Complain','Geography','IsActiveMember','Gender']
```

Despite having good correlation with Exited column, Age_segment will be substituted by its numerical type column, Age.



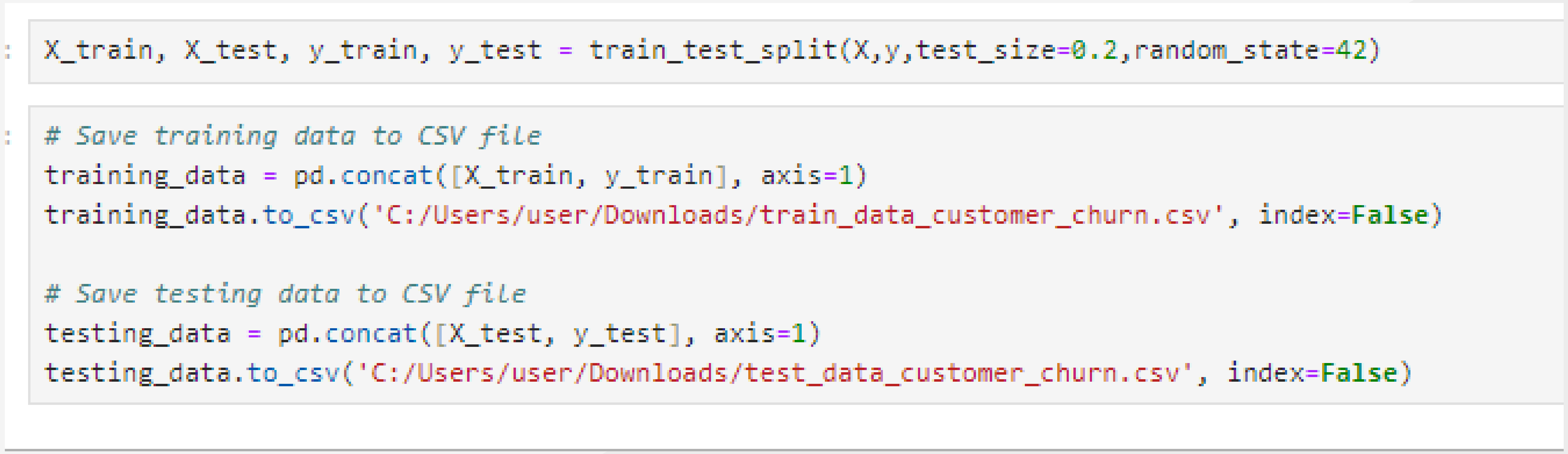
```
# Handle missing values using mean imputation
imputer = SimpleImputer(strategy='mean')
df[num_col] = imputer.fit_transform(df[num_col])

# Perform Min-Max scaling
min_max_scaler = MinMaxScaler(feature_range=(0, 1))
df[num_col] = min_max_scaler.fit_transform(df[num_col])
```

```
scaled_df = pd.DataFrame(data=df, columns=num_col)
```

```
 # Apply one-hot encoding to the categorical columns
X = pd.get_dummies(X[cat_col], columns=cat_col)
```

```
# Create a DataFrame from scaled_data
scaled_df = pd.DataFrame(data=df, columns=num_col)

# Concatenate the one-hot encoded columns and the scaled numerical columns
X = pd.concat([X, scaled_df], axis=1)
```

Next, we impute missing values using mean imputation to the numerical columns and apply one hot encoding to the categorical columns

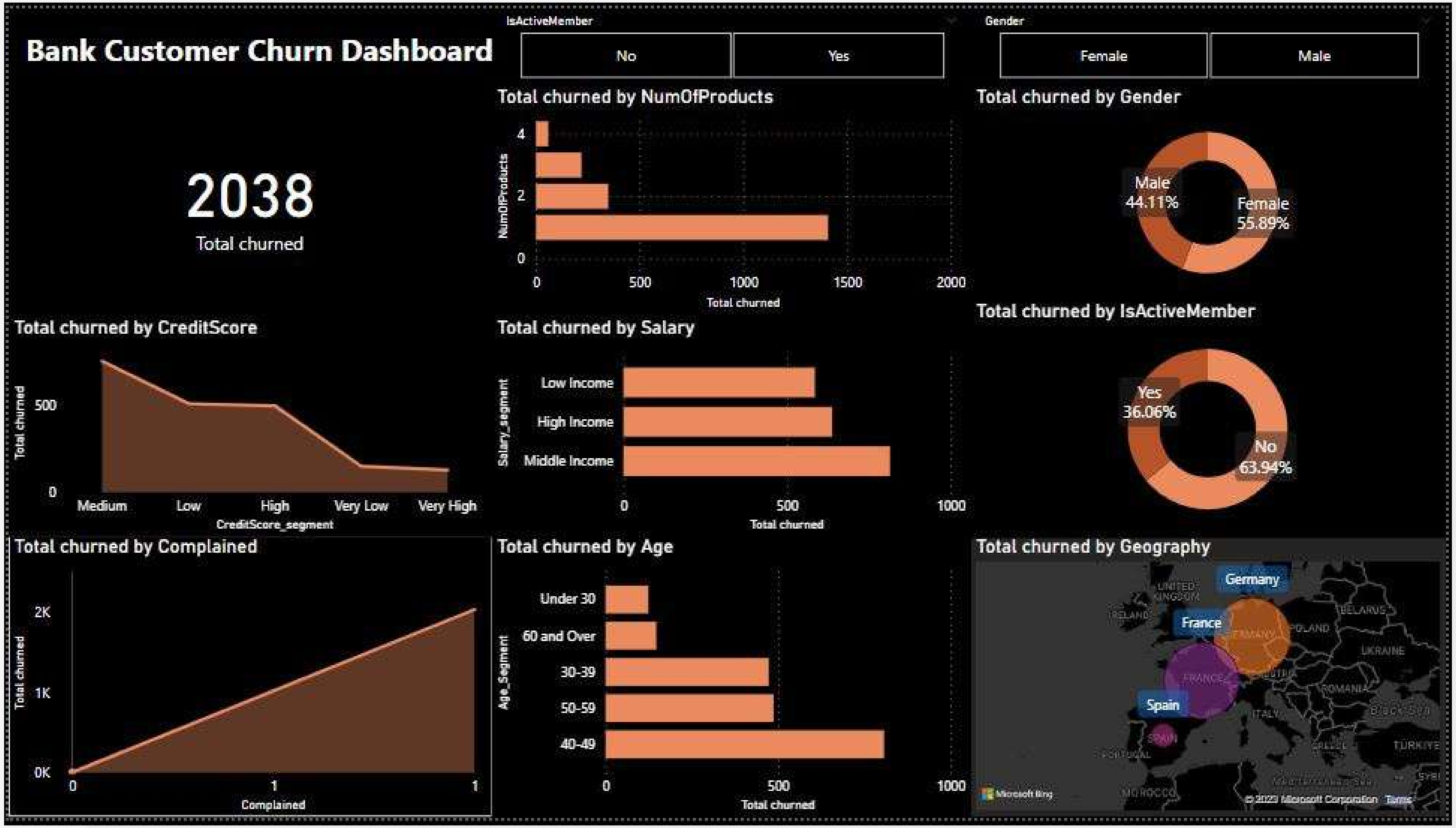


```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,random_state=42)
```

```
# Save training data to CSV file
training_data = pd.concat([X_train, y_train], axis=1)
training_data.to_csv('C:/Users/user/Downloads/train_data_customer_churn.csv', index=False)

# Save testing data to CSV file
testing_data = pd.concat([X_test, y_test], axis=1)
testing_data.to_csv('C:/Users/user/Downloads/test_data_customer_churn.csv', index=False)
```

Then, we split the data to 80 and 20 ratio, and export to separate csv files.

DATA VISUALIZATION

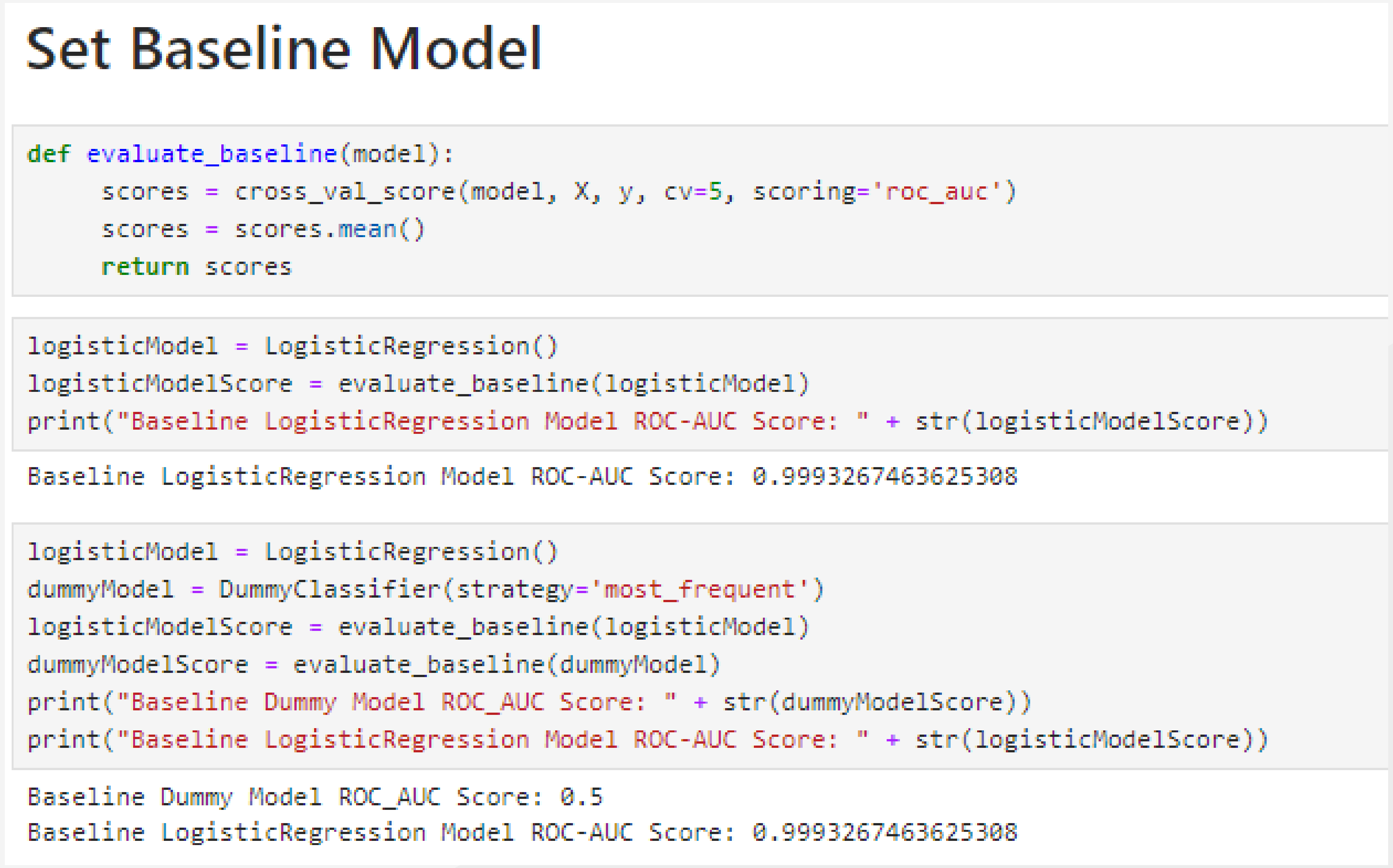


This dashboard was created using Microsoft Power BI and Power Query Transform. Feel free to slice the data using the slicers provided. The purpose of the two slicers were to avoid visualization misunderstandings.

KEY FINDINGS & INSIGHTS

1. Customers with **very high** and **very low credit score** tend **not** to churn.
2. Customers who **rarely complain** about bank products tend **not** to churn.
3. Customers who subscribes to **more bank products** tend **not** to churn.
4. Customers with **Low** and **High Income** tend **not** to churn.
5. **The youngest** and **oldest group** of customers tend **not** to churn.
6. **Males** are **less likely** to churn than females.
7. **Active members** are **less likely** to churn than non-active members.
8. Customers from **Spain** are **less likely** to churn than customers from Germany or France.

MODELING & EVALUATION



Set Baseline Model

```
def evaluate_baseline(model):
    scores = cross_val_score(model, X, y, cv=5, scoring='roc_auc')
    scores = scores.mean()
    return scores
```

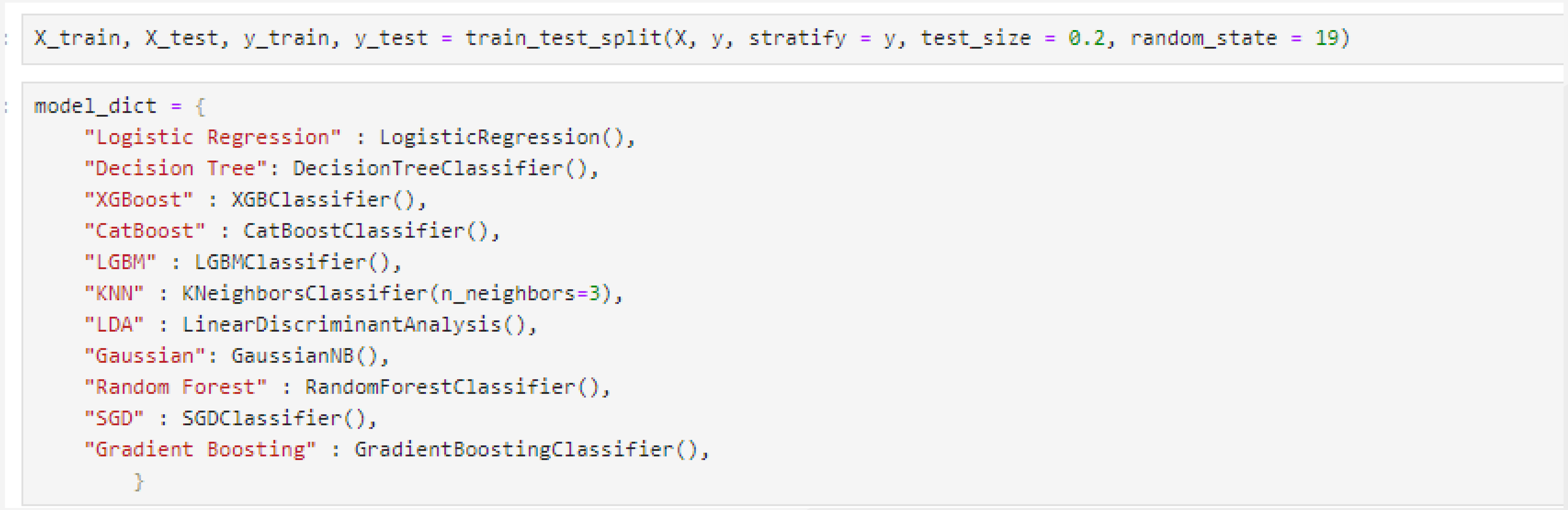
```
logisticModel = LogisticRegression()
logisticModelScore = evaluate_baseline(logisticModel)
print("Baseline LogisticRegression Model ROC-AUC Score: " + str(logisticModelScore))
```

```
Baseline LogisticRegression Model ROC-AUC Score: 0.9993267463625308
```

```
logisticModel = LogisticRegression()
dummyModel = DummyClassifier(strategy='most_frequent')
logisticModelScore = evaluate_baseline(logisticModel)
dummyModelScore = evaluate_baseline(dummyModel)
print("Baseline Dummy Model ROC_AUC Score: " + str(dummyModelScore))
print("Baseline LogisticRegression Model ROC-AUC Score: " + str(logisticModelScore))
```

```
Baseline Dummy Model ROC_AUC Score: 0.5
Baseline LogisticRegression Model ROC-AUC Score: 0.9993267463625308
```

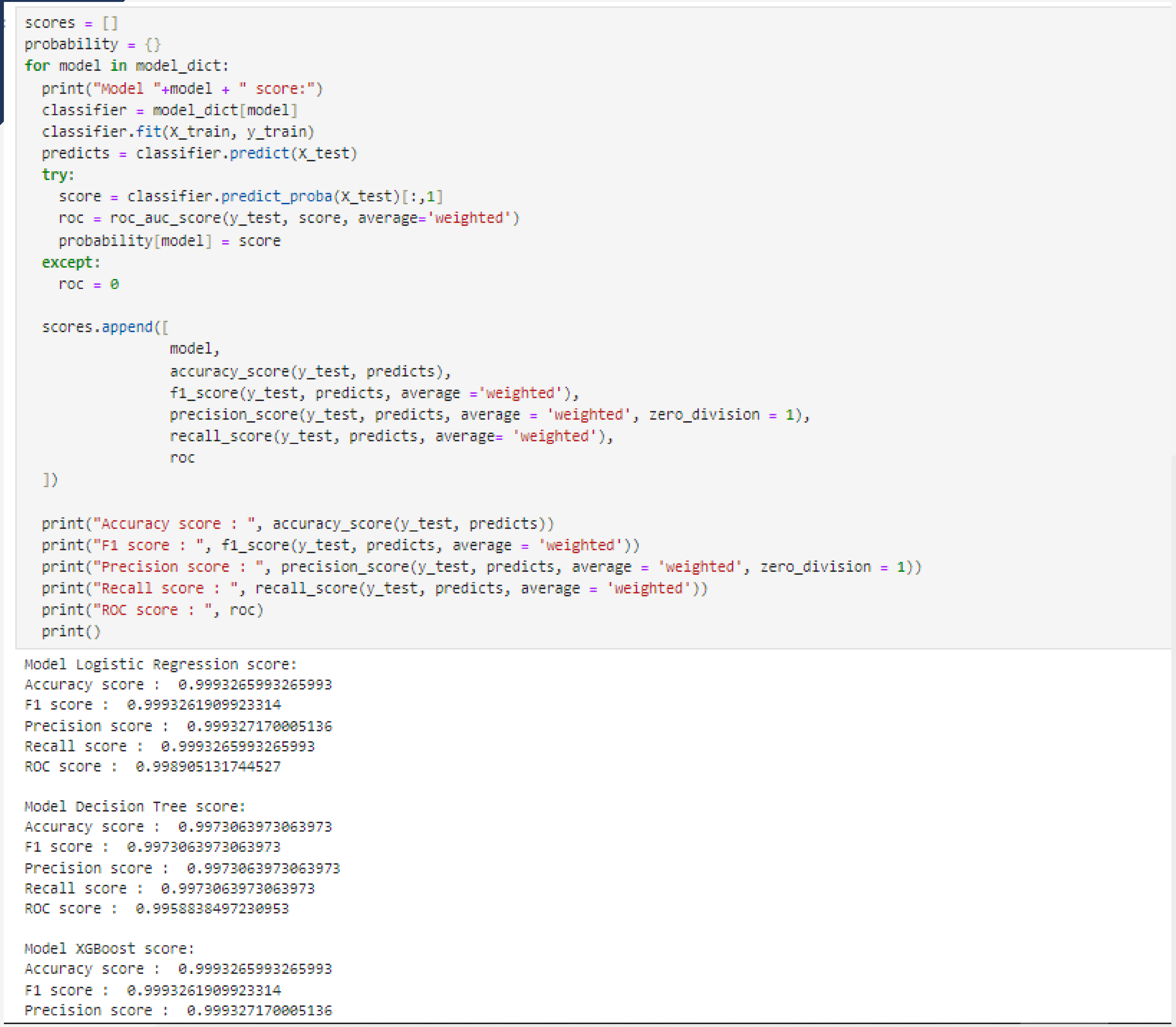
First step, we set a baseline model as comparison for our better model. The ROC_AUC score predicts with 0.99 score.



```
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify = y, test_size = 0.2, random_state = 19)
```

```
model_dict = {
    "Logistic Regression" : LogisticRegression(),
    "Decision Tree": DecisionTreeClassifier(),
    "XGBoost" : XGBClassifier(),
    "CatBoost" : CatBoostClassifier(),
    "LGBM" : LGBMClassifier(),
    "KNN" : KNeighborsClassifier(n_neighbors=3),
    "LDA" : LinearDiscriminantAnalysis(),
    "Gaussian": GaussianNB(),
    "Random Forest" : RandomForestClassifier(),
    "SGD" : SGDClassifier(),
    "Gradient Boosting" : GradientBoostingClassifier(),
        }
```

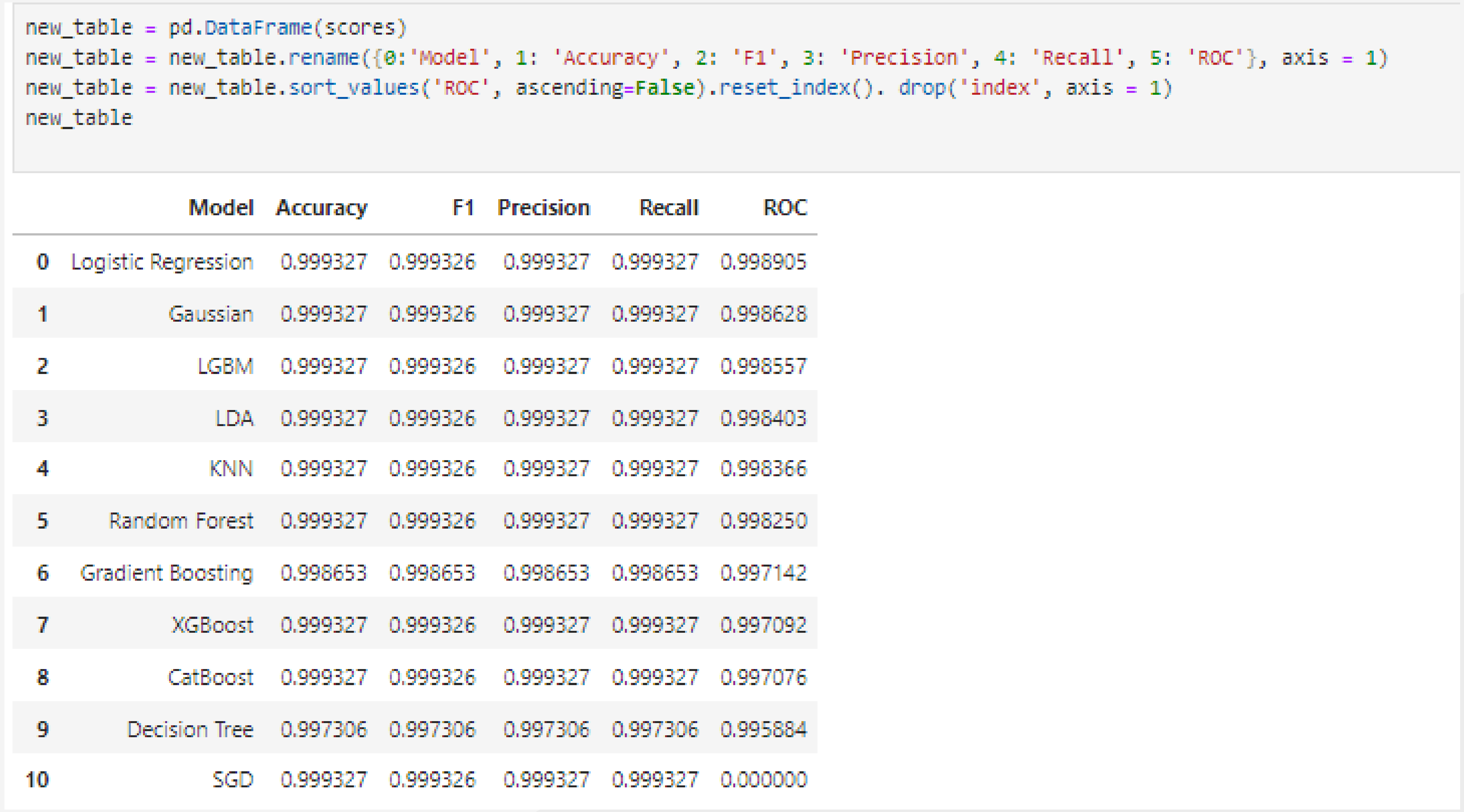
Next, we split the training data, and prepare the model list that we are going to fit to our train data.



```python
scores = []
probability = {}
for model in model_dict:
  print("Model "+model + " score:")
  classifier = model_dict[model]
  classifier.fit(X_train, y_train)
  predicts = classifier.predict(X_test)
  try:
    score = classifier.predict_proba(X_test)[:,1]
    roc = roc_auc_score(y_test, score, average='weighted')
    probability[model] = score
  except:
    roc = 0

  scores.append([
               model,
               accuracy_score(y_test, predicts),
               f1_score(y_test, predicts, average ='weighted'),
               precision_score(y_test, predicts, average = 'weighted', zero_division = 1),
               recall_score(y_test, predicts, average= 'weighted'),
               roc
  ])

  print("Accuracy score : ", accuracy_score(y_test, predicts))
  print("F1 score : ", f1_score(y_test, predicts, average = 'weighted'))
  print("Precision score : ", precision_score(y_test, predicts, average = 'weighted', zero_division = 1))
  print("Recall score : ", recall_score(y_test, predicts, average = 'weighted'))
  print("ROC score : ", roc)
  print()
```

```
Model Logistic Regression score:
Accuracy score :  0.9993265993265993
F1 score :  0.9993261909923314
Precision score :  0.999327170005136
Recall score :  0.9993265993265993
ROC score :  0.998905131744527

Model Decision Tree score:
Accuracy score :  0.9973063973063973
F1 score :  0.9973063973063973
Precision score :  0.9973063973063973
Recall score :  0.9973063973063973
ROC score :  0.9958838497230953

Model XGBoost score:
Accuracy score :  0.9993265993265993
F1 score :  0.9993261909923314
Precision score :  0.999327170005136
```

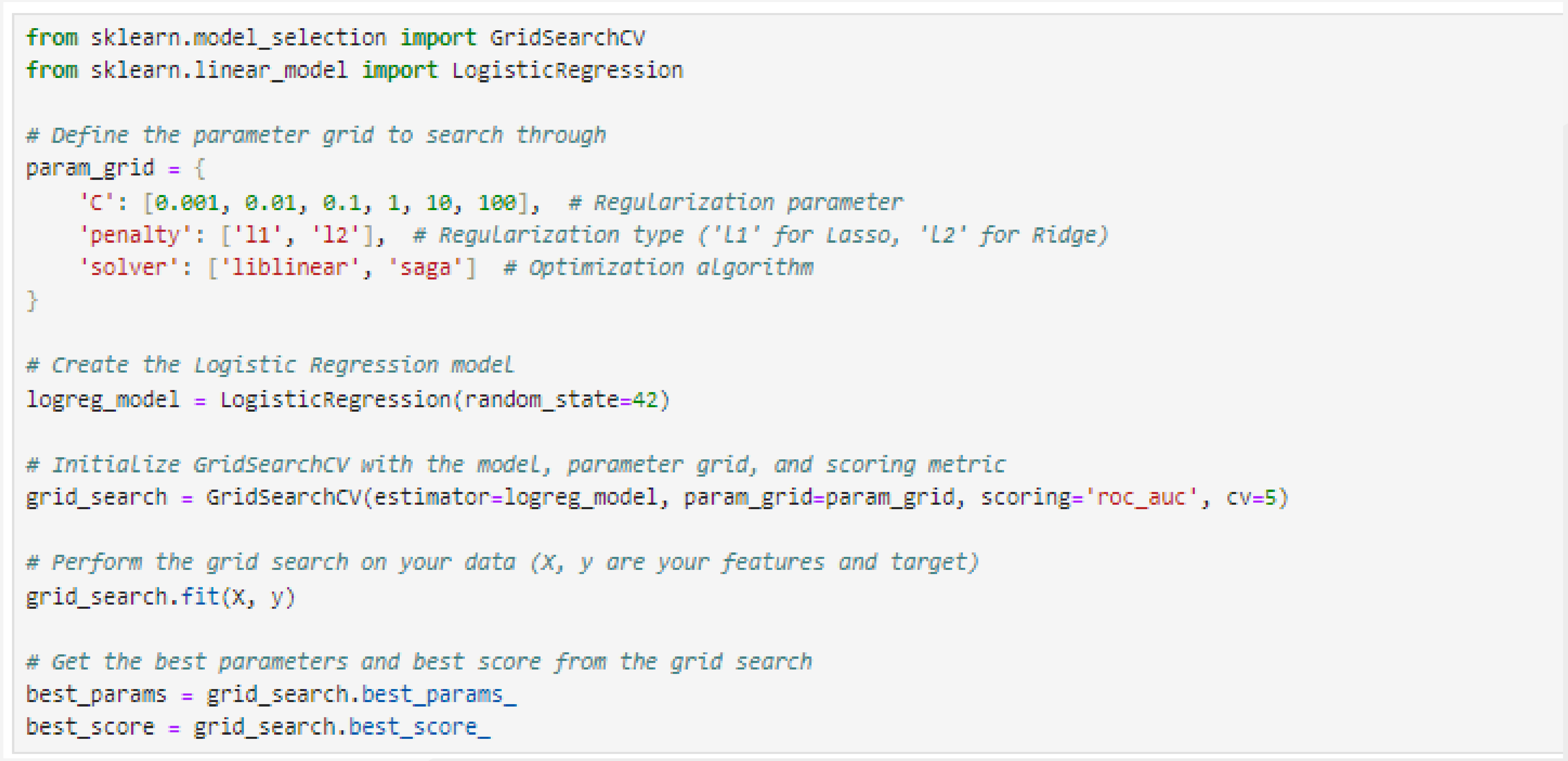
Then, we are going to fit each models and calculate the metrics.



```
new_table = pd.DataFrame(scores)
new_table = new_table.rename({0:'Model', 1: 'Accuracy', 2: 'F1', 3: 'Precision', 4: 'Recall', 5: 'ROC'}, axis = 1)
new_table = new_table.sort_values('ROC', ascending=False).reset_index(). drop('index', axis = 1)
new_table
```

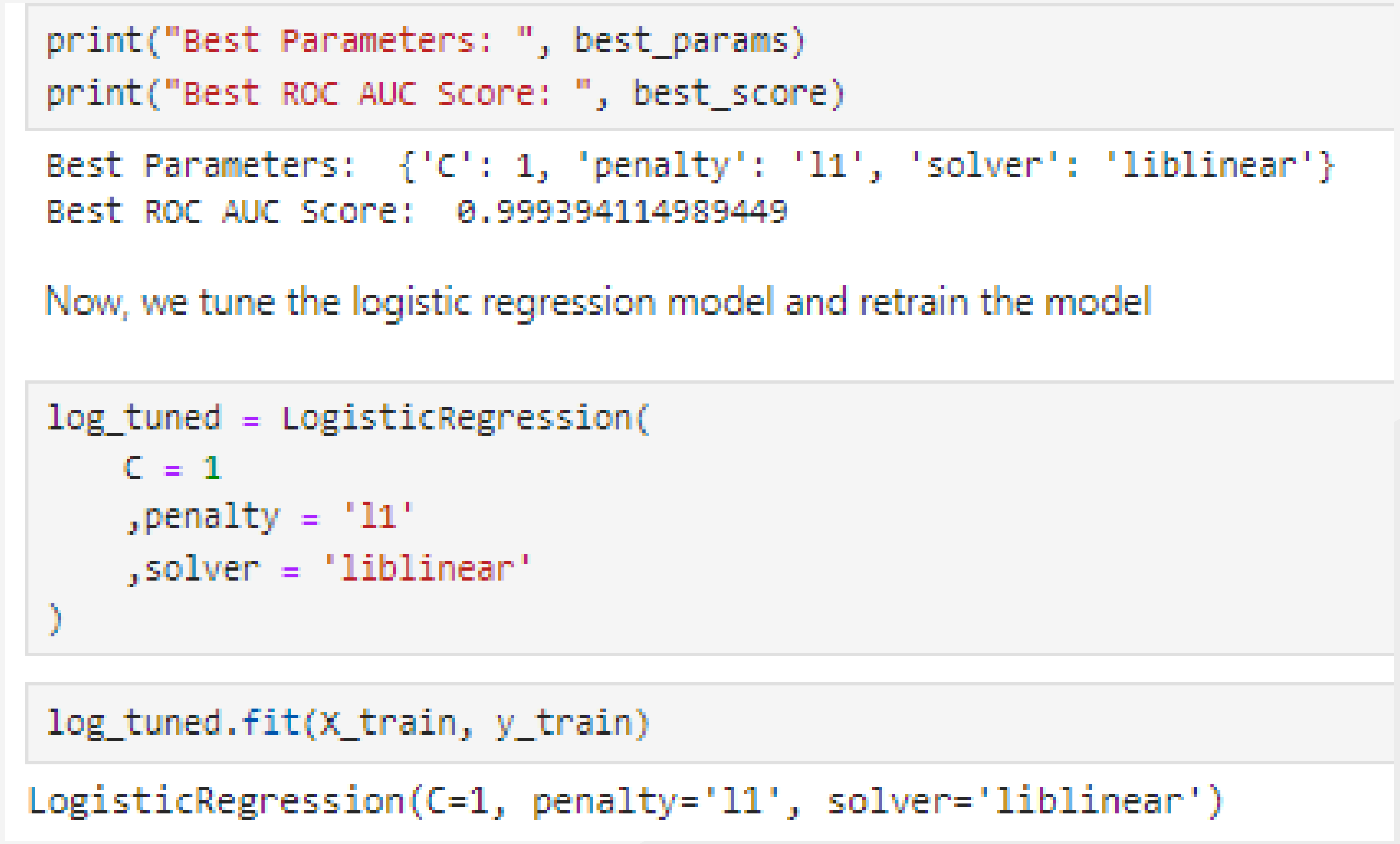
	Model	Accuracy	F1	Precision	Recall	ROC
0	Logistic Regression	0.999327	0.999326	0.999327	0.999327	0.998905
1	Gaussian	0.999327	0.999326	0.999327	0.999327	0.998628
2	LGBM	0.999327	0.999326	0.999327	0.999327	0.998557
3	LDA	0.999327	0.999326	0.999327	0.999327	0.998403
4	KNN	0.999327	0.999326	0.999327	0.999327	0.998366
5	Random Forest	0.999327	0.999326	0.999327	0.999327	0.998250
6	Gradient Boosting	0.998653	0.998653	0.998653	0.998653	0.997142
7	XGBoost	0.999327	0.999326	0.999327	0.999327	0.997092
8	CatBoost	0.999327	0.999326	0.999327	0.999327	0.997076
9	Decision Tree	0.997306	0.997306	0.997306	0.997306	0.995884
10	SGD	0.999327	0.999326	0.999327	0.999327	0.000000

We can see that the highest ROC score is achieved by the Logistic Regression model. We will continue with this model.



```python
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression

# Define the parameter grid to search through
param_grid = {
    'C': [0.001, 0.01, 0.1, 1, 10, 100],  # Regularization parameter
    'penalty': ['l1', 'l2'],  # Regularization type ('l1' for Lasso, 'l2' for Ridge)
    'solver': ['liblinear', 'saga']  # Optimization algorithm
}

# Create the Logistic Regression model
logreg_model = LogisticRegression(random_state=42)

# Initialize GridSearchCV with the model, parameter grid, and scoring metric
grid_search = GridSearchCV(estimator=logreg_model, param_grid=param_grid, scoring='roc_auc', cv=5)

# Perform the grid search on your data (X, y are your features and target)
grid_search.fit(X, y)

# Get the best parameters and best score from the grid search
best_params = grid_search.best_params_
best_score = grid_search.best_score_
```

Next, we would tune the parameters using gridSearch method to gain better hyperparameters in order to achieve better ROC score from our model.



```
print("Best Parameters: ", best_params)
print("Best ROC AUC Score: ", best_score)
```

```
Best Parameters:  {'C': 1, 'penalty': 'l1', 'solver': 'liblinear'}
Best ROC AUC Score:  0.999394114989449
```

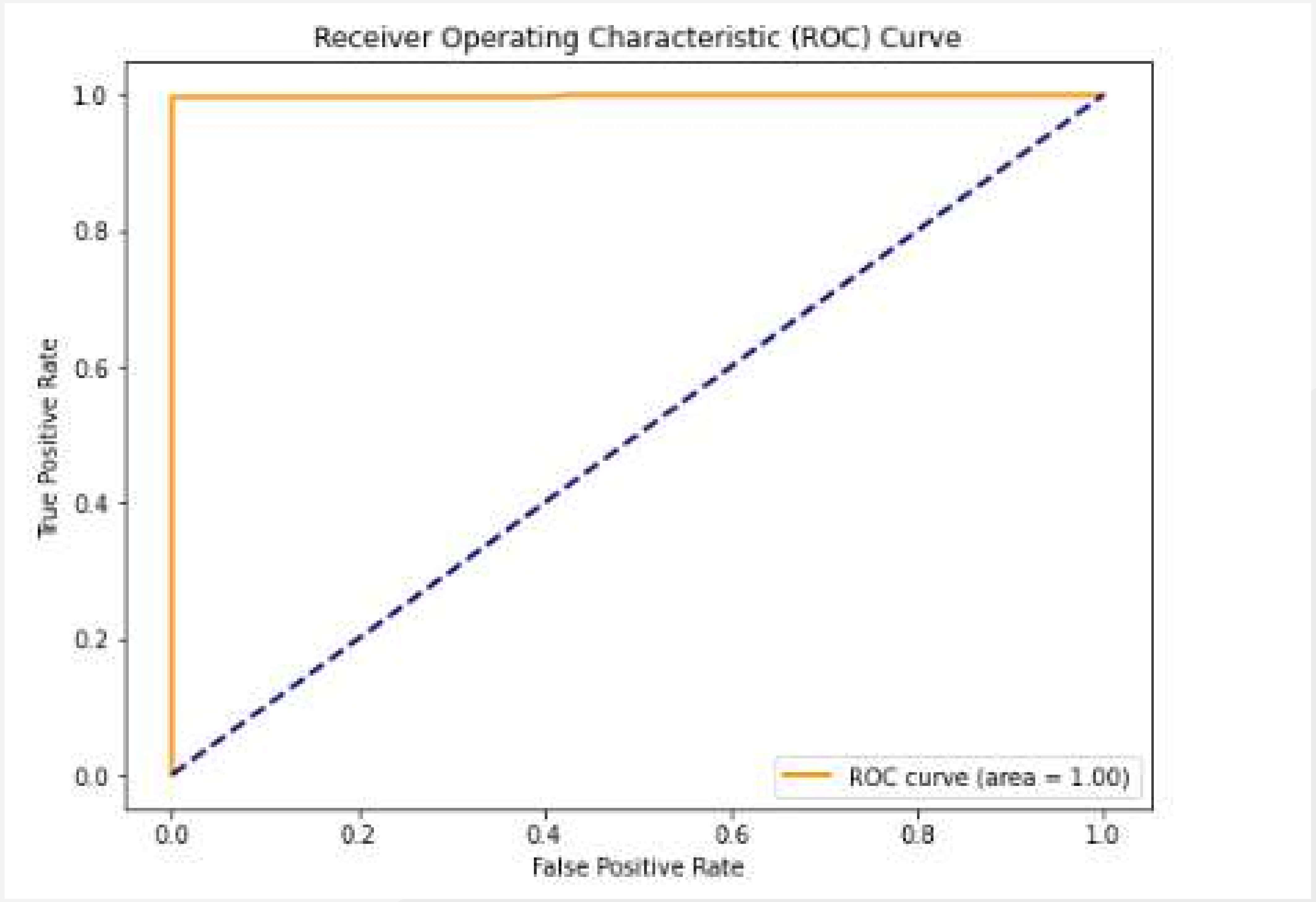
Now, we tune the logistic regression model and retrain the model

```
log_tuned = LogisticRegression(
    C = 1
    ,penalty = 'l1'
    ,solver = 'liblinear'
)
```

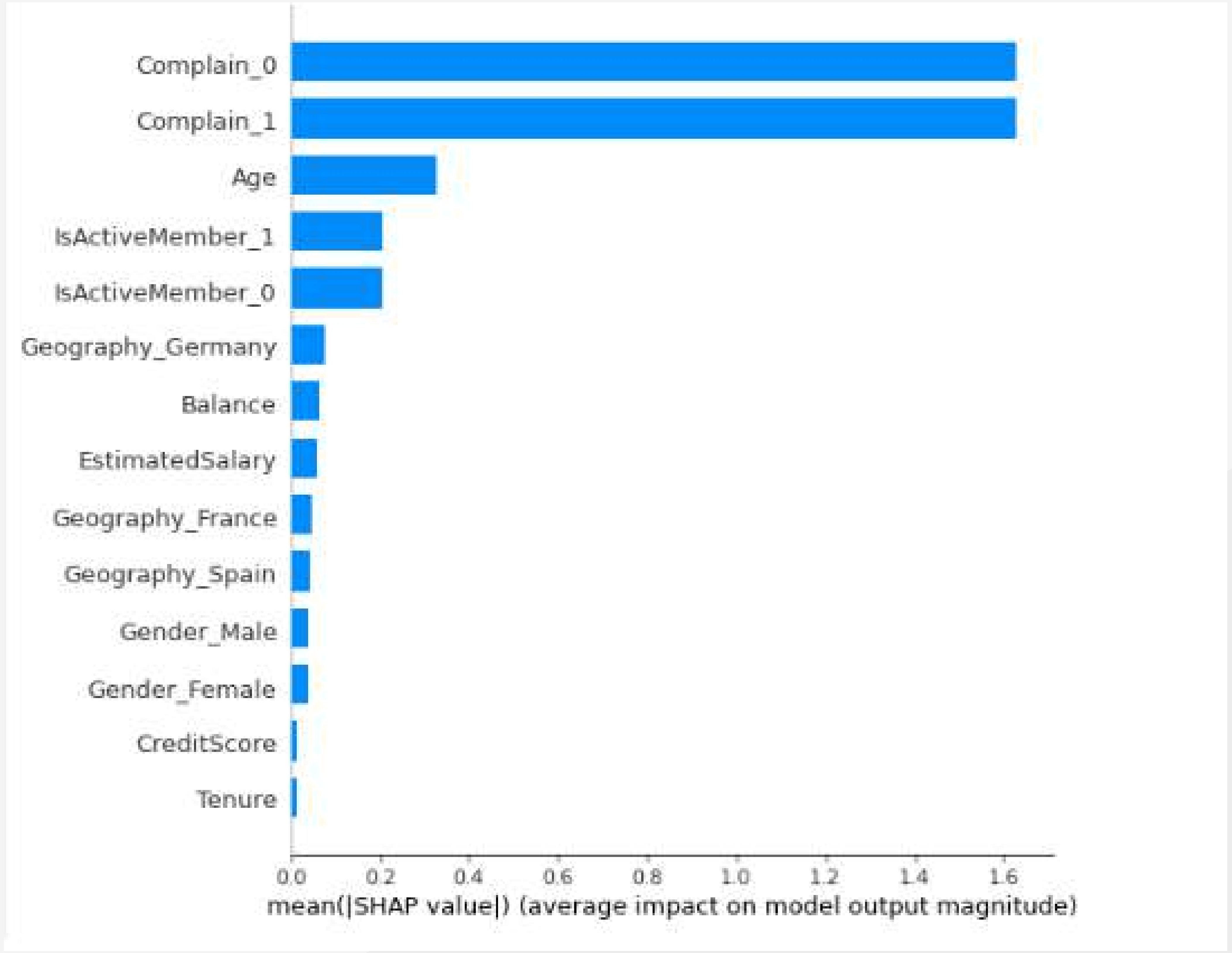
```
log_tuned.fit(X_train, y_train)
```

```
LogisticRegression(C=1, penalty='l1', solver='liblinear')
```

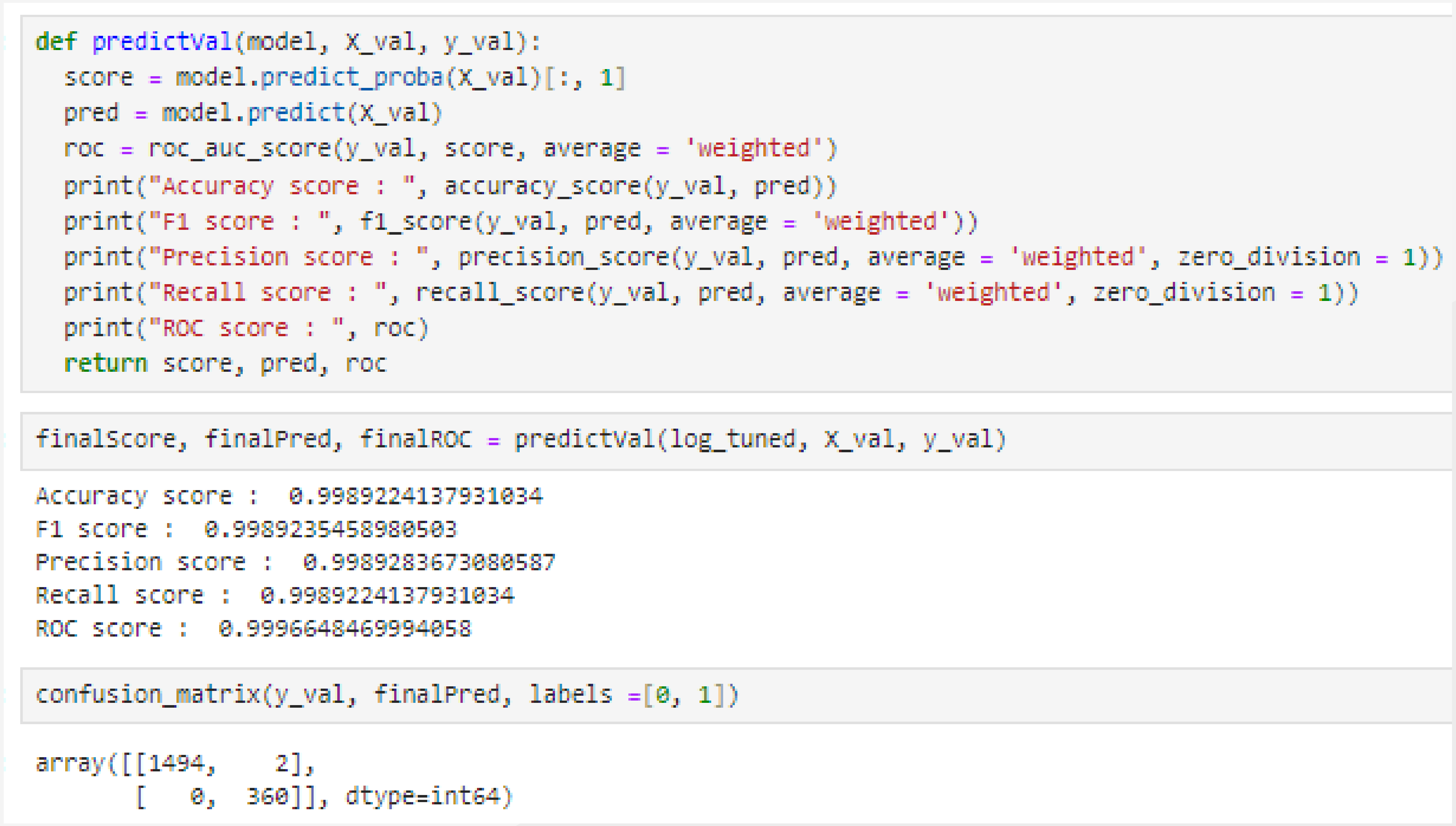
We then set the new parameters to our previous model and fit it again to the training data.



We achieved a high ROC curve.



From the shap plot, the most impact came from Complain, Age, and IsActiveMember features.



```
def predictVal(model, X_val, y_val):
  score = model.predict_proba(X_val)[:, 1]
  pred = model.predict(X_val)
  roc = roc_auc_score(y_val, score, average = 'weighted')
  print("Accuracy score : ", accuracy_score(y_val, pred))
  print("F1 score : ", f1_score(y_val, pred, average = 'weighted'))
  print("Precision score : ", precision_score(y_val, pred, average = 'weighted', zero_division = 1))
  print("Recall score : ", recall_score(y_val, pred, average = 'weighted', zero_division = 1))
  print("ROC score : ", roc)
  return score, pred, roc
```

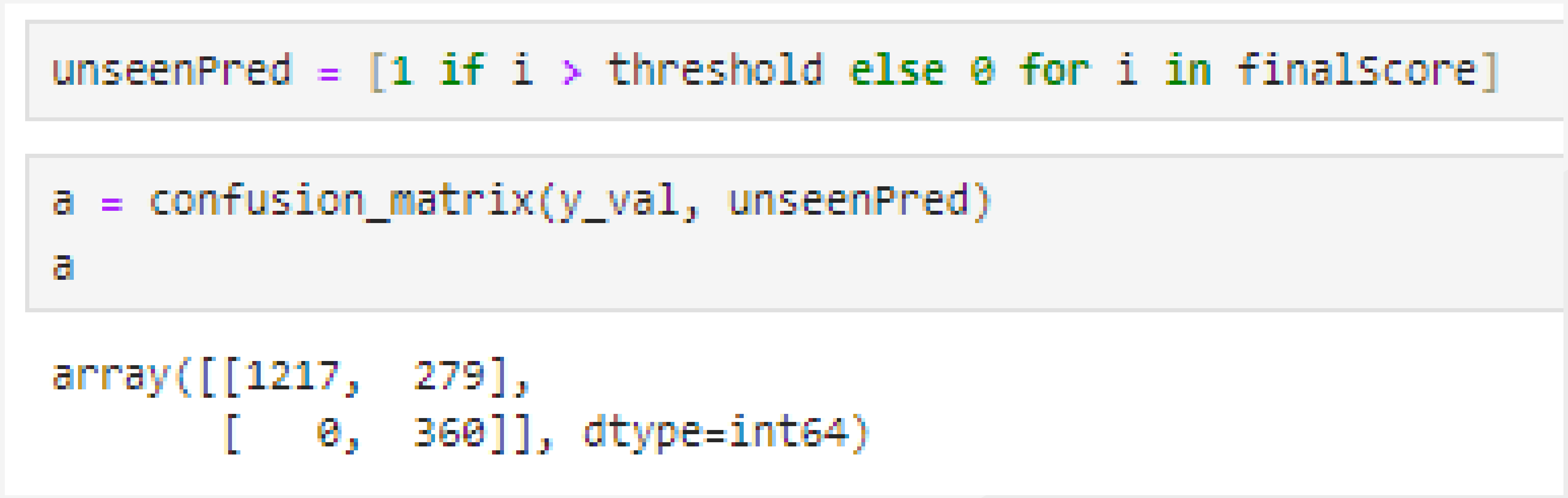
```
finalScore, finalPred, finalROC = predictVal(log_tuned, X_val, y_val)
```

```
Accuracy score :  0.9989224137931034
F1 score :  0.9989235458980503
Precision score :  0.9989283673080587
Recall score :  0.9989224137931034
ROC score :  0.9996648469994058
```

```
confusion_matrix(y_val, finalPred, labels =[0, 1])
```

```
array([[1494,    2],
       [   0,  360]], dtype=int64)
```

We go on with predicting the final data using the model we tuned.

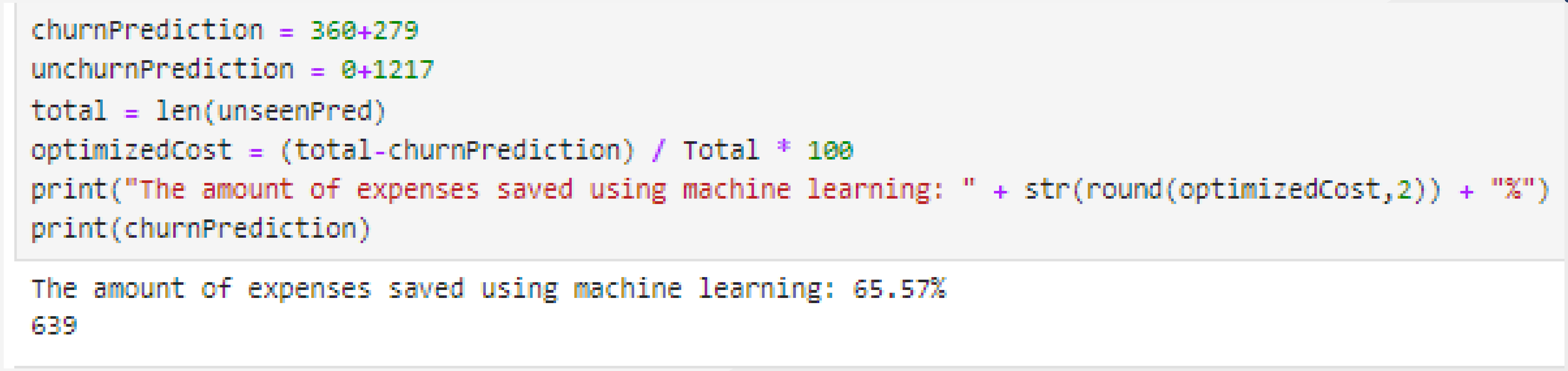


```
unseenPred = [1 if i > threshold else 0 for i in finalScore]
```

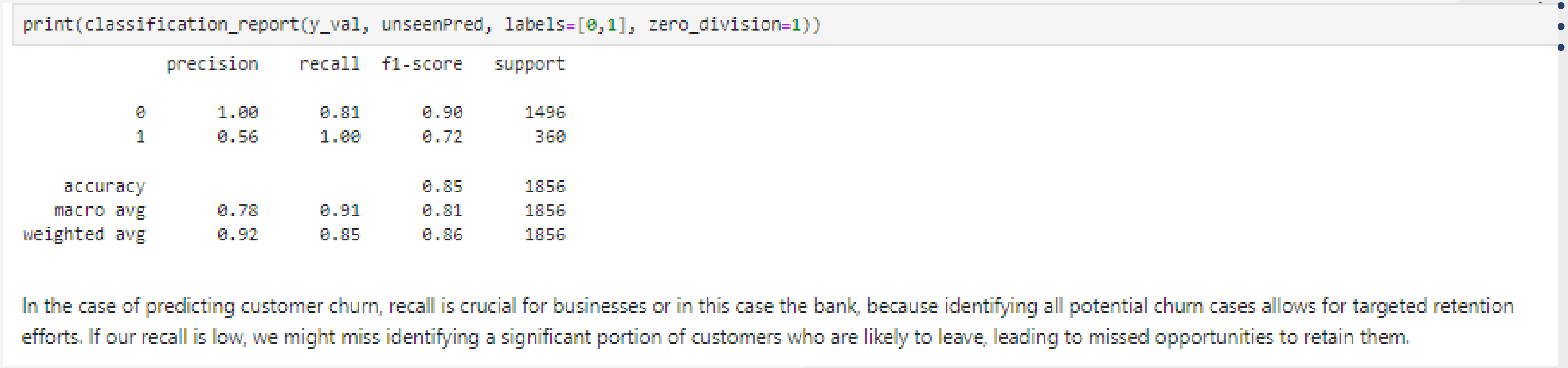
```
a = confusion_matrix(y_val, unseenPred)
a
```

```
array([[1217,  279],
       [   0,  360]], dtype=int64)
```

Then, we predict the unseen data using the tuned logistic regression model we had.


```
churnPrediction = 360+279
unchurnPrediction = 0+1217
total = len(unseenPred)
optimizedCost = (total-churnPrediction) / Total * 100
print("The amount of expenses saved using machine learning: " + str(round(optimizedCost,2)) + "%")
print(churnPrediction)
```

```
The amount of expenses saved using machine learning: 65.57%
639
```



```
print(classification_report(y_val, unseenPred, labels=[0,1], zero_division=1))
```

```
              precision    recall  f1-score   support

           0       1.00      0.81      0.90      1496
           1       0.56      1.00      0.72       360

    accuracy                           0.85      1856
   macro avg       0.78      0.91      0.81      1856
weighted avg       0.92      0.85      0.86      1856
```

In the case of predicting customer churn, recall is crucial for businesses or in this case the bank, because identifying all potential churn cases allows for targeted retention efforts. If our recall is low, we might miss identifying a significant portion of customers who are likely to leave, leading to missed opportunities to retain them.

BUSINESS CASE STUDY

By using the model prediction output to as guidance to reach out customers, the Bank could save up to **65.57%** in expenses (Marketing efforts, time costs, operational costs, etc.)

CONCLUSION & RECOMMENDATIONS

CONCLUSION

Supervised Machine Learning has been proven to be able to predict the customer ' s churn response with 1.00 roc_auc score and recall score. It also helps the company to reduce expenses up to 66% from reaching out customers.

RECOMMENDATIONS

1. Tailored Loyalty Programs: Develop personalized loyalty programs based on customer segments. Offer incentives and rewards to high-churn-risk groups to encourage their loyalty.

2. Improved Customer Service: Enhance customer service, especially for customers who frequently complain. Swift and effective complaint resolution can significantly improve customer satisfaction and retention rates.

3. Product Bundling: Encourage customers to subscribe to multiple bank products by offering bundled services. Create packages that cater to various needs, providing an incentive for customers to stay.

4. Income-Adjusted Services: Customize services based on income levels. Offer financial products that align with the financial capacity of customers, ensuring affordability and value for money.

5. Age-Specific Engagement: Understand and address the unique requirements of different age groups. Develop age-specific products and services, ensuring relevance and appeal across generations.

RECOMMENDATIONS

6. Gender-Inclusive Services: Create banking services that address the specific needs of both genders. Recognize diverse financial goals and preferences, making the bank more inclusive.

7. Active Member Recognition: Acknowledge and reward active members. Offer exclusive services, discounts, or privileges to keep active members engaged and loyal.

8. Cultural Sensitivity: Recognize and respect cultural differences, especially for customers from different countries. Offer services that resonate with their cultural values, building trust and customer loyalt.

9. Regular Feedback Mechanism: Implement a systematic feedback system. Regularly collect feedback from customers to understand their evolving needs and concerns, allowing the bank to adapt its strategies accordingly.

10. Data-Driven Decisions: Invest in advanced analytics and AI tools to continually analyze customer behavior. Predictive models can identify potential churners early, allowing the bank to take proactive measures.

RECOMMENDATIONS

11. Increase data points and volume to the model to prevent underfitting and improves overall score in predicting unseen datasets.
12. Investigate the cause of complains upon the system, fixed them, iterate, and hopefully gain new insights.

THANK YOU

CONTACTS

+6287764260779

arvioanandi@gmail.com

PROJECT LINK

Bank Customer Churn Prediction