Import Libraries

In [71]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, confus
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_predict
```

Exploratory Data Analysis

In [2]:
```
df = pd.read_csv('/content/Churn_Modelling.csv')
```

In [3]:
```
df.head()
```

Out[3]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	N
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	

In [4]:
```
df.shape
```

Out[4]:
```
(10000, 14)
```

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   RowNumber        10000 non-null  int64  
 1   CustomerId       10000 non-null  int64  
 2   Surname          10000 non-null  object 
 3   CreditScore      10000 non-null  int64  
 4   Geography        10000 non-null  object 
 5   Gender           10000 non-null  object 
 6   Age              10000 non-null  int64  
 7   Tenure           10000 non-null  int64  
 8   Balance          10000 non-null  float64
 9   NumOfProducts    10000 non-null  int64  
 10  HasCrCard        10000 non-null  int64  
 11  IsActiveMember   10000 non-null  int64  
 12  EstimatedSalary  10000 non-null  float64
 13  Exited           10000 non-null  int64  
dtypes: float64(2), int64(9), object(3)
memory usage: 1.1+ MB
```

In [6]:
```
df['Surname'].nunique()
```

Out[6]:
```
2932
```

In [7]:
```
df['Geography'].value_counts()
```

Out[7]:
```
France     5014
Germany    2509
Spain      2477
Name: Geography, dtype: int64
```

In [8]:
```
df['Gender'].value_counts()
```

Out[8]:
```
Male      5457
Female    4543
Name: Gender, dtype: int64
```

In [9]:
```
df['Exited'].value_counts()
```

Out[9]:
```
0    7963
1    2037
Name: Exited, dtype: int64
```

Statistical Analysis

In [10]:
```
df.describe()
```

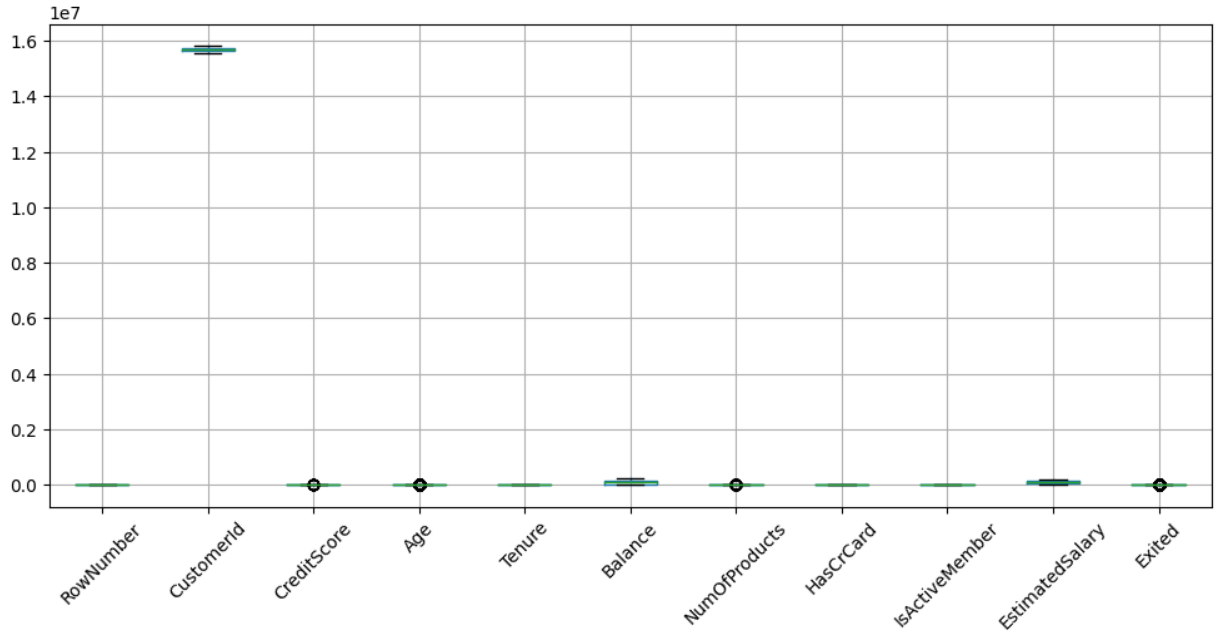
Out[10]:

	RowNumber	CustomerId	CreditScore	Age	Tenure	Balance	NumO
count	10000.00000	1.000000e+04	10000.000000	10000.000000	10000.000000	10000.000000	100
mean	5000.50000	1.569094e+07	650.528800	38.921800	5.012800	76485.889288	
std	2886.89568	7.193619e+04	96.653299	10.487806	2.892174	62397.405202	
min	1.00000	1.556570e+07	350.000000	18.000000	0.000000	0.000000	
25%	2500.75000	1.562853e+07	584.000000	32.000000	3.000000	0.000000	
50%	5000.50000	1.569074e+07	652.000000	37.000000	5.000000	97198.540000	
75%	7500.25000	1.575323e+07	718.000000	44.000000	7.000000	127644.240000	
max	10000.00000	1.581569e+07	850.000000	92.000000	10.000000	250898.090000	

BoxPlot

In [11]:
```
plt.figure(figsize = (12, 5))
df.boxplot(rot = 45)
```

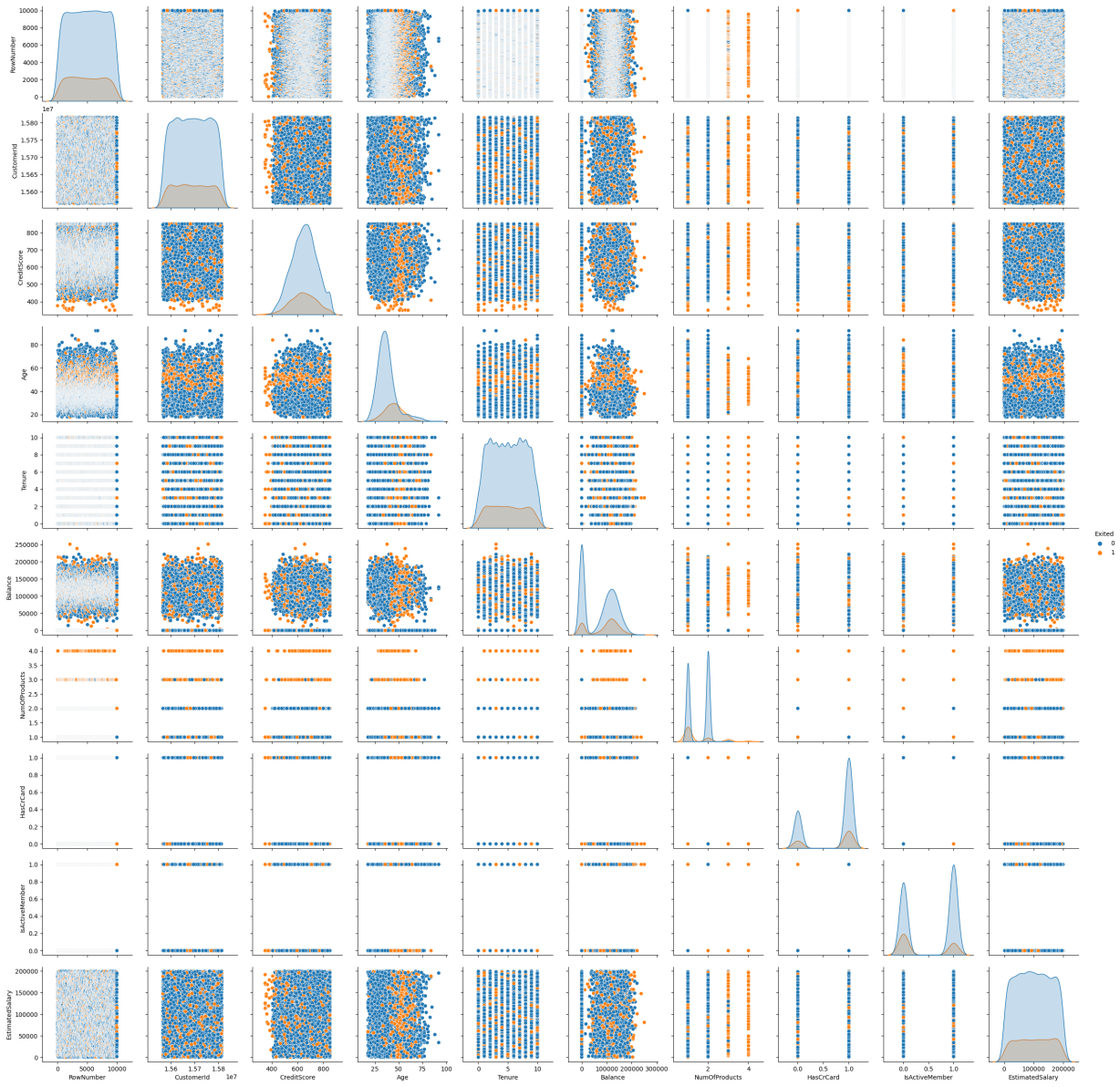
Out[11]: <Axes: >



PairPlot

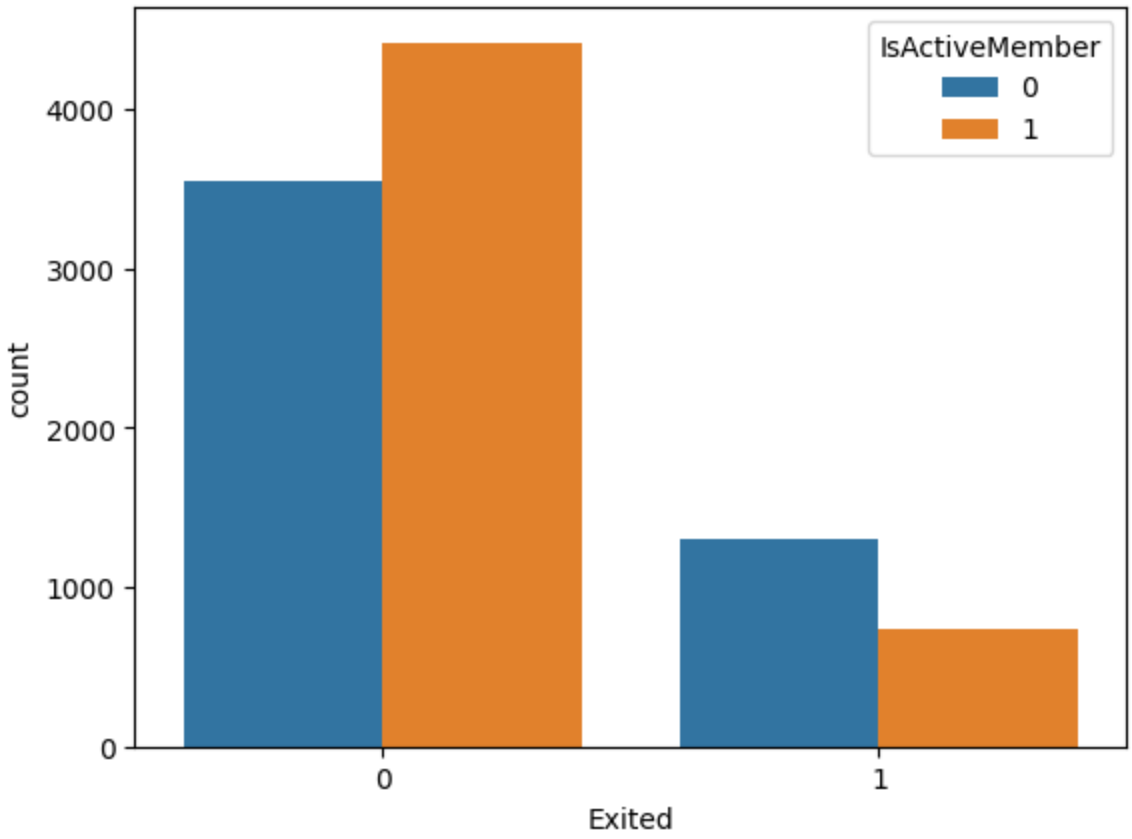
In [12]:
```
sns.pairplot(data = df, hue = 'Exited')
```

Out[12]: <seaborn.axisgrid.PairGrid at 0x7dcb09d2df90>



In [13]:
```
sns.countplot(x= df['Exited'], data = df, hue = df['IsActiveMember'])
```

Out[13]:
```
<Axes: xlabel='Exited', ylabel='count'>
```

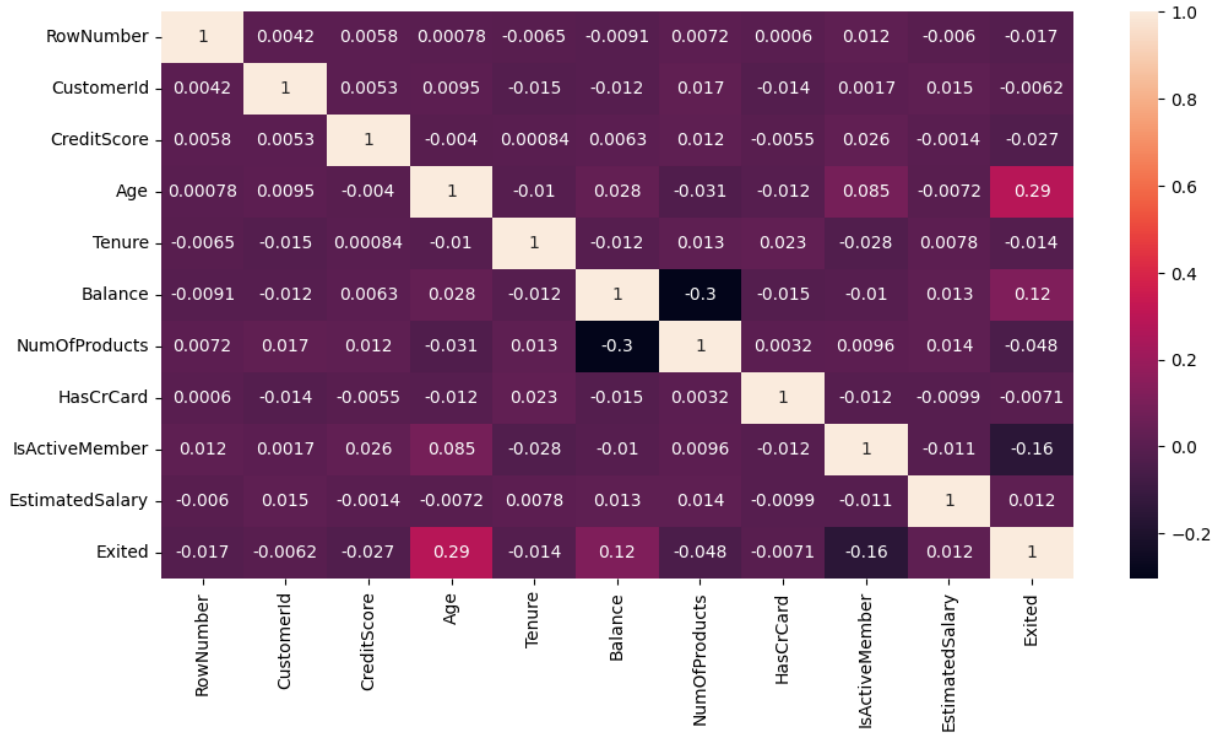


Heatmap & Correlation

In [14]:
```
plt.figure(figsize = (12, 6))
sns.heatmap(df.corr(), annot= True)
```

```
<ipython-input-14-49e34adb52de>:2: FutureWarning: The default value of numeric_
only in DataFrame.corr is deprecated. In a future version, it will default to F
alse. Select only valid columns or specify the value of numeric_only to silence
this warning.
  sns.heatmap(df.corr(), annot= True)
```

Out[14]: <Axes: >



Data Preprocessing

In [69]:
```
missing_values = []
for col in df.columns:
  missing_values.append(sum(df[col].isnull()))
missing_values = pd.DataFrame(missing_values)
col = pd.DataFrame(df.columns)
total_missing = pd.concat([col, missing_values], axis = 1)
total_missing.columns = ['Columns', 'Missing_values']
total_missing
```

Out[69]:

	Columns	Missing_values
0	RowNumber	0
1	CustomerId	0
2	Surname	0
3	CreditScore	0
4	Geography	0
5	Gender	0
6	Age	0
7	Tenure	0
8	Balance	0
9	NumOfProducts	0
10	HasCrCard	0
11	IsActiveMember	0
12	EstimatedSalary	0
13	Exited	0

Outlier Detection

In [17]:
```
outlier = []
for col in df.columns:
  if df[col].dtypes != 'object':
    percentile25 = df[col].quantile(0.25)
    percentile75 = df[col].quantile(0.75)
    iqr = percentile75- percentile25

    lower_bound = percentile25 - 1.5 * iqr
    upper_bound = percentile75 + 1.5 * iqr

    outlier.append(sum((df[col] > upper_bound) | (df[col] < lower_bound)))
```

In [18]:
```
outlier = pd.DataFrame(outlier)
column = df.drop(['Geography', 'Surname', 'Gender'], axis = 1)
column = pd.DataFrame(column.columns)
total_outlier = pd.concat([column, outlier], axis = 1)
total_outlier.columns = ['Column', 'Outlier']
total_outlier
```

Out[18]:

	Column	Outlier
0	RowNumber	0
1	CustomerId	0
2	CreditScore	15
3	Age	359
4	Tenure	0
5	Balance	0
6	NumOfProducts	60
7	HasCrCard	0
8	IsActiveMember	0
9	EstimatedSalary	0
10	Exited	2037

Categorical Variables (Encoding)

In [19]:
```
encoder = LabelEncoder()
df['Gender'] = encoder.fit_transform(df['Gender'])
```

In [20]:
```
# we will select all the features except RowNumber, CustomerId, SurName as they d

x = df.iloc[:, 3:-1].values
y = df.iloc[:, -1].values
```

In [21]:
```
ct = ColumnTransformer(transformers = [('encoding', OneHotEncoder(), [1])], remai
x = np.array(ct.fit_transform(x))
```

In [22]:
```
# Split into Training and Testing

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, randor
```

Logistic Regression

In [23]:
```
regression = LogisticRegression()
regression.fit(x_train, y_train)
regression_prediction = regression.predict(x_test)
```

Evaluation

In [25]:
```
confusion_matrix(y_test, regression_prediction)
```

Out[25]:
```
array([[1573,   34],
       [ 365,   28]])
```

In [26]:
```
accuracy_score(y_test, regression_prediction)
```

Out[26]:
```
0.8005
```

In [27]:
```
precision_score(y_test, regression_prediction)
```

Out[27]:
```
0.45161290322580644
```

In [28]:
```
recall_score(y_test, regression_prediction)
```

Out[28]:
```
0.07124681933842239
```

Random Forest

In [29]:
```
rf_model = RandomForestClassifier(random_state = 42)
rf_model.fit(x_train, y_train)
rf_prediction = rf_model.predict(x_test)
```

Evaluation

In [30]:
```
confusion_matrix(y_test, rf_prediction)
```

Out[30]:
```
array([[1545,   62],
       [ 207,  186]])
```

In [31]:
```
accuracy_score(y_test, rf_prediction)
```

Out[31]:
```
0.8655
```

In [32]:
```
precision_score(y_test, rf_prediction)
```

Out[32]:
```
0.75
```

In [33]:
```
recall_score(y_test, rf_prediction)
```

Out[33]:
```
0.4732824427480916
```

Gradient Boosting

In [34]:
```
gb_model = GradientBoostingClassifier()
gb_model.fit(x_train, y_train)
gb_prediction = gb_model.predict(x_test)
```

Evaluation

In [35]:
```
confusion_matrix(y_test, gb_prediction)
```

Out[35]:
```
array([[1542,   65],
       [ 206,  187]])
```

In [36]:
```
accuracy_score(y_test, gb_prediction)
```

Out[36]:
```
0.8645
```

In [37]:
```
precision_score(y_test, gb_prediction)
```

Out[37]:
```
0.7420634920634921
```

In [38]:
```
recall_score(y_test, gb_prediction)
```

Out[38]:
```
0.4758269720101781
```

Remove Noise

In [39]:
```
df_new = df.copy()
```

In [40]:
```
df_new.head()
```

Out[40]:

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

Compute Outlier

In [43]:
```
for col in df_new.columns[:-1]:
  if df_new[col].dtypes != 'object':
    percentile25 = df_new[col].quantile(0.25)
    percentile75 = df_new[col].quantile(0.75)
    iqr = percentile75- percentile25

    lower_bound = percentile25 - 1.5 * iqr
    upper_bound = percentile75 + 1.5 * iqr
    df_new[col] = np.where(
        df_new[col] > upper_bound,
        upper_bound,
        np.where(
            df_new[col] < lower_bound,
            lower_bound,
            df_new[col]
        )
    )
```

In [45]:
```
df_new = pd.get_dummies(df_new, columns = ['Geography'],drop_first = True)
```

In [46]:
```
df_new.head()
```

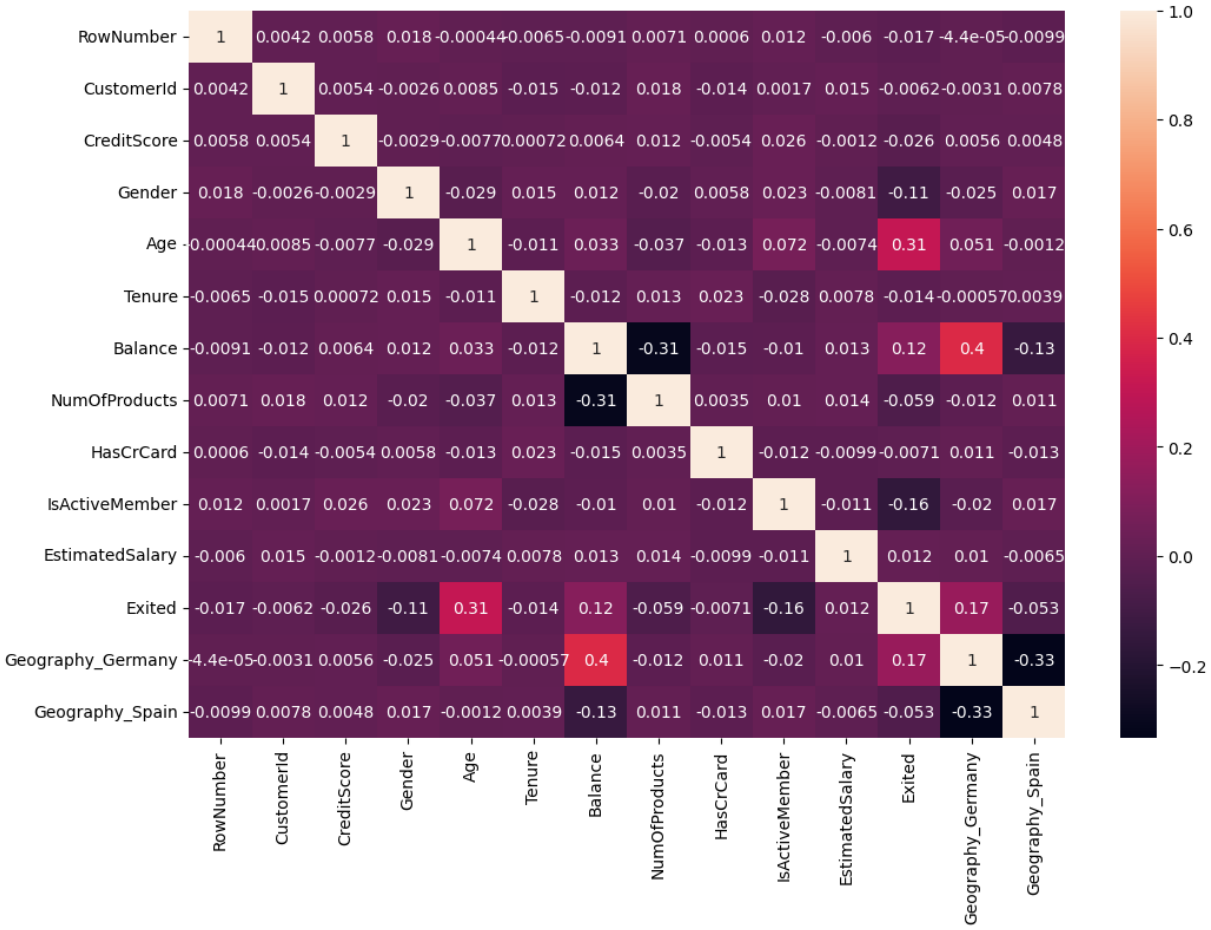
Out[46]:

	RowNumber	CustomerId	Surname	CreditScore	Gender	Age	Tenure	Balance	NumOfProduc
0	1.0	15634602.0	Hargrave	619.0	0.0	42.0	2.0	0.00	1
1	2.0	15647311.0	Hill	608.0	0.0	41.0	1.0	83807.86	1
2	3.0	15619304.0	Onio	502.0	0.0	42.0	8.0	159660.80	3
3	4.0	15701354.0	Boni	699.0	0.0	39.0	1.0	0.00	2
4	5.0	15737888.0	Mitchell	850.0	0.0	43.0	2.0	125510.82	1

In [48]:
```
plt.figure(figsize = (12, 8))
sns.heatmap(df_new.corr(), annot = True)
```

```
<ipython-input-48-f31fa33abab1>:2: FutureWarning: The default value of numeric_
only in DataFrame.corr is deprecated. In a future version, it will default to F
alse. Select only valid columns or specify the value of numeric_only to silence
this warning.
  sns.heatmap(df_new.corr(), annot = True)
```

Out[48]: <Axes: >



In [49]:
```
Xx = df_new.drop(['RowNumber', 'CustomerId', 'Surname', 'Exited'], axis = 1)
yy = df_new['Exited']
```

In [50]:
```
train_x, test_x, train_y, test_y = train_test_split(Xx, yy, test_size= 0.2, randc
```

Data Scaling

In [51]:
```
scaling = StandardScaler()
train_x = scaling.fit_transform(train_x)
test_x = scaling.transform(test_x)
```

Logistic Regression

In [52]:
```
lg_new = LogisticRegression()
lg_new.fit(train_x, train_y)
lg_pred = lg_new.predict(test_x)
```

In [53]:
```
accuracy_score(test_y, lg_pred)
```

Out[53]: 0.8165

Random Forest

In [54]:
```
rf_new = RandomForestClassifier(random_state = 42)
rf_new.fit(train_x, train_y)
rf_new_pred = rf_new.predict(test_x)
```

In [55]:
```
accuracy_score(test_y, rf_new_pred)
```

Out[55]: 0.867

Gradient Boosting

In [57]:
```
gb_new = GradientBoostingClassifier()
gb_new.fit(train_x, train_y)
gb_new_pred = gb_new.predict(test_x)
```

In [58]:
```
accuracy_score(test_y, gb_new_pred)
```

Out[58]: 0.8655

Tuning Random Forest

In [59]:
```
param_grid = {
    'n_estimators': [50, 100, 150],
    'max_depth': [4,5,6],
    'criterion': ['gini', 'entropy']
}
```

In [60]:
```
# Using GridSearhCV for hyperparameter tuning

grid_search = GridSearchCV(rf_new, param_grid = param_grid, cv = 3)
grid_search.fit(train_x, train_y)
```

Out[60]:
```
GridSearchCV(cv=3, estimator=RandomForestClassifier(random_state=42),
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [4, 5, 6],
                         'n_estimators': [50, 100, 150]})
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [61]:
```
# Best Parameters

grid_search.best_params_
```

Out[61]:
```
{'criterion': 'entropy', 'max_depth': 6, 'n_estimators': 150}
```

In [62]:
```
forest_model = RandomForestClassifier(criterion = 'entropy', max_depth = 6, n_est
```

In [63]:
```
validation = cross_val_predict(forest_model, train_x, train_y, cv = 5)
```

In [64]:
```
accuracy_score(train_y, validation)
```

Out[64]:
```
0.855625
```

In [66]:
```
forest_model.fit(train_x, train_y)
```

Out[66]:
```
RandomForestClassifier(criterion='entropy', max_depth=6, n_estimators=150,
                       random_state=42)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [67]:
```
forest_prediction = forest_model.predict(test_x)
```

In [68]:
```
accuracy_score(test_y, forest_prediction)
```

Out[68]:
```
0.8585
```