Customer Churn Rate

In [1]:
```
#import libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import warnings
warnings.filterwarnings('ignore')
```

Loading the dataset

In [2]:
```
churn = pd.read_csv(r"C:\Users\HP\PGA 32\Machine Learning\CODSOFT\Customer_Churn\Churn_Modelling.csv")
```

In [3]:
```
churn
```

Out[3]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsA
0	1	15634602	Hargrave	619	France	Female	42	2	0.00	1	1	
1	2	15647311	Hill	608	Spain	Female	41	1	83807.86	1	0	
2	3	15619304	Onio	502	France	Female	42	8	159660.80	3	1	
3	4	15701354	Boni	699	France	Female	39	1	0.00	2	0	
4	5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	1	
...	...	...	...	...	...	...	...	...	...	...	...	
9995	9996	15606229	Obijiaku	771	France	Male	39	5	0.00	2	1	
9996	9997	15569892	Johnstone	516	France	Male	35	10	57369.61	1	1	
9997	9998	15584532	Liu	709	France	Female	36	7	0.00	1	0	
9998	9999	15682355	Sabbatini	772	Germany	Male	42	3	75075.31	2	1	
9999	10000	15628319	Walker	792	France	Female	28	4	130142.79	1	1	

10000 rows × 14 columns

In [4]:
```
churn.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 [5]:
```
churn = churn.drop(['RowNumber', 'CustomerId', 'Surname'], axis=1)
churn
```

Out[5]:

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
0	619	France	Female	42	2	0.00	1	1	1	101348.88	1
1	608	Spain	Female	41	1	83807.86	1	0	1	112542.58	0
2	502	France	Female	42	8	159660.80	3	1	0	113931.57	1
3	699	France	Female	39	1	0.00	2	0	0	93826.63	0
4	850	Spain	Female	43	2	125510.82	1	1	1	79084.10	0
...	...	...	...	...	...	...	...	...	...	...	...
9995	771	France	Male	39	5	0.00	2	1	0	96270.64	0
9996	516	France	Male	35	10	57369.61	1	1	1	101699.77	0
9997	709	France	Female	36	7	0.00	1	0	1	42085.58	1
9998	772	Germany	Male	42	3	75075.31	2	1	0	92888.52	1
9999	792	France	Female	28	4	130142.79	1	1	0	38190.78	0

10000 rows × 11 columns

Missing Values

In [6]:
```
churn.isna().sum()
```

Out[6]:
```
CreditScore        0
Geography          0
Gender             0
Age                0
Tenure             0
Balance            0
NumOfProducts      0
HasCrCard          0
IsActiveMember     0
EstimatedSalary    0
Exited             0
dtype: int64
```

In [7]:
```
churn.shape
```

Out[7]:
```
(10000, 11)
```

In [8]:
```
churn.duplicated().sum()
```

Out[8]:
```
0
```

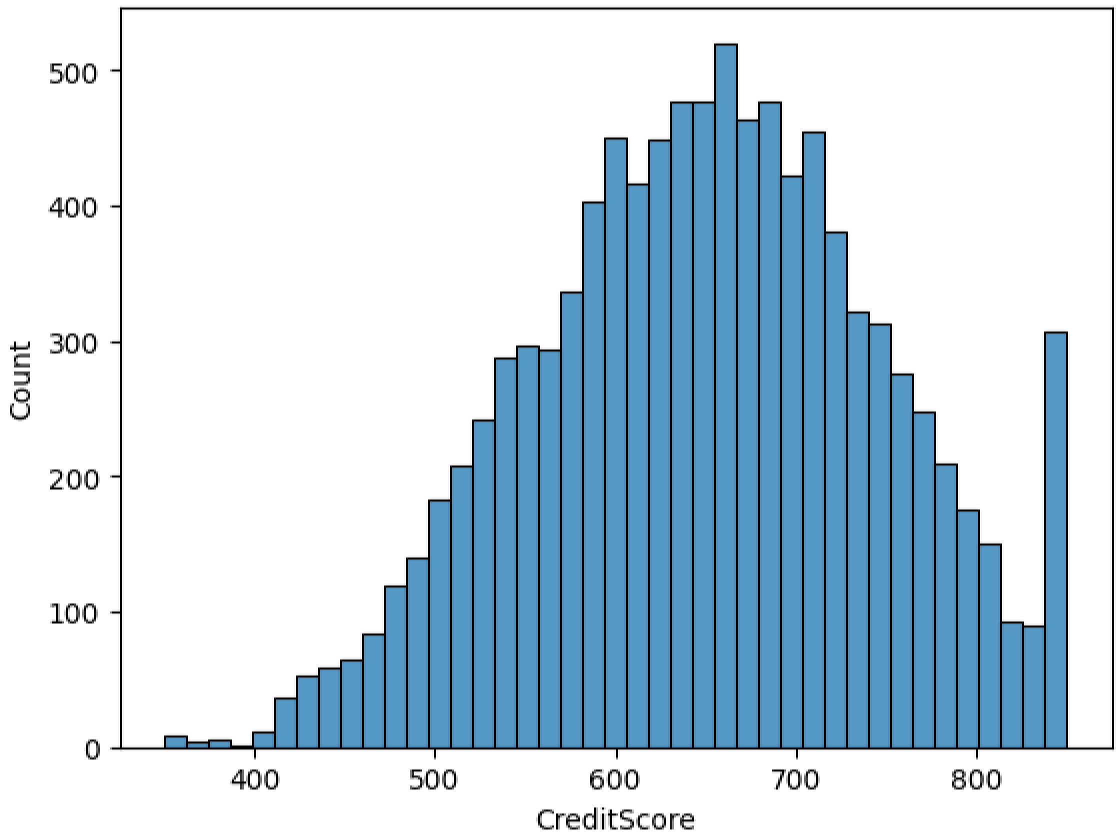
Numerical Columns

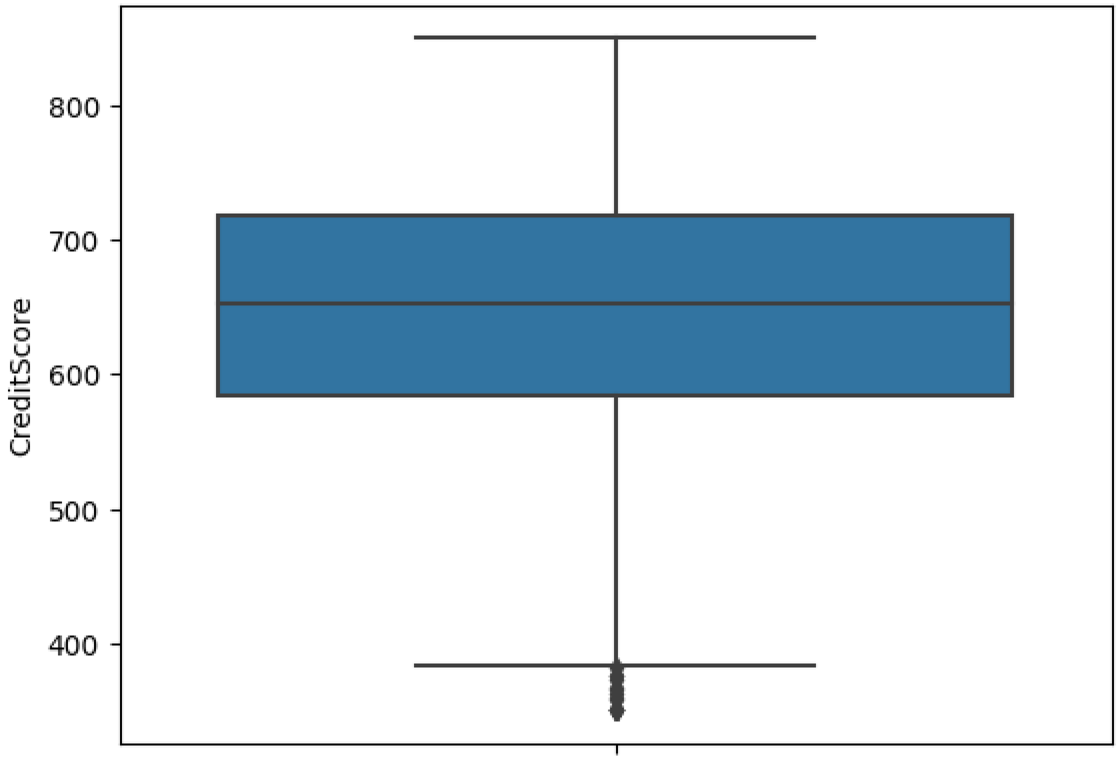
In [9]:
```
churn.dtypes[churn.dtypes!="object"]
```

Out[9]:
```
CreditScore          int64
Age                  int64
Tenure               int64
Balance            float64
NumOfProducts        int64
HasCrCard            int64
IsActiveMember       int64
EstimatedSalary    float64
Exited               int64
dtype: object
```

In [10]:
```
def numerical(data,var,graph_plot=True):
    missing=data[var].isnull().sum()
    min_n=data[var].min()
    max_n=data[var].max()
    var_n=data[var].var()
    std_n=data[var].std()
    p1=data[var].quantile(.01)
    p10=data[var].quantile(.1)
    p25=data[var].quantile(.25)
    p50=data[var].quantile(.5)
    p75=data[var].quantile(.75)
    p99=data[var].quantile(.99)
    iqr=p75-p25


    if graph_plot==True:
        sns.histplot(data[var])
        plt.show()
        sns.boxplot(y=data[var])
        plt.show()

    results={"missing":missing,"min":min_n,"max":max_n,"var":var_n,"std":std_n,
            "p1":p1,"p10":p10,"p25":p25,"p50":p50,"p75":p75,"p99":p99}
    return results
```

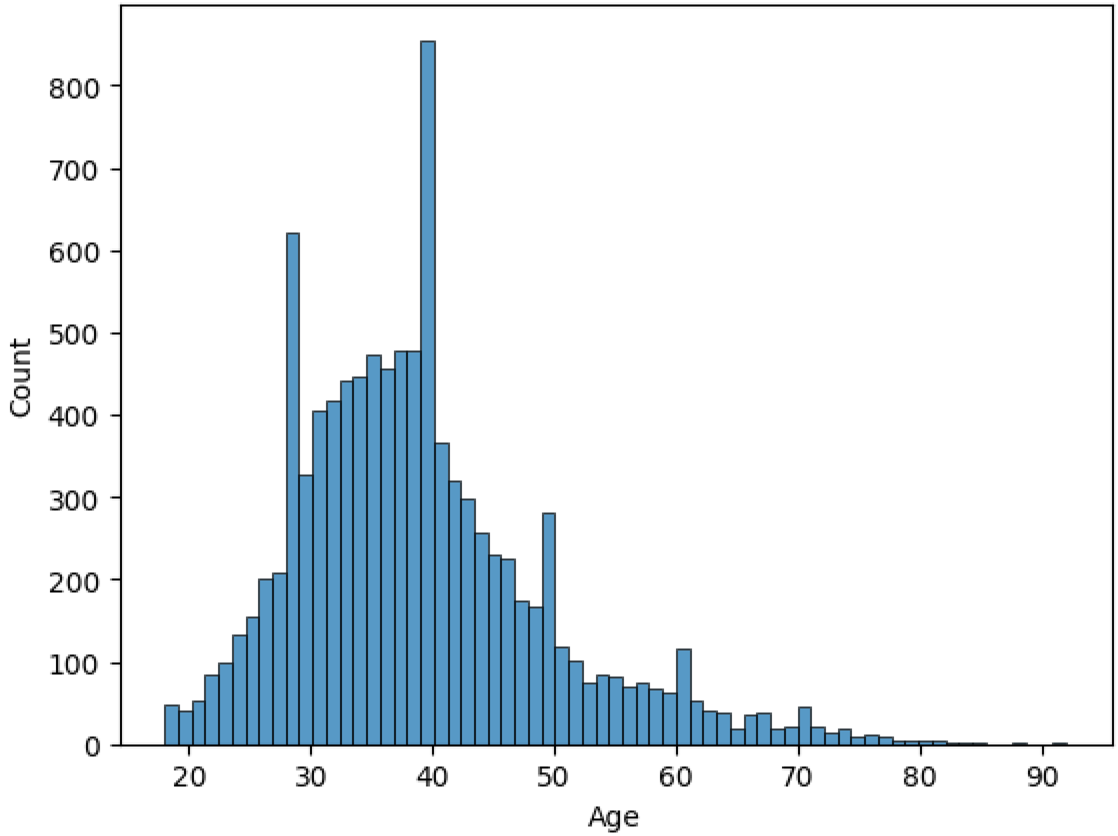
In [11]:
```
numerical(data=churn, var="CreditScore")
```

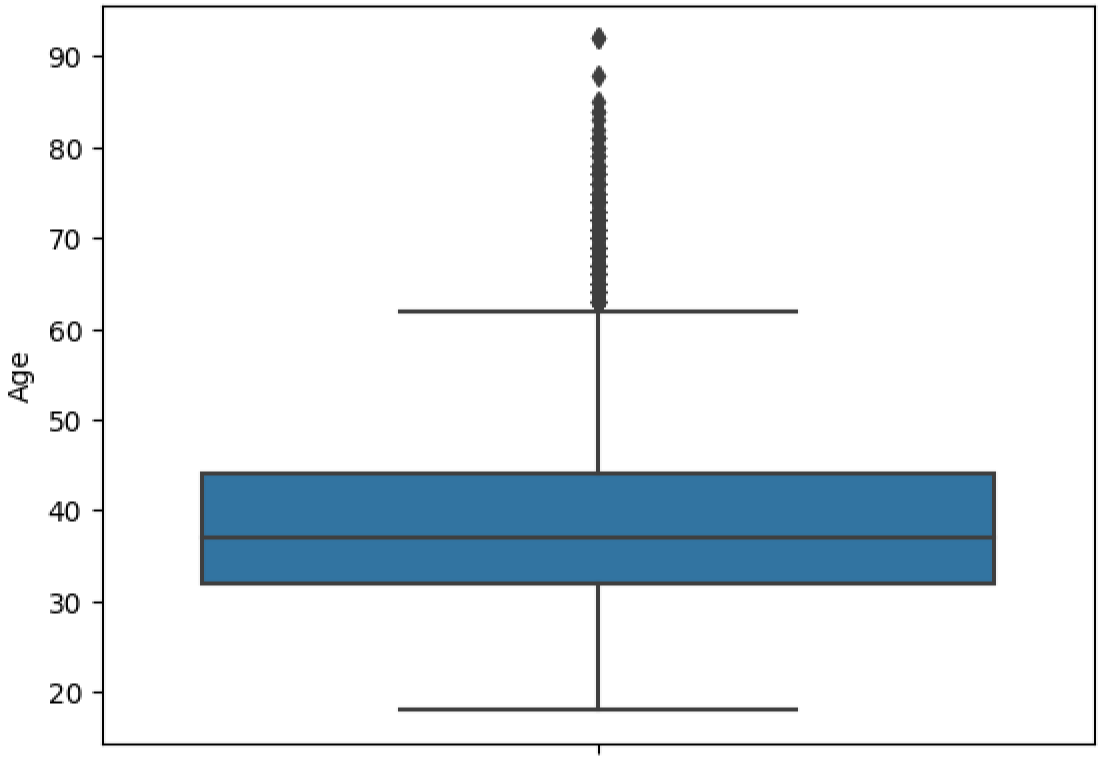




Out[11]:
```
{'missing': 0,
 'min': 350,
 'max': 850,
 'var': 9341.860156575705,
 'std': 96.65329873613061,
 'p1': 432.0,
 'p10': 521.0,
 'p25': 584.0,
 'p50': 652.0,
 'p75': 718.0,
 'p99': 850.0}
```

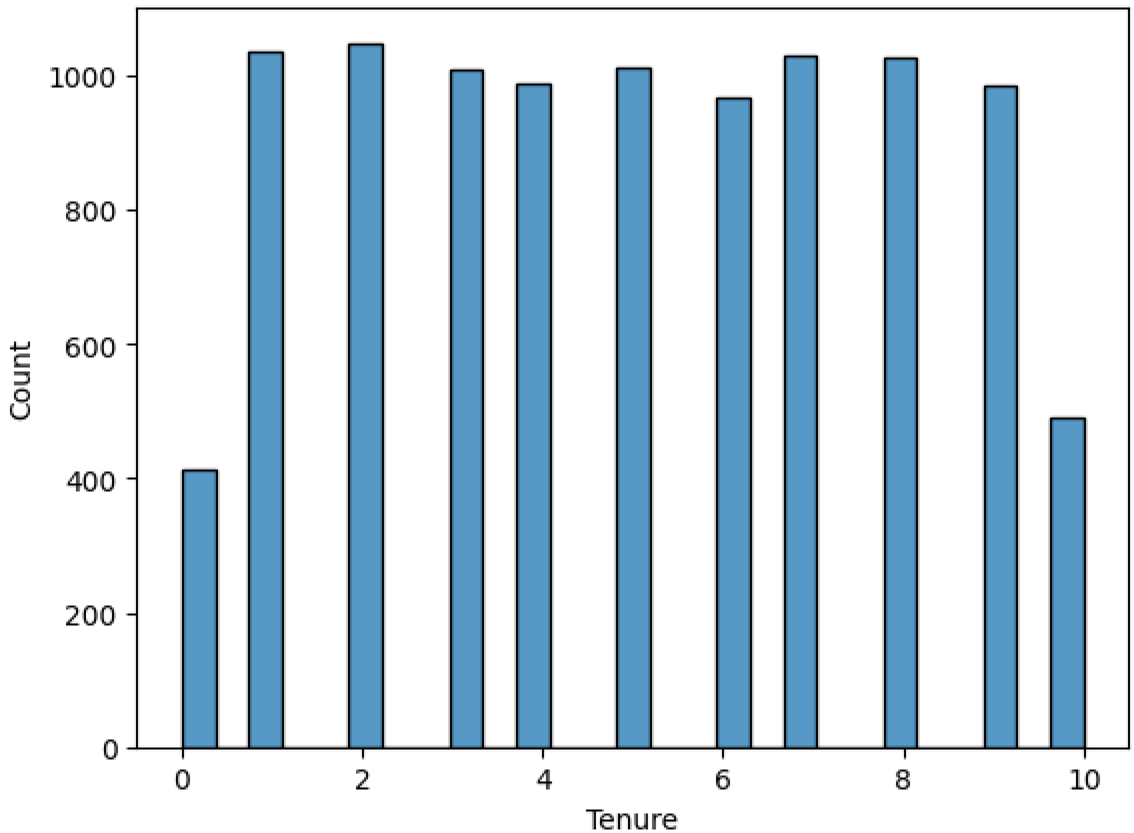
In [12]:
```
numerical(data=churn, var="Age")
```

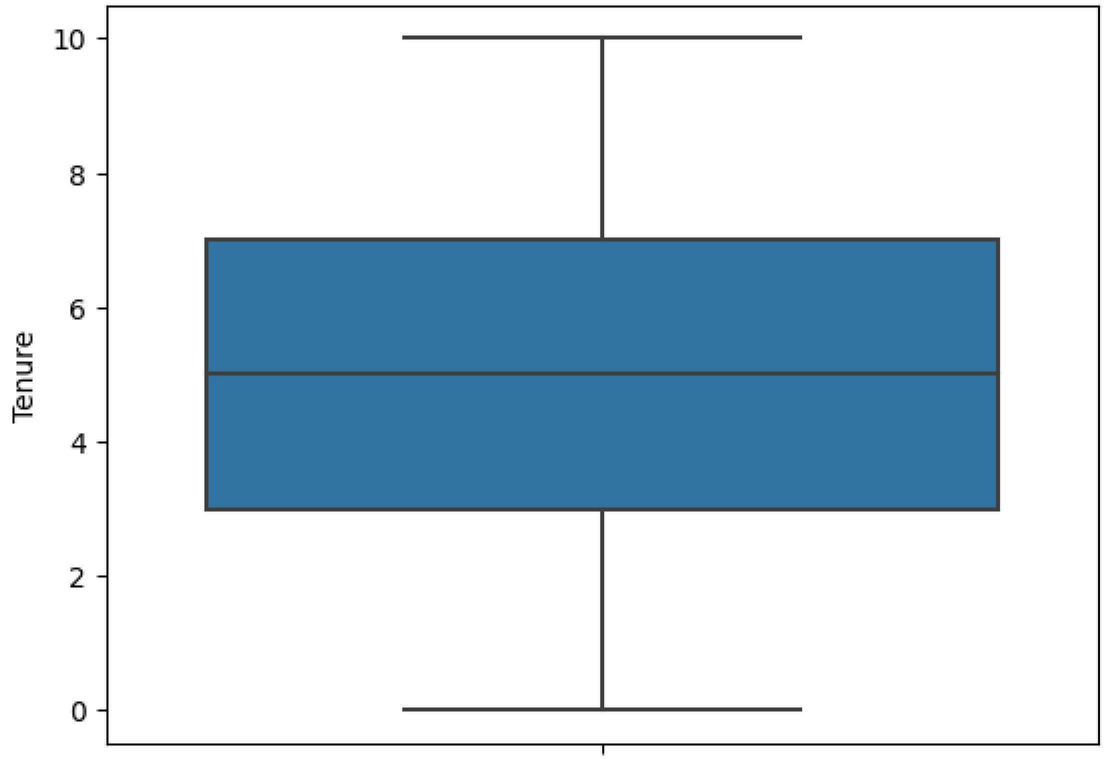




Out[12]:
```
{'missing': 0,
 'min': 18,
 'max': 92,
 'var': 109.99408416841645,
 'std': 10.487806451704591,
 'p1': 21.0,
 'p10': 27.0,
 'p25': 32.0,
 'p50': 37.0,
 'p75': 44.0,
 'p99': 72.0}
```

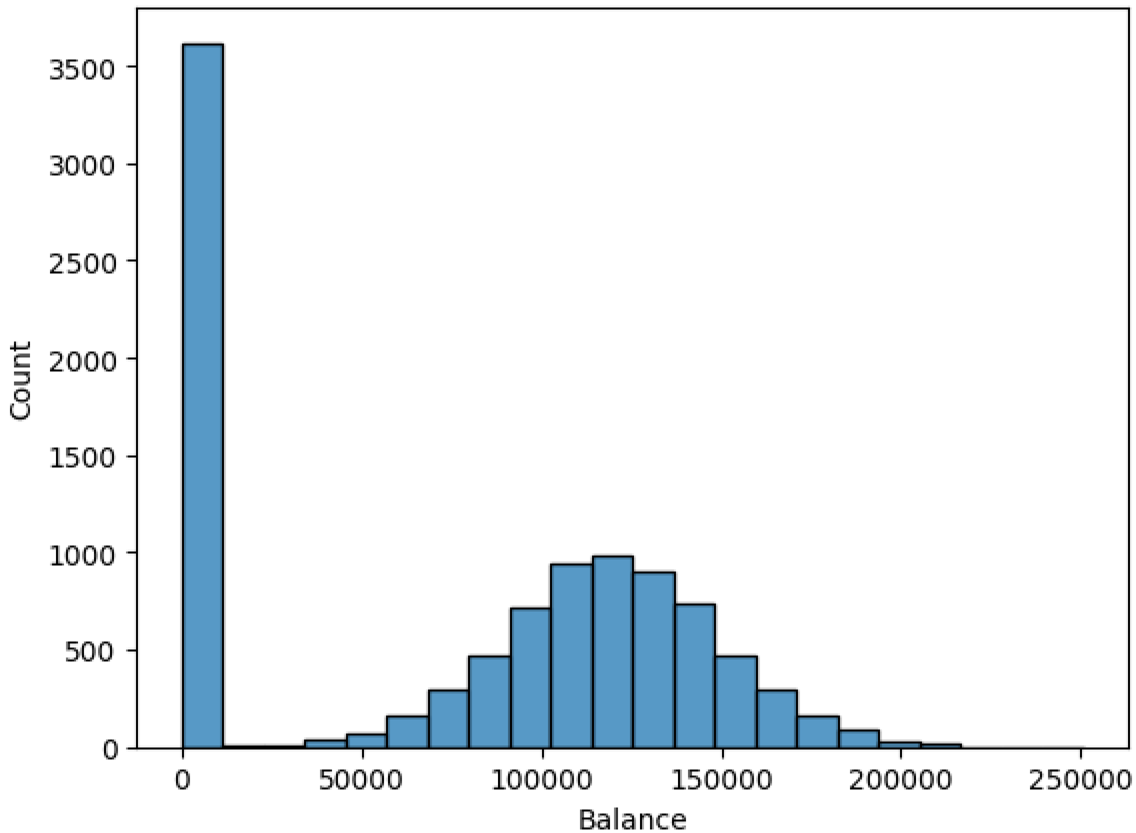
In [13]:
```
numerical(data=churn, var="Tenure")
```

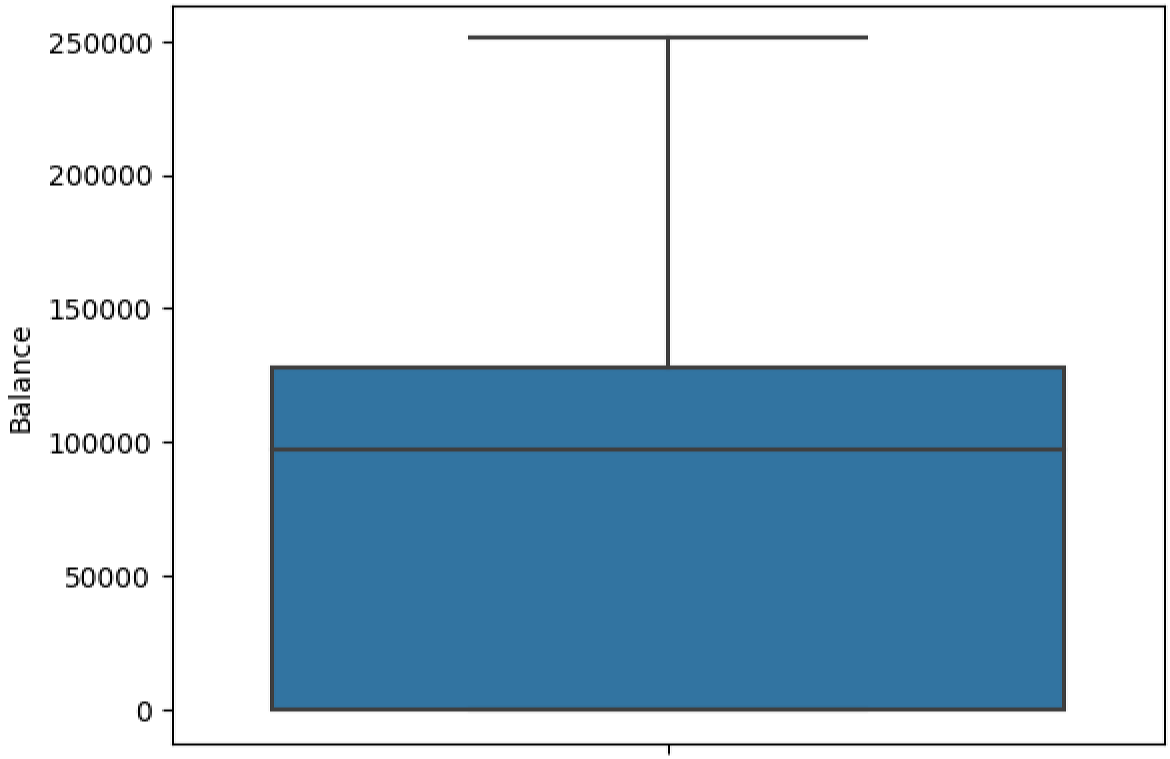




Out[13]:
```
{'missing': 0,
 'min': 0,
 'max': 10,
 'var': 8.364672627262866,
 'std': 2.892174377049708,
 'p1': 0.0,
 'p10': 1.0,
 'p25': 3.0,
 'p50': 5.0,
 'p75': 7.0,
 'p99': 10.0}
```

In [14]:
```
numerical(data=churn, var="Balance")
```

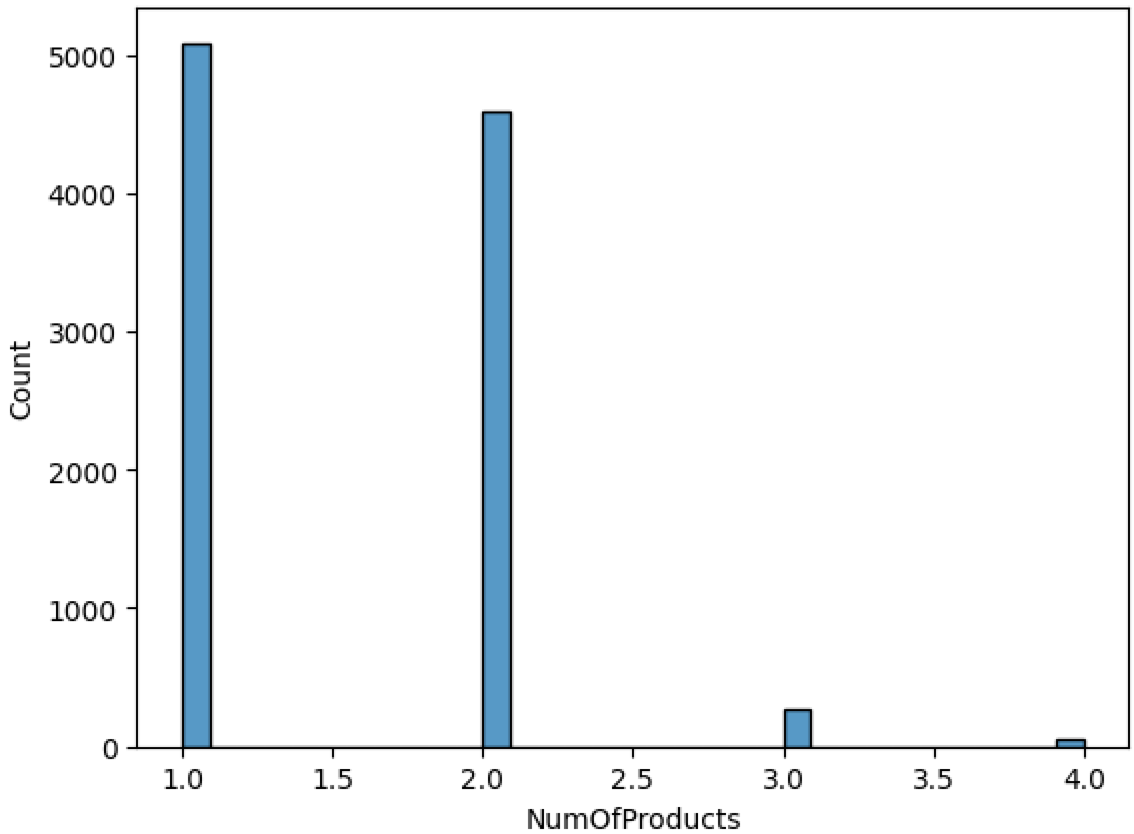


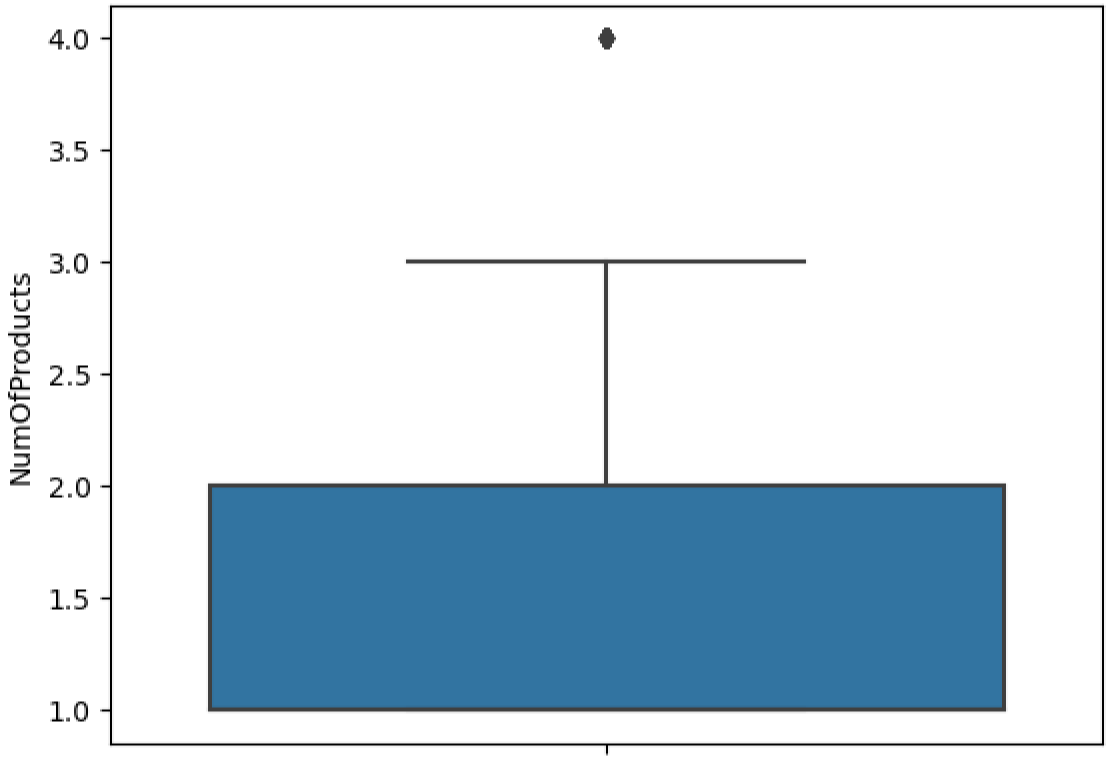


Out[14]:
```
{'missing': 0,
 'min': 0.0,
 'max': 250898.09,
 'var': 3893436175.9907765,
 'std': 62397.40520238623,
 'p1': 0.0,
 'p10': 0.0,
 'p25': 0.0,
 'p50': 97198.54000000001,
 'p75': 127644.24,
 'p99': 185967.98540000003}
```

In [15]:

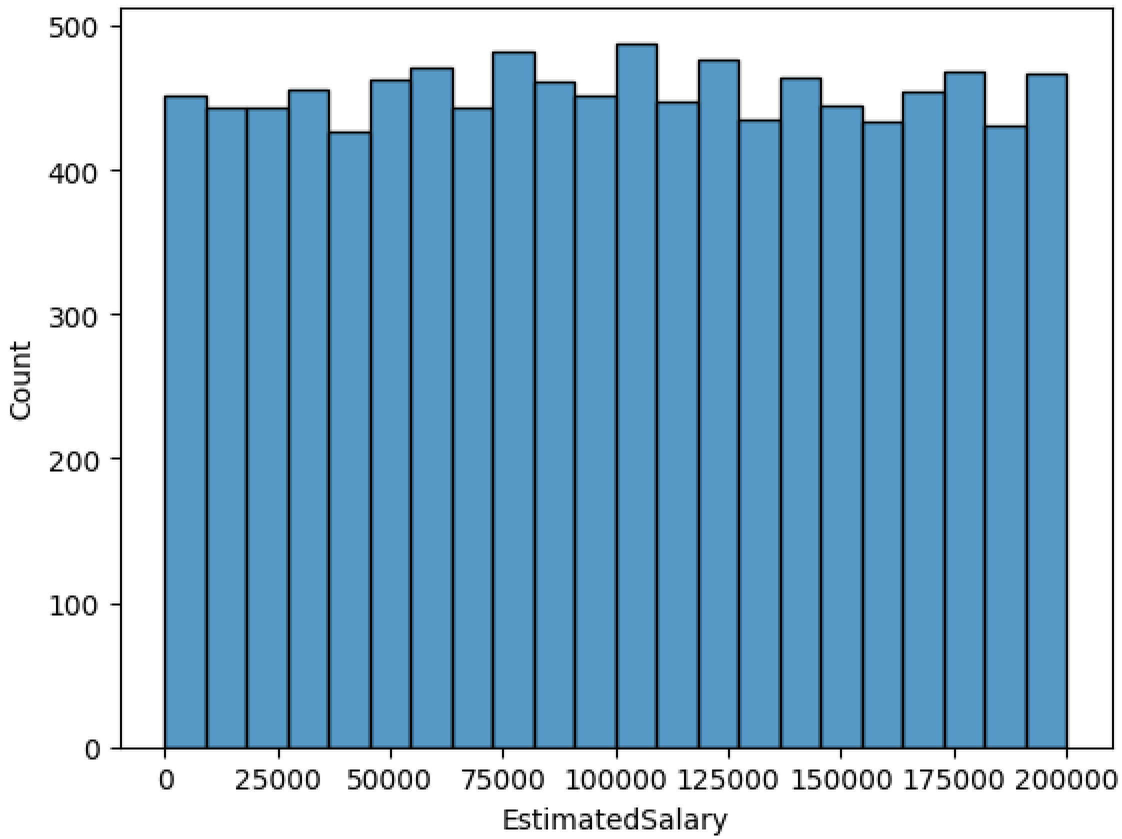
```
numerical(data=churn, var="NumOfProducts")
```

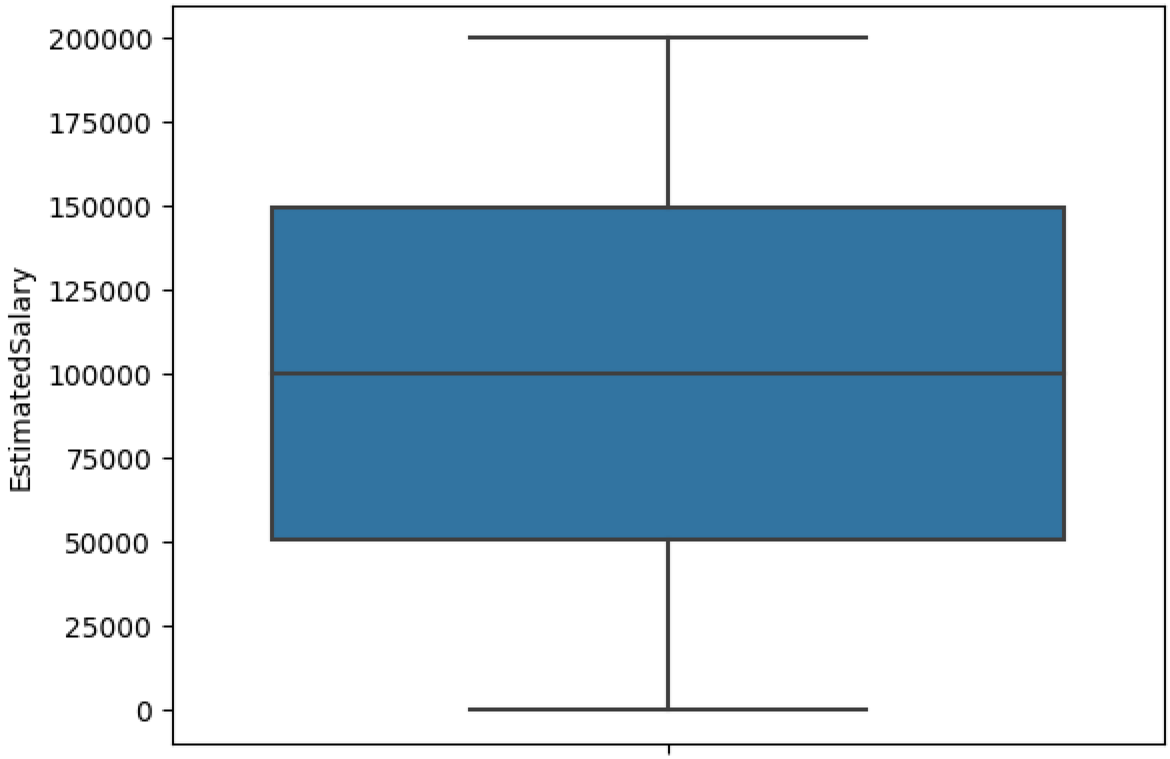




Out[15]:
```
{'missing': 0,
 'min': 1,
 'max': 4,
 'var': 0.3383217921792214,
 'std': 0.5816543579989936,
 'p1': 1.0,
 'p10': 1.0,
 'p25': 1.0,
 'p50': 1.0,
 'p75': 2.0,
 'p99': 3.0}
```

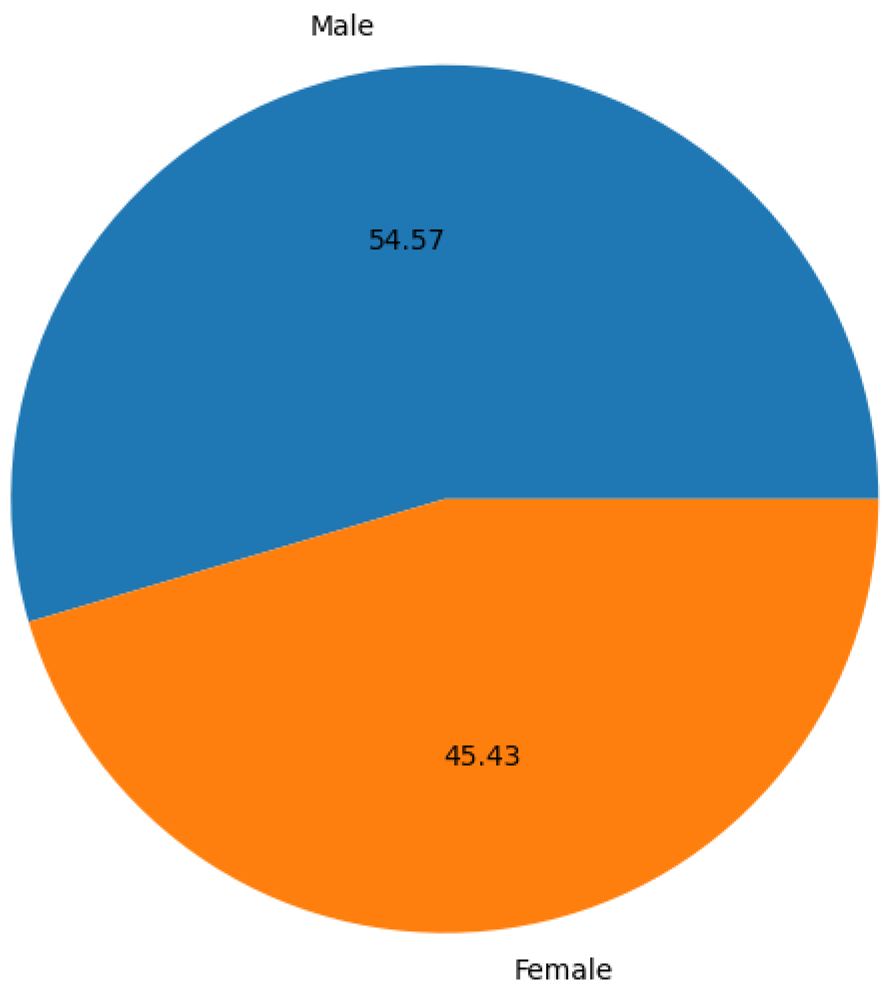
In [16]:
```
numerical(data=churn, var="EstimatedSalary")
```





Out[16]:
```
{'missing': 0,
 'min': 11.58,
 'max': 199992.48,
 'var': 3307456784.134519,
 'std': 57510.49281769822,
 'p1': 1842.8253000000004,
 'p10': 20273.58,
 'p25': 51002.11,
 'p50': 100193.915,
 'p75': 149388.2475,
 'p99': 198069.7345}
```

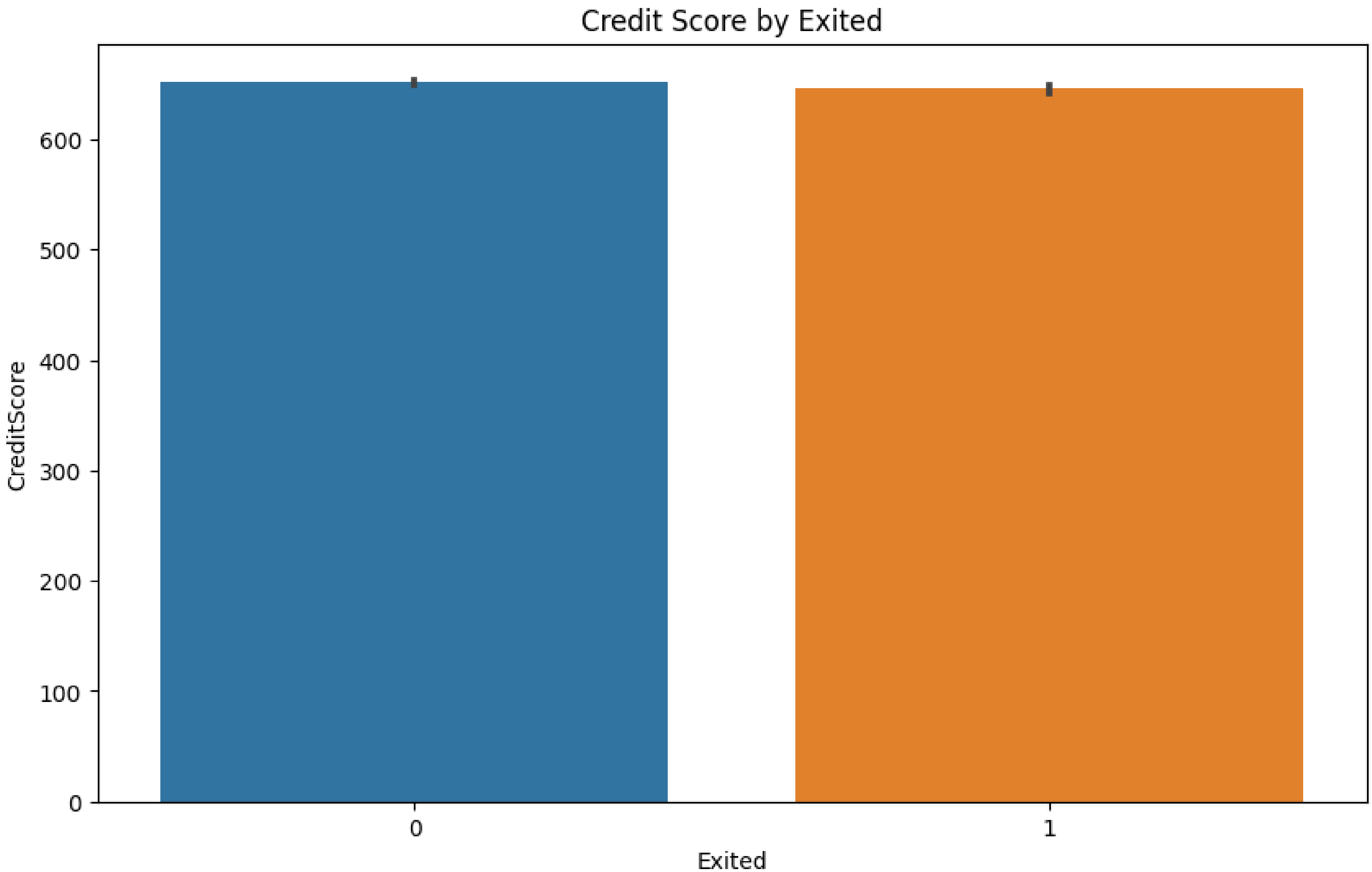
In [17]:
```
plt.figure(figsize=(7,8))
plt.pie(churn['Gender'].value_counts(),labels=['Male','Female'],autopct='%0.2f')
plt.show()
```

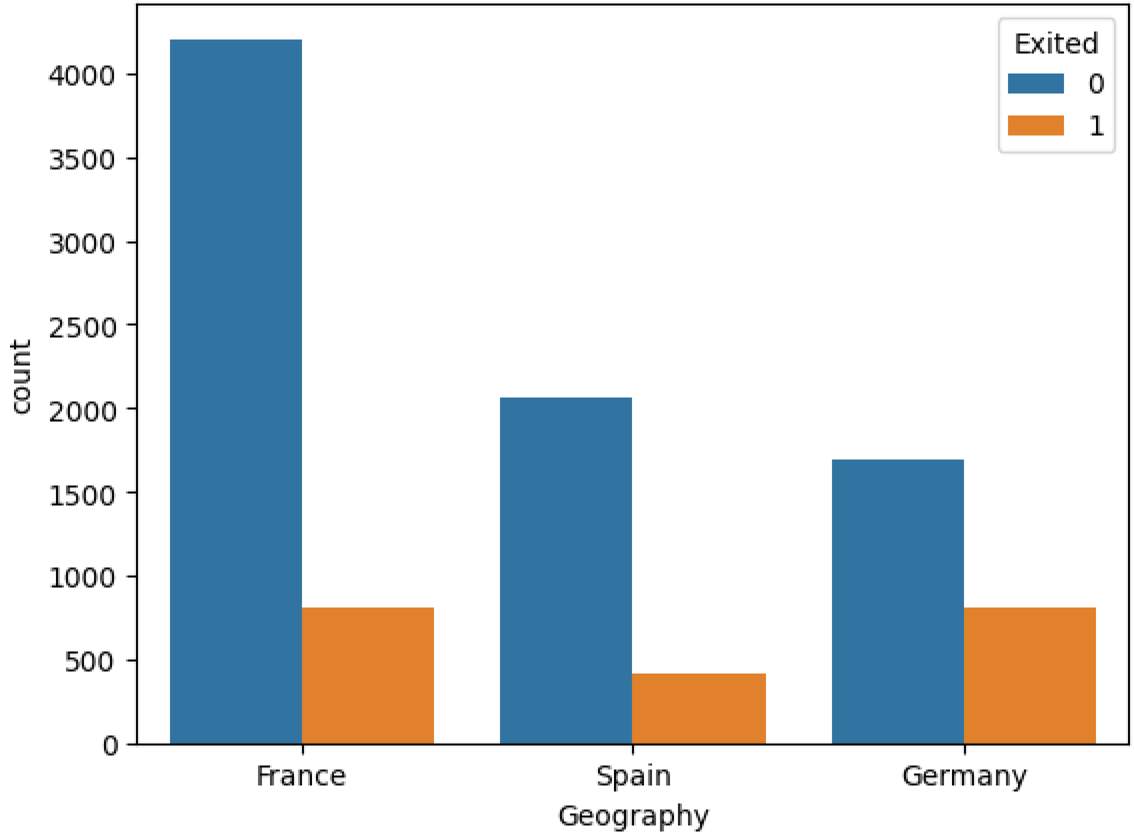


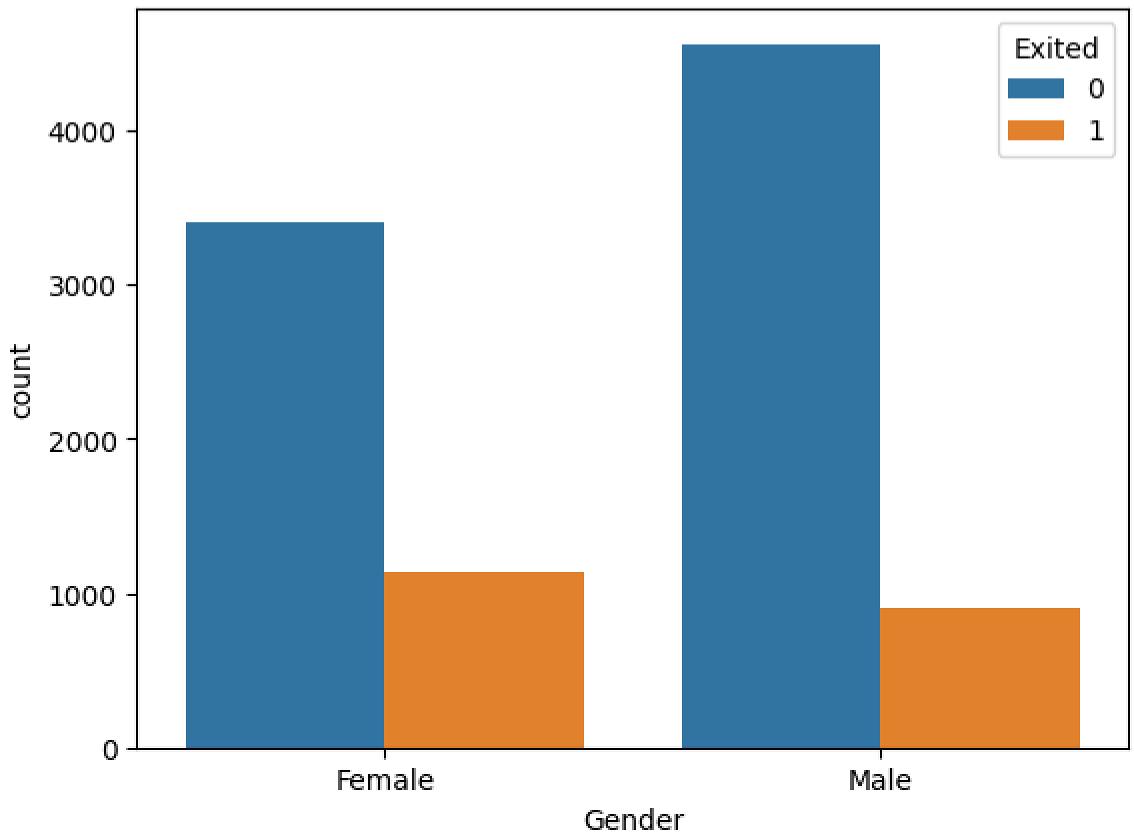
In [18]:
```
churn.columns
```

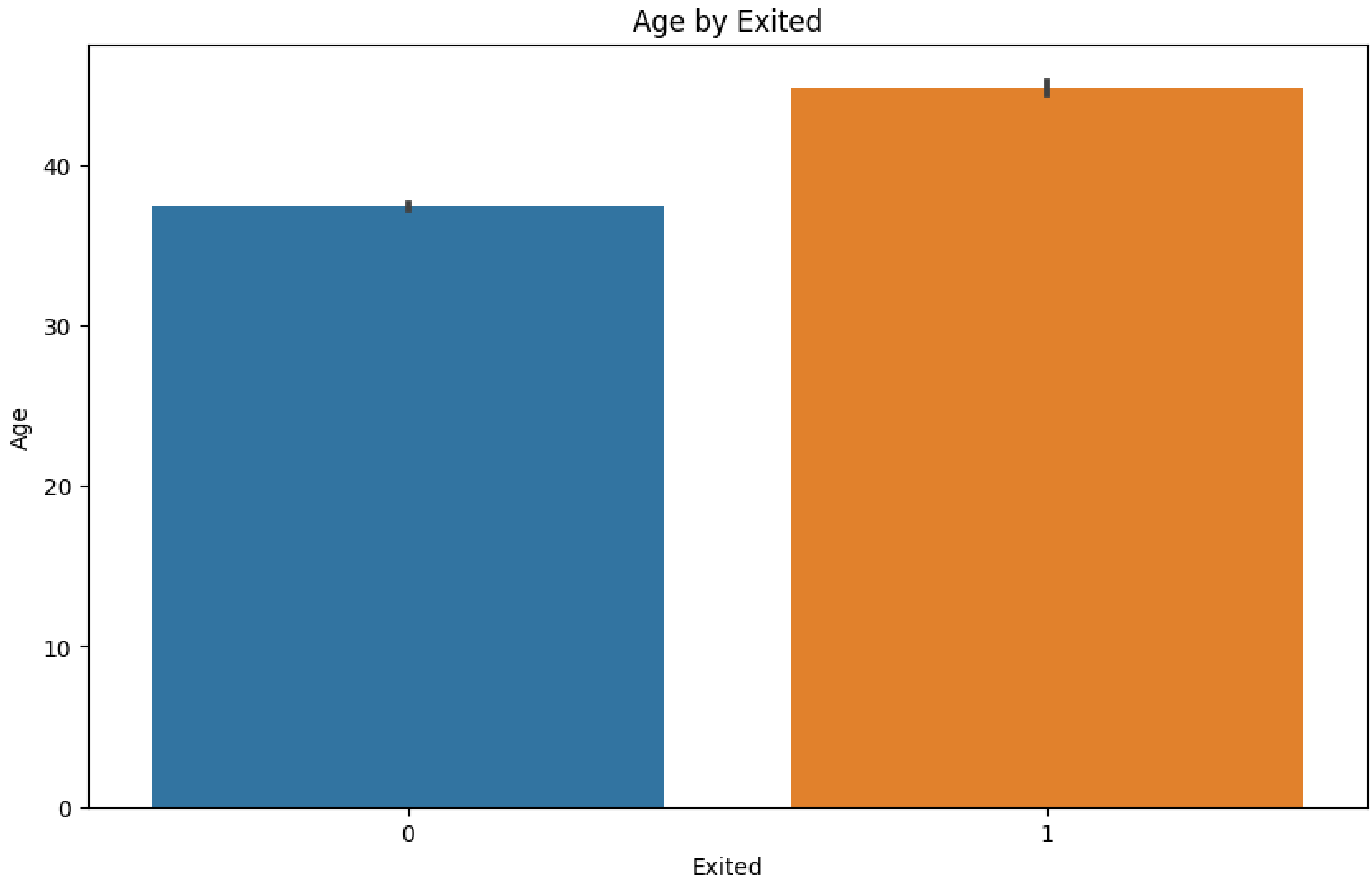
Out[18]:
```
Index(['CreditScore', 'Geography', 'Gender', 'Age', 'Tenure', 'Balance',
       'NumOfProducts', 'HasCrCard', 'IsActiveMember', 'EstimatedSalary',
       'Exited'],
      dtype='object')
```

In [19]:
```
# Create a bar graph for CreditScore
plt.figure(figsize=(10, 6))
sns.barplot(data=churn, x='Exited', y='CreditScore')
plt.title('Credit Score by Exited')
plt.show()

# Create a count graph for Geograpy
sns.countplot(x='Geography',hue='Exited',data=churn)
plt.show()


# Create a count graph for Gender
sns.countplot(x='Gender',hue='Exited',data=churn)
plt.show()


# Create a bar graph for Age
plt.figure(figsize=(10, 6))
sns.barplot(data=churn, x='Exited', y='Age')
plt.title('Age by Exited')
plt.show()

# Create a bar graph for Tenure
plt.figure(figsize=(10, 6))
sns.barplot(data=churn, x='Exited', y='Tenure')
plt.title('Tenure by Exited')
plt.show()

# Create a bar graph for Balance
plt.figure(figsize=(10, 6))
sns.barplot(data=churn, x='Exited', y='Balance')
plt.title('Balance by Exited')
plt.show()

# Create a bar graph for NumOfProducts
plt.figure(figsize=(10, 6))
sns.barplot(data=churn, x='Exited', y='NumOfProducts')
plt.title('NumOfProducts by Exited')
plt.show()
```

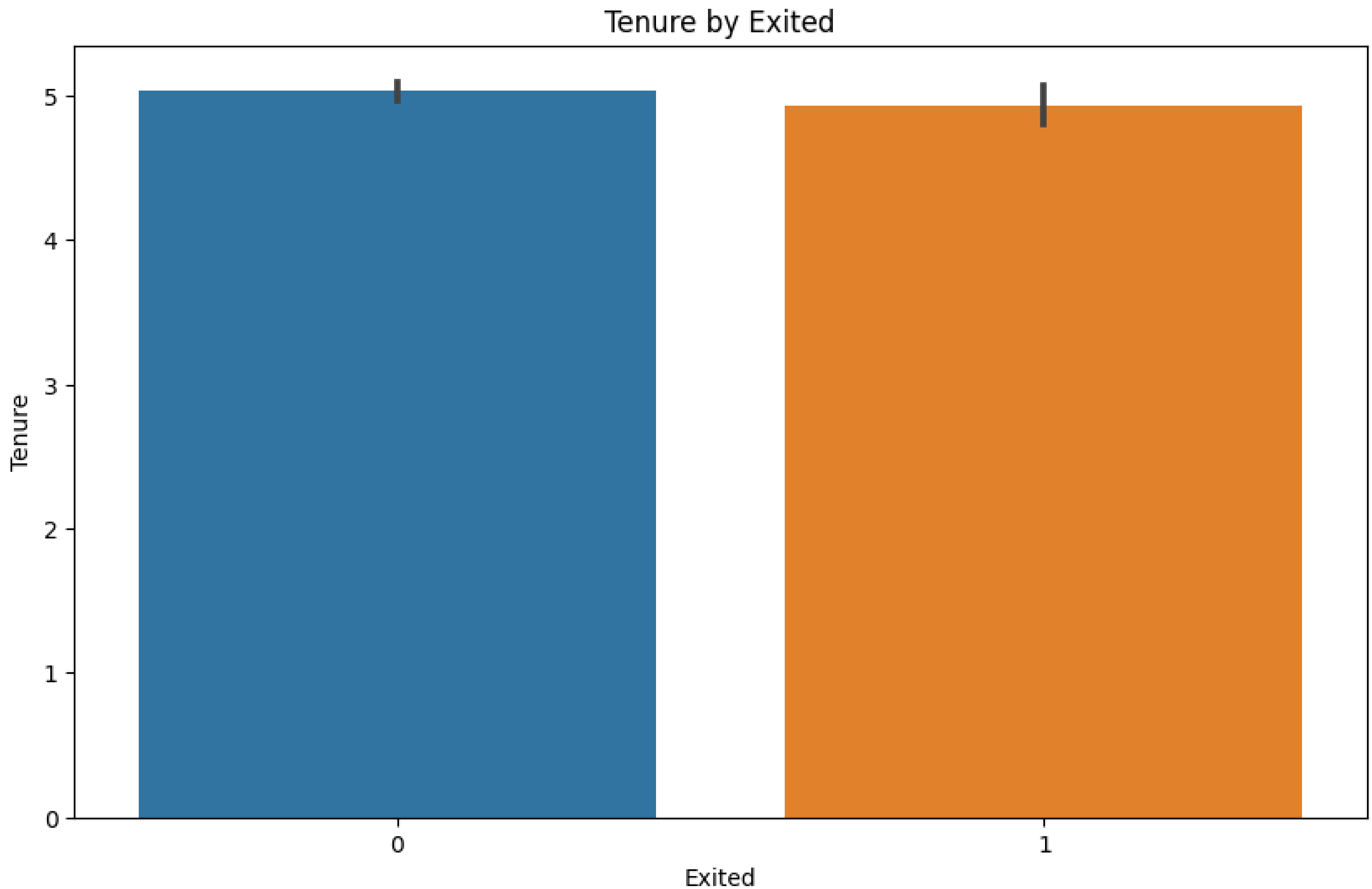

Credit Score by Exited
CreditScore
0
100
200
300
400
500
600
0
1
Exited

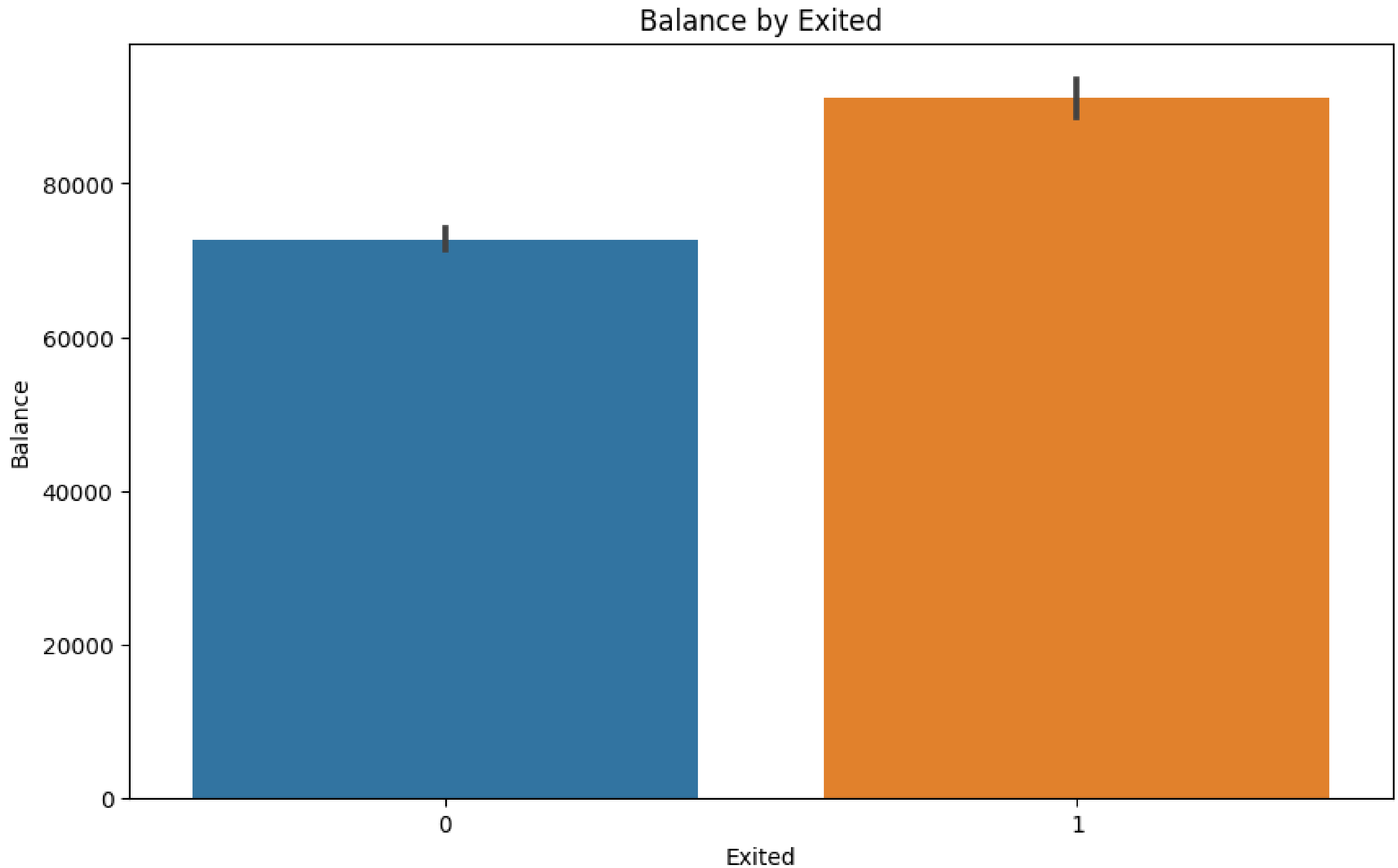

Exited
0
1
4000
3500
3000
2500
2000
1500
1000
500
0
count
France
Spain
Germany
Geography

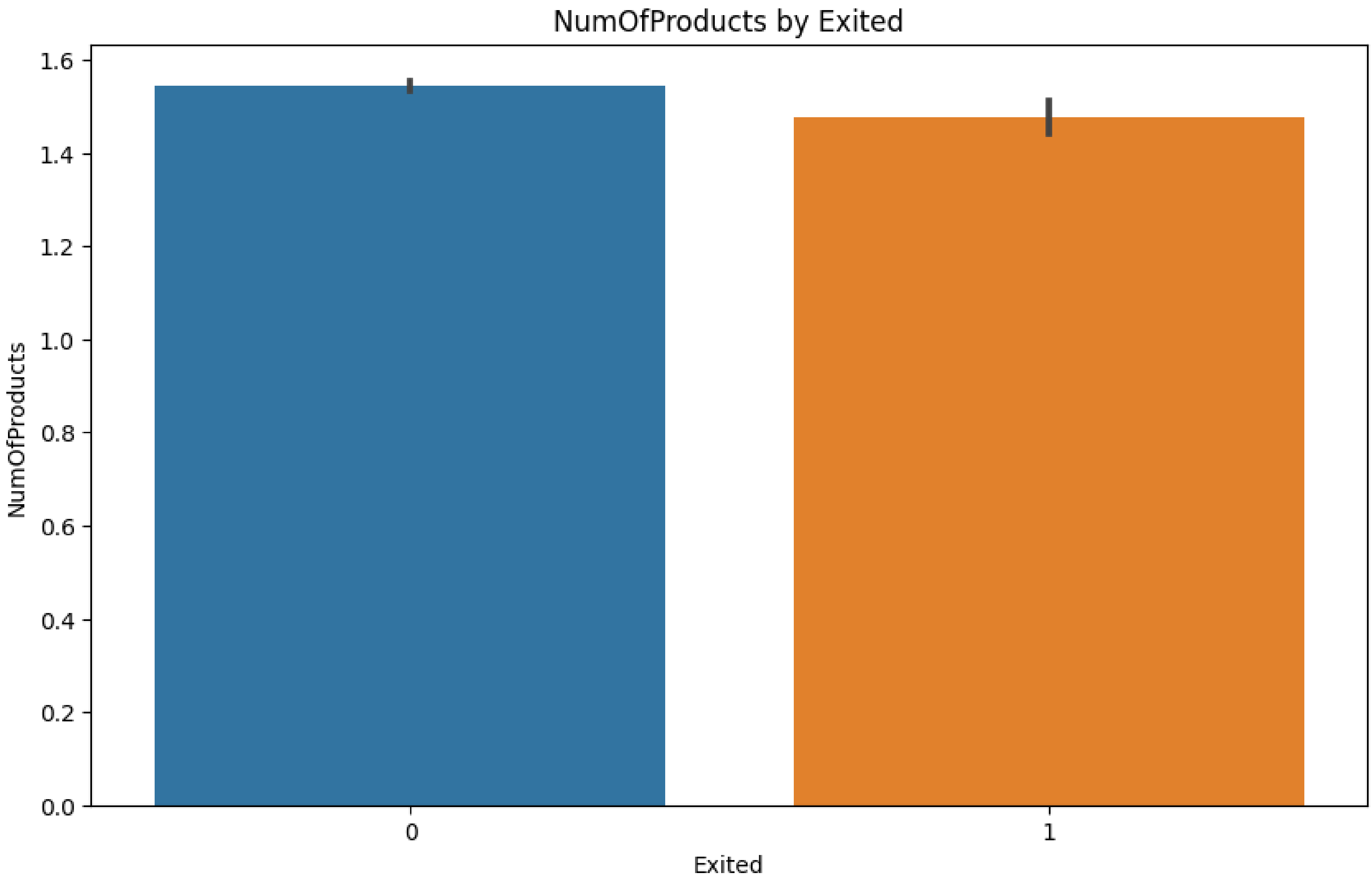

Exited
0
1
4000
3000
2000
1000
0
count
Female
Male
Gender



Age by Exited
40
30
20
10
0
Age
0
1
Exited


Tenure by Exited
Tenure
Exited
0
1
2
3
4
5
0
1


Balance by Exited
80000
60000
40000
20000
0
Balance
0
1
Exited


NumOfProducts by Exited
NumOfProducts
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
1
Exited

One-hot encode categorical variables

In [20]:
```
churn = pd.get_dummies(churn, columns=['Gender','Geography'])
order = ['CreditScore', 'Age', 'Tenure', 'Balance', 'NumOfProducts',
       'HasCrCard', 'IsActiveMember', 'EstimatedSalary', 'Gender_Female',
       'Gender_Male', 'Geography_France', 'Geography_Germany', 'Geography_Spain', 'Exited']
churn = churn[order]
churn.head()
```

Out[20]:

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

Correlation Heatmap

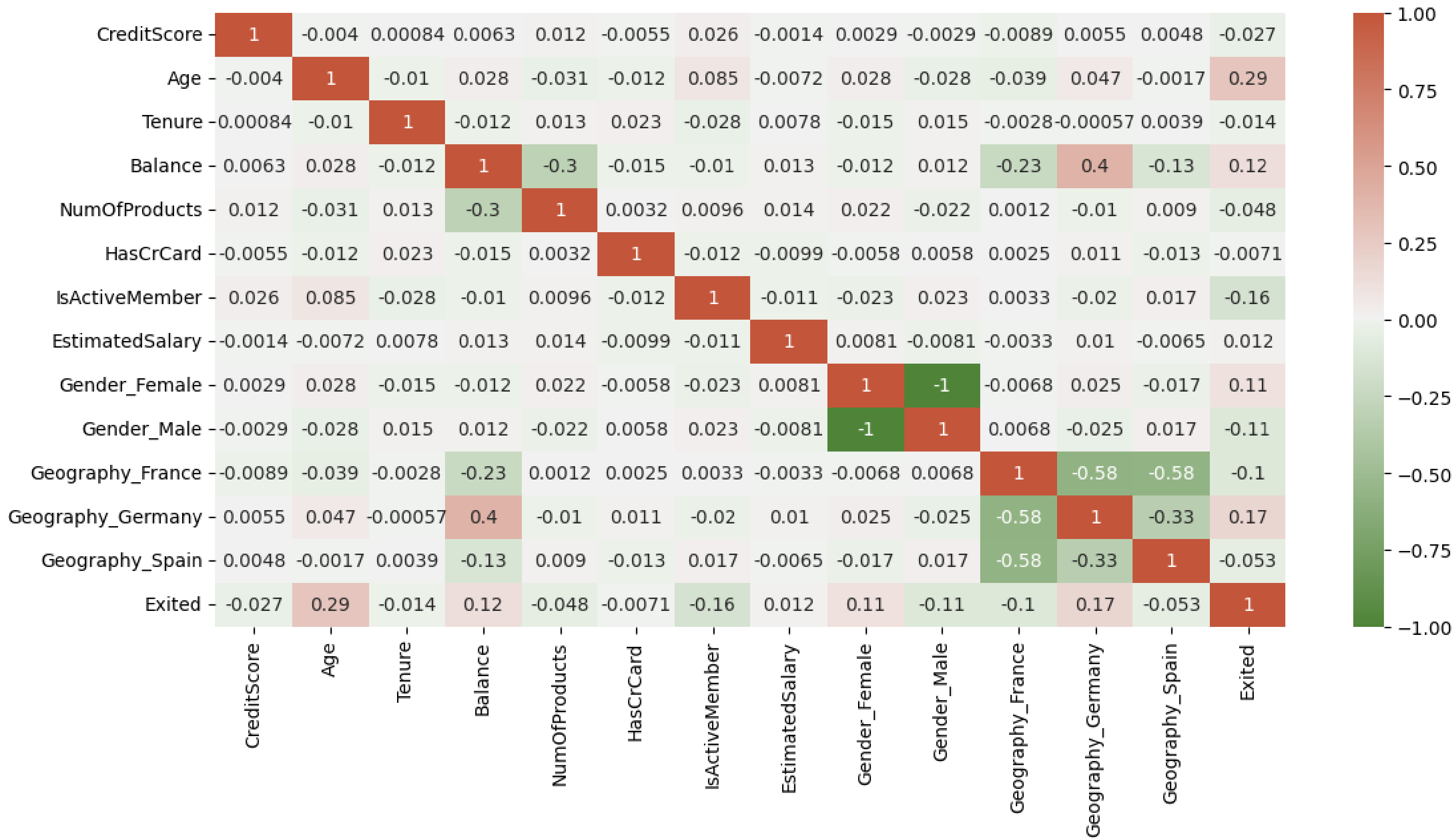
In [21]:
```
churn.corr()
```

Out[21]:

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary
CreditScore	1.000000	-0.003965	0.000842	0.006268	0.012238	-0.005458	0.025651	-0.001384
Age	-0.003965	1.000000	-0.009997	0.028308	-0.030680	-0.011721	0.085472	-0.007201
Tenure	0.000842	-0.009997	1.000000	-0.012254	0.013444	0.022583	-0.028362	0.007784
Balance	0.006268	0.028308	-0.012254	1.000000	-0.304180	-0.014858	-0.010084	0.012797
NumOfProducts	0.012238	-0.030680	0.013444	-0.304180	1.000000	0.003183	0.009612	0.014204
HasCrCard	-0.005458	-0.011721	0.022583	-0.014858	0.003183	1.000000	-0.011866	-0.009933
IsActiveMember	0.025651	0.085472	-0.028362	-0.010084	0.009612	-0.011866	1.000000	-0.011421
EstimatedSalary	-0.001384	-0.007201	0.007784	0.012797	0.014204	-0.009933	-0.011421	1.000000
Gender_Female	0.002857	0.027544	-0.014733	-0.012087	0.021859	-0.005766	-0.022544	0.008112
Gender_Male	-0.002857	-0.027544	0.014733	0.012087	-0.021859	0.005766	0.022544	-0.008112

In [22]:

```
plt.figure(figsize=(13,6))
# Create a diverging colormap
cmap = sns.diverging_palette(120, 20, as_cmap=True)

corr = churn.corr()
sns.heatmap(corr, annot=True, cmap=cmap, vmin=-1, vmax=1)
plt.show()
```



In [23]:
```
X = churn.drop(['Exited'], axis=1)
y = churn['Exited']
```

In [24]:
```
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [25]:
```
# Standardize the features (important for some models like Logistic Regression)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Logistic Regression model

In [26]:
```
logistic_model = LogisticRegression()
logistic_model.fit(X_train_scaled, y_train)
```

Out[26]:
```
▾ LogisticRegression

LogisticRegression()
```

In [27]:
```
# Make predictions on the test set
y_pred = logistic_model.predict(X_test_scaled)
```

In [28]:
```
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matx = confusion_matrix(y_test, y_pred)
report = classification_report(y_test, y_pred)
```

In [29]:

```
print(f"Logistic Regression Accuracy Score: {accuracy:.2f}")

print(f'Confusion Matrix:\n{conf_matx}')
# print(f'Accuracy: {accuracy}')
print(f'Classification Report:\n{report}')
```

```
Logistic Regression Accuracy Score: 0.81
Confusion Matrix:
[[1543   64]
 [ 314   79]]
Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.96      0.89      1607
           1       0.55      0.20      0.29       393

    accuracy                           0.81      2000
   macro avg       0.69      0.58      0.59      2000
weighted avg       0.78      0.81      0.77      2000
```

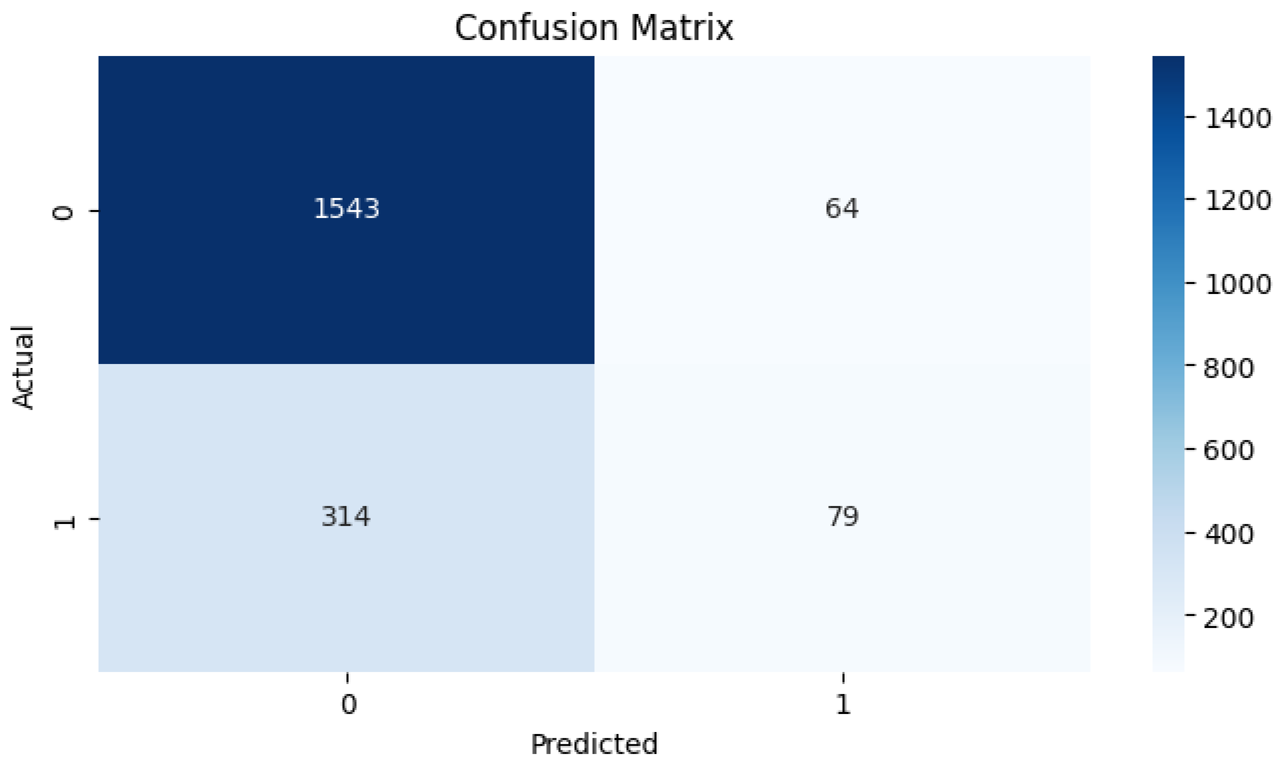
In [30]:

```
#Check the test score and train score to the Logistic Regression algorithm
print(f'The Test accuracy: {logistic_model.score(X_test_scaled,y_test)*100:.2f}')

#Train score for the data
print(f'The Train accuracy: {logistic_model.score(X_train_scaled, y_train)*100:.2f}')
```

```
The Test accuracy: 81.10
The Train accuracy: 81.14
```

In [31]:

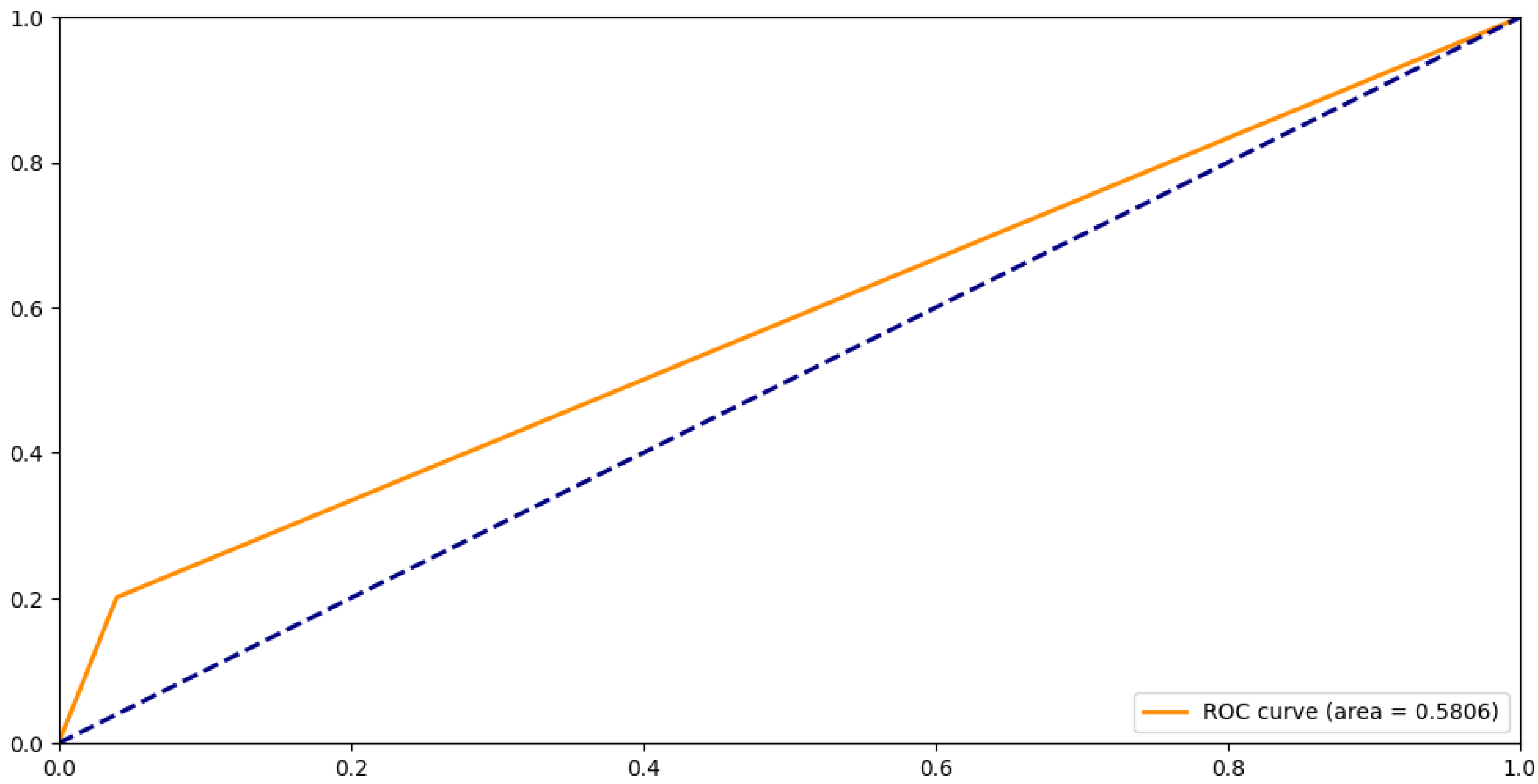
```
conf_matx = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 4))
sns.heatmap(conf_matx, annot=True, fmt="d", cmap="Blues")
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```



In [32]:

```
from sklearn.metrics import accuracy_score, confusion_matrix, roc_curve, roc_auc_score, auc
```

In [33]:
```
fpr, tpr, _ = roc_curve(y_test, y_pred)
mnb_cv_roc_auc = auc(fpr, tpr)

plt.figure(figsize=(12,6))
plt.plot(fpr, tpr, color="darkorange", lw=2, label=f"ROC curve (area = {mnb_cv_roc_auc :.4f})")
plt.plot([0, 1], [0, 1], color="navy", lw=2, linestyle="--")
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.legend(loc="lower right")
plt.show()
```



Random Forest Model

In [34]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
```

In [35]:
```
rf1=RandomForestClassifier(n_estimators=500, min_samples_split=30, min_samples_leaf=5)
rf1.fit(X_train, y_train)
```

Out[35]:
```
RandomForestClassifier
RandomForestClassifier(min_samples_leaf=5, min_samples_split=30,
                       n_estimators=500)
```

In [36]:
```
rf1.score(X_train, y_train)

```

Out[36]: 0.882125

In [37]:
```
rf1.score(X_test, y_test)
```

Out[37]: 0.8625