CustomerChurn

June 30, 2021

1 Predicting Customer Churn

1.1 *Installing/Importing all the required lib*

[1]:
```
# loading necessary libraries
import pandas as pd
import numpy as np
import seaborn as sns
import random
import matplotlib.pyplot as plt
from matplotlib import pyplot
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import confusion_matrix, classification_report, f1_score,␣
↪precision_score, recall_score, roc_auc_score, roc_curve
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from catboost import CatBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from lightgbm import LGBMClassifier
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from sklearn.metrics import accuracy_score,recall_score
from xgboost import XGBClassifier
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score, GridSearchCV

import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)
warnings.filterwarnings("ignore", category=FutureWarning)
warnings.filterwarnings("ignore", category=UserWarning)

%config InlineBackend.figure_format = 'retina'

# to display all columns and rows:
```

```
pd.set_option('display.max_columns', None); pd.set_option('display.max_rows',␣
↪None);
```

- ***Installing remaining necessary using pip***

[2]:
```
# import sys
# !{sys.executable} -m pip install -U seaborn
# !{sys.executable} -m pip install -U nbconvert
# !{sys.executable} -m pip install -U ipypublish
# !{sys.executable} -m pip install -U LaTeX
# !{sys.executable} -m pip install -U pandoc
# !{sys.executable} -m pip install -U nb_pdf_template
```

1.2 *Loading Data into Pandas Dataframe*

[3]:
```
raw_data = pd.read_csv("Churn_Prediction.csv",index_col=0)
```

1.3 *Analyzing Data*

[4]:
```
raw_data.head(10)
```

[4]:
```
           CustomerId   Surname  CreditScore Geography  Gender  Age  Tenure  \
RowNumber
1            15634602  Hargrave          619    France  Female   42       2
2            15647311      Hill          608     Spain  Female   41       1
3            15619304      Onio          502    France  Female   42       8
4            15701354      Boni          699    France  Female   39       1
5            15737888  Mitchell          850     Spain  Female   43       2
6            15574012       Chu          645     Spain    Male   44       8
7            15592531  Bartlett          822    France    Male   50       7
8            15656148    Obinna          376   Germany  Female   29       4
9            15792365        He          501    France    Male   44       4
10           15592389        H?          684    France    Male   27       2

             Balance  NumOfProducts  HasCrCard  IsActiveMember  \
RowNumber
1               0.00              1          1               1
2           83807.86              1          0               1
3          159660.80              3          1               0
4               0.00              2          0               0
5          125510.82              1          1               1
6          113755.78              2          1               0
7               0.00              2          1               1
8          115046.74              4          1               0
9          142051.07              2          0               1
10         134603.88              1          1               1
```

```
           EstimatedSalary  Exited
RowNumber
1                101348.88       1
2                112542.58       0
3                113931.57       1
4                 93826.63       0
5                 79084.10       0
6                149756.71       1
7                 10062.80       0
8                119346.88       1
9                 74940.50       0
10                71725.73       0
```

- ***EDA, Checking DType of columns and Null values***

[5]:
```python
# Missing Observation Analysis
raw_data.isnull().sum()
```

[5]:
```
CustomerId         0
Surname            0
CreditScore        0
Geography          0
Gender             0
Age                0
Tenure             0
Balance            0
NumOfProducts      0
HasCrCard          0
IsActiveMember     0
EstimatedSalary    0
Exited             0
dtype: int64
```

[6]:
```python
# To get summary statistic of a data set you can use describe()
raw_data.describe([0.05,0.25,0.50,0.75,0.90,0.95,0.99])
```

[6]:
```
         CustomerId   CreditScore           Age        Tenure        Balance  \
count  1.000000e+04  10000.000000  10000.000000  10000.000000   10000.000000
mean   1.569094e+07    650.528800     38.921800      5.012800   76485.889288
std    7.193619e+04     96.653299     10.487806      2.892174   62397.405202
min    1.556570e+07    350.000000     18.000000      0.000000       0.000000
5%     1.557882e+07    489.000000     25.000000      1.000000       0.000000
25%    1.562853e+07    584.000000     32.000000      3.000000       0.000000
50%    1.569074e+07    652.000000     37.000000      5.000000   97198.540000
75%    1.575323e+07    718.000000     44.000000      7.000000  127644.240000
90%    1.579083e+07    778.000000     53.000000      9.000000  149244.792000
95%    1.580303e+07    812.000000     60.000000      9.000000  162711.669000
99%    1.581311e+07    850.000000     72.000000     10.000000  185967.985400
```

```
max    1.581569e+07    850.000000     92.000000     10.000000  250898.090000

       NumOfProducts    HasCrCard  IsActiveMember  EstimatedSalary  \
count   10000.000000  10000.00000    10000.000000     10000.000000
mean        1.530200      0.70550        0.515100    100090.239881
std         0.581654      0.45584        0.499797     57510.492818
min         1.000000      0.00000        0.000000        11.580000
5%          1.000000      0.00000        0.000000      9851.818500
25%         1.000000      0.00000        0.000000     51002.110000
50%         1.000000      1.00000        1.000000    100193.915000
75%         2.000000      1.00000        1.000000    149388.247500
90%         2.000000      1.00000        1.000000    179674.704000
95%         2.000000      1.00000        1.000000    190155.375500
99%         3.000000      1.00000        1.000000    198069.734500
max         4.000000      1.00000        1.000000    199992.480000

             Exited
count  10000.000000
mean       0.203700
std        0.402769
min        0.000000
5%         0.000000
25%        0.000000
50%        0.000000
75%        0.000000
90%        1.000000
95%        1.000000
99%        1.000000
max        1.000000
```

[7]:
```
# Using info() to check dtypes of column
raw_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10000 entries, 1 to 10000
Data columns (total 13 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   CustomerId       10000 non-null  int64
 1   Surname          10000 non-null  object
 2   CreditScore      10000 non-null  int64
 3   Geography        10000 non-null  object
 4   Gender           10000 non-null  object
 5   Age              10000 non-null  int64
 6   Tenure           10000 non-null  int64
 7   Balance          10000 non-null  float64
 8   NumOfProducts    10000 non-null  int64
 9   HasCrCard        10000 non-null  int64
```

```
 10  IsActiveMember   10000 non-null  int64
 11  EstimatedSalary  10000 non-null  float64
 12  Exited           10000 non-null  int64
dtypes: float64(2), int64(8), object(3)
memory usage: 1.1+ MB
```

[8]:
```
# Dependent Variable - Exited
# Lets check the frequency of the two classes(0 & 1) of Exited column(dependent␣
↪variable)
raw_data["Exited"].value_counts()
```

[8]:
```
0    7963
1    2037
Name: Exited, dtype: int64
```

[9]:
```
print("Number of unique values in each column:=")
for col in raw_data.columns:
    print("{}: {}".format(col,raw_data[col].nunique()))
```

```
Number of unique values in each column:=
CustomerId: 10000
Surname: 2932
CreditScore: 460
Geography: 3
Gender: 2
Age: 70
Tenure: 11
Balance: 6382
NumOfProducts: 4
HasCrCard: 2
IsActiveMember: 2
EstimatedSalary: 9999
Exited: 2
```

[10]:
```
# User defined function for code reuse. This function can return categorical␣
↪and numerical column names by just passing df and type needed

def col_name(df,col_type):
    if col_type == 'cat':
        return [col for col in df.columns if col in "O" # "O" Column having␣
↪DType "Object"
                                or df[col].nunique() <= 11 # as found out above␣
↪anything above 11 number of unique valuesshould be treated as categorical␣
↪variable
                                and col not in "Exited"] # "Exited" is our dependant␣
↪variable
    elif col_type == 'num':
```

```
    return [col for col in df.columns if df[col].dtype != "object" # No␣
↪"Oject" type
                    and df[col].nunique() > 11 # As found above anything␣
↪over 11 number of unique values should be treated as numerical variable
                    and col not in "CustomerId"] # "CustomerId" is not a␣
↪numerical variable in this situation
```

[11]:
```
# Categorical Variables
cat_var = col_name(raw_data,"cat") #

cat_var
```

[11]:
```
['Geography',
 'Gender',
 'Tenure',
 'NumOfProducts',
 'HasCrCard',
 'IsActiveMember']
```

[12]:
```
# Numeric Variables
num_var = col_name(raw_data,"num")

num_var
```

[12]:
```
['CreditScore', 'Age', 'Balance', 'EstimatedSalary']
```

- ***This is just for example purpose that you can divide your into leavers and non_leavers for doing some analysis***

[13]:
```
# Customers leaving the bank
leavers = raw_data.loc[raw_data["Exited"]==1]

# Customers who did not leave the bank
non_leavers = raw_data.loc[raw_data["Exited"]==0]
```

[14]:
```
leavers["NumOfProducts"].value_counts().sort_values()
```

[14]:
```
4      60
3     220
2     348
1    1409
Name: NumOfProducts, dtype: int64
```

[15]:
```
non_leavers["NumOfProducts"].value_counts().sort_values()
```

[15]:
```
3      46
1    3675
2    4242
```

```
Name: NumOfProducts, dtype: int64
```

[16]:
```python
# Checking the credit score for customer left/leaving
leavers["CreditScore"].describe([0.05,0.25,0.50,0.75,0.90,0.95,0.99])
```

[16]:
```
count    2037.000000
mean      645.351497
std       100.321503
min       350.000000
5%        479.000000
25%       578.000000
50%       646.000000
75%       716.000000
90%       776.400000
95%       812.200000
99%       850.000000
max       850.000000
Name: CreditScore, dtype: float64
```

[17]:
```python
# Checking the Age for customer left/leaving
leavers["Age"].describe([0.05,0.25,0.50,0.75,0.90,0.95,0.99])
```

[17]:
```
count    2037.000000
mean       44.837997
std         9.761562
min        18.000000
5%         29.000000
25%        38.000000
50%        45.000000
75%        51.000000
90%        58.000000
95%        61.000000
99%        68.000000
max        84.000000
Name: Age, dtype: float64
```

[18]:
```python
# Checking the Age for customer not left/leaving
non_leavers["Age"].describe([0.05,0.25,0.50,0.75,0.90,0.95,0.99])
```

[18]:
```
count    7963.000000
mean       37.408389
std        10.125363
min        18.000000
5%         24.000000
25%        31.000000
50%        36.000000
75%        41.000000
```

```
90%         49.000000
95%         59.000000
99%         73.000000
max         92.000000
Name: Age, dtype: float64
```

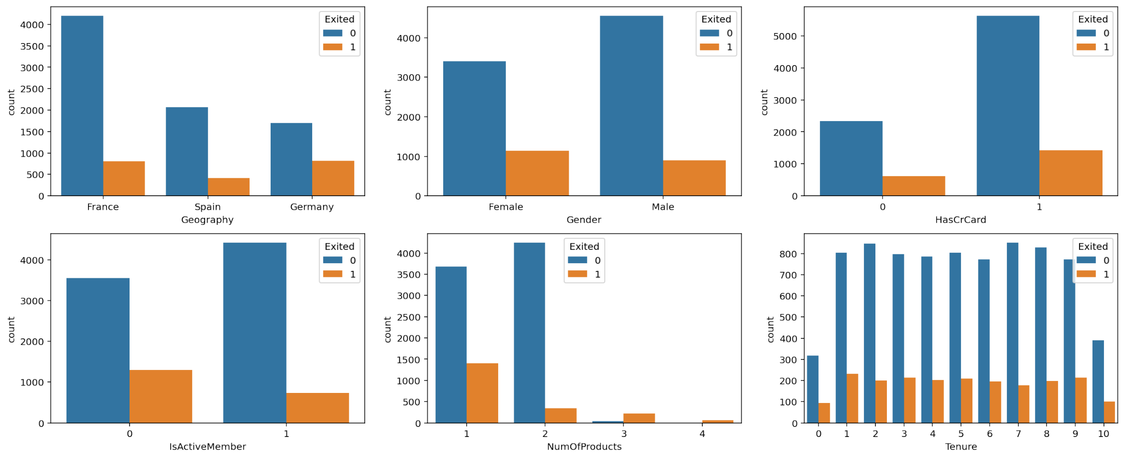
[19]:
```
# Checking the credit score for customer not left/leaving
raw_data[raw_data['Exited']==0]["CreditScore"].describe([0.05,0.25,0.50,0.75,0.
↪90,0.95,0.99])
```

[19]:
```
count    7963.000000
mean      651.853196
std        95.653837
min       405.000000
5%        492.000000
25%       585.000000
50%       653.000000
75%       718.000000
90%       778.000000
95%       812.000000
99%       850.000000
max       850.000000
Name: CreditScore, dtype: float64
```

1.4 *Data Visualization*

[20]:
```
def show_dependent_variable(raw_data):
    fig, axarr = plt.subplots(2, 3, figsize=(20, 8))
    sns.countplot(x = 'Geography', hue = 'Exited',data = raw_data, ax =␣
↪axarr[0][0])
    sns.countplot(x = 'Gender', hue = 'Exited',data = raw_data, ax =␣
↪axarr[0][1])
    sns.countplot(x = 'HasCrCard', hue = 'Exited',data = raw_data, ax =␣
↪axarr[0][2])
    sns.countplot(x = 'IsActiveMember', hue = 'Exited',data = raw_data, ax =␣
↪axarr[1][0])
    sns.countplot(x = 'NumOfProducts', hue = 'Exited',data = raw_data, ax =␣
↪axarr[1][1])
    sns.countplot(x = 'Tenure', hue = 'Exited',data = raw_data, ax =␣
↪axarr[1][2])
```

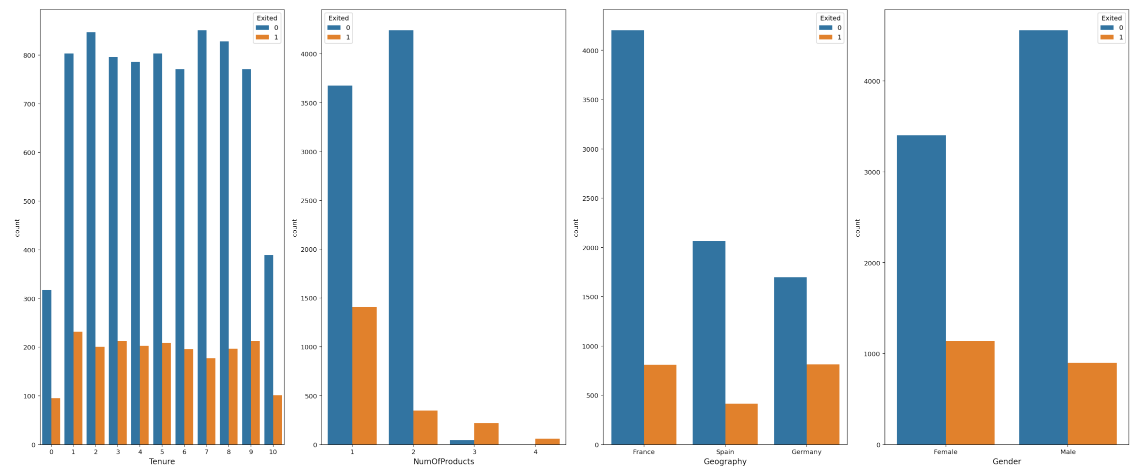
[21]:
```
show_dependent_variable(raw_data)
```



- ***You can acheive same outcome by using User define function. Func 'plot' to plot columns Vs Exited***

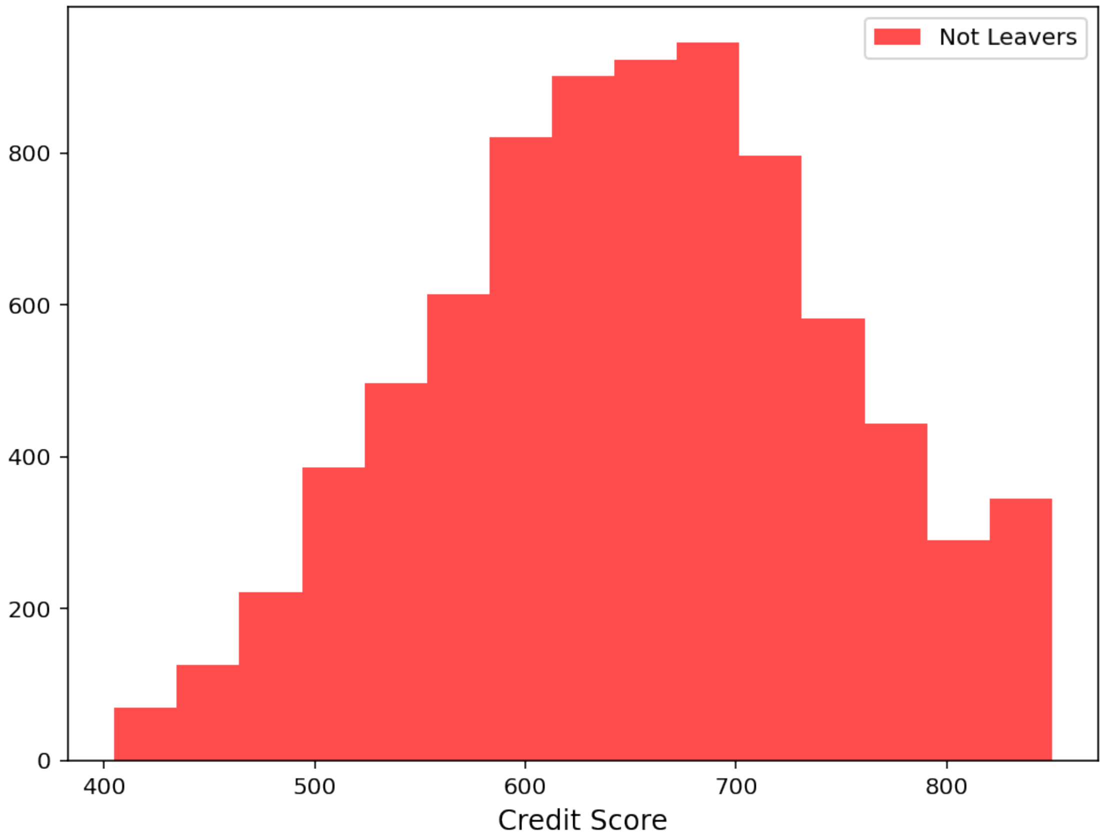
[22]:
```
# Defining a function to re-use the code for plotting

def plot(df,cols):
    n = n=len(cols) # n will be used to calculate number of columns(subplot)
    fig, axarr = plt.subplots(1,n, figsize=(n*6,10))

    for i,col in enumerate(cols):
        plt.sca(axarr[i])
        sns.countplot(x=df[col],hue=df['Exited'],data = df) # Seaborn Countplot
        plt.xlabel(col,fontsize='large')
        plt.xticks(rotation=0)

    plt.tight_layout()
    plt.show()
```

[23]:
```
plot(raw_data,["Tenure","NumOfProducts","Geography","Gender"])
```

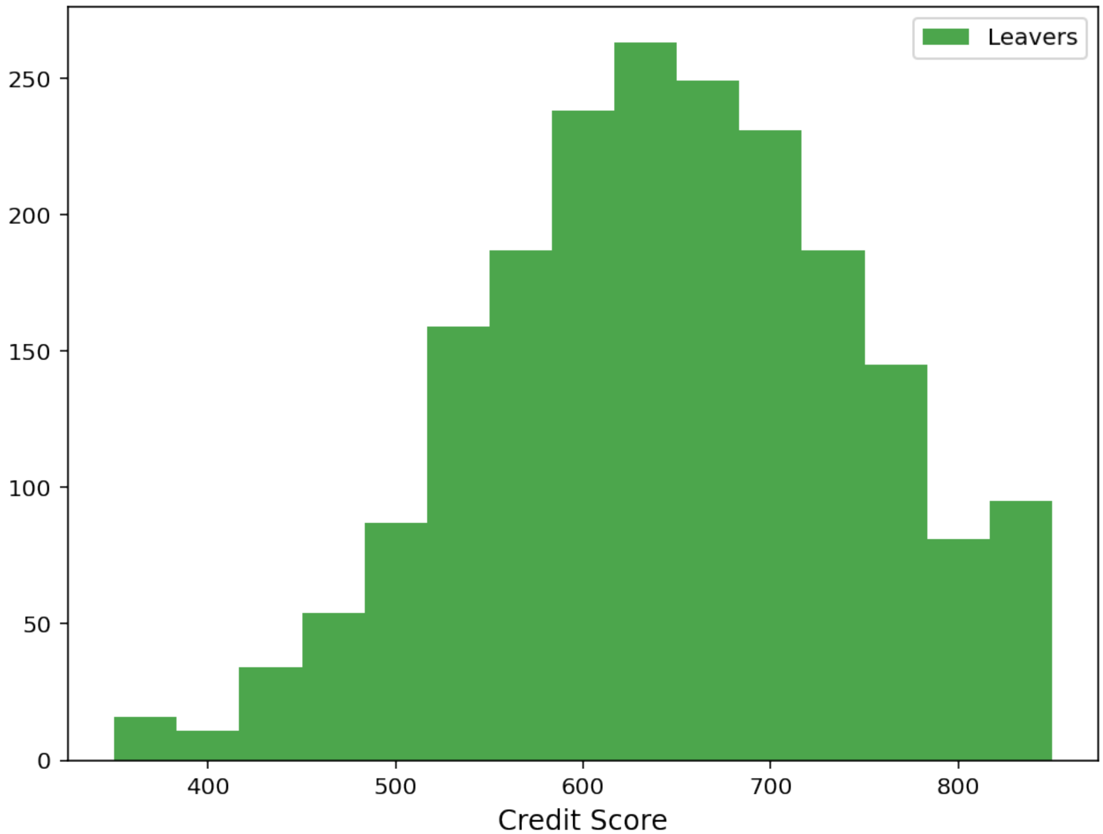


- ***Quick look into the distribution of few variables for leavers and non_leavers***

[24]:
```
# data distribution of the Credit Score for customer not left/leaving
pyplot.figure(figsize=(8,6))
pyplot.xlabel('Credit Score',fontsize='large')
pyplot.hist(non_leavers["CreditScore"],bins=15, alpha=0.7, label='Not␣
↪Leavers',color='red')
pyplot.legend(loc='upper right')
pyplot.show()
```

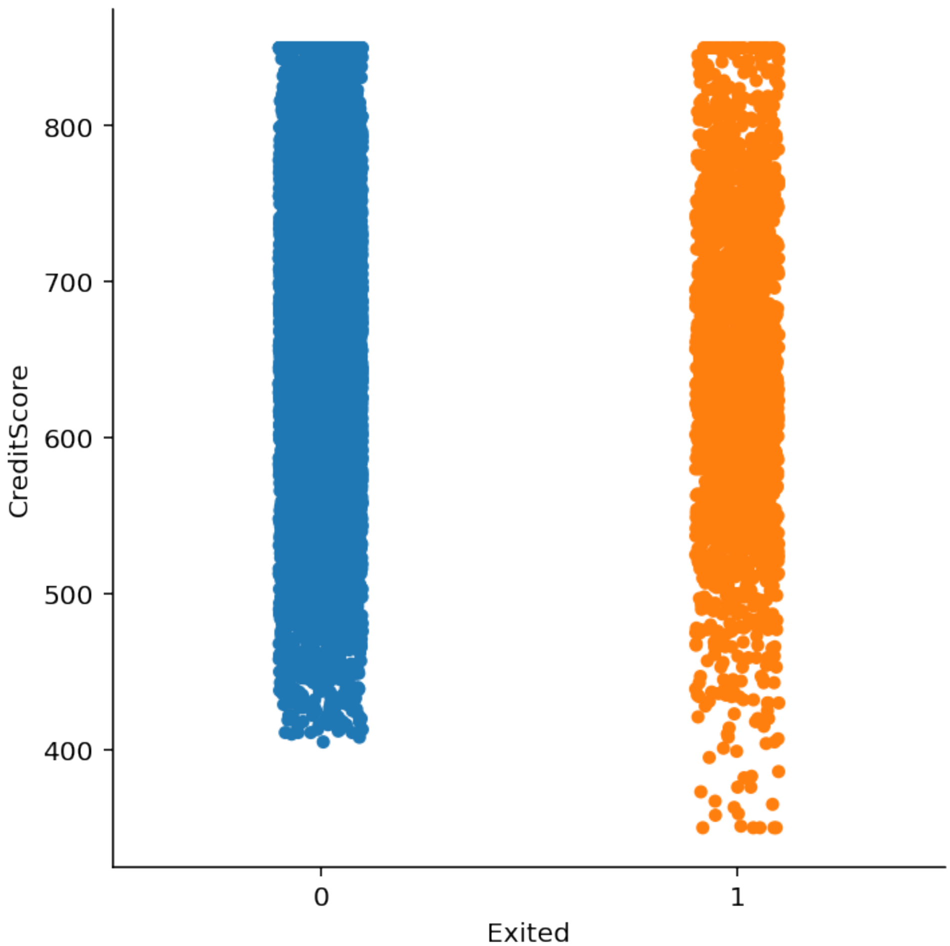


[25]:
```
# data distribution of the Credit Score for customer left/leaving
pyplot.figure(figsize=(8,6))
pyplot.xlabel('Credit Score',fontsize='large')
pyplot.hist(leavers["CreditScore"],bins=15, alpha=0.7,␣
 ↪label='Leavers',color='green')
pyplot.legend(loc='upper right')
pyplot.show()
```



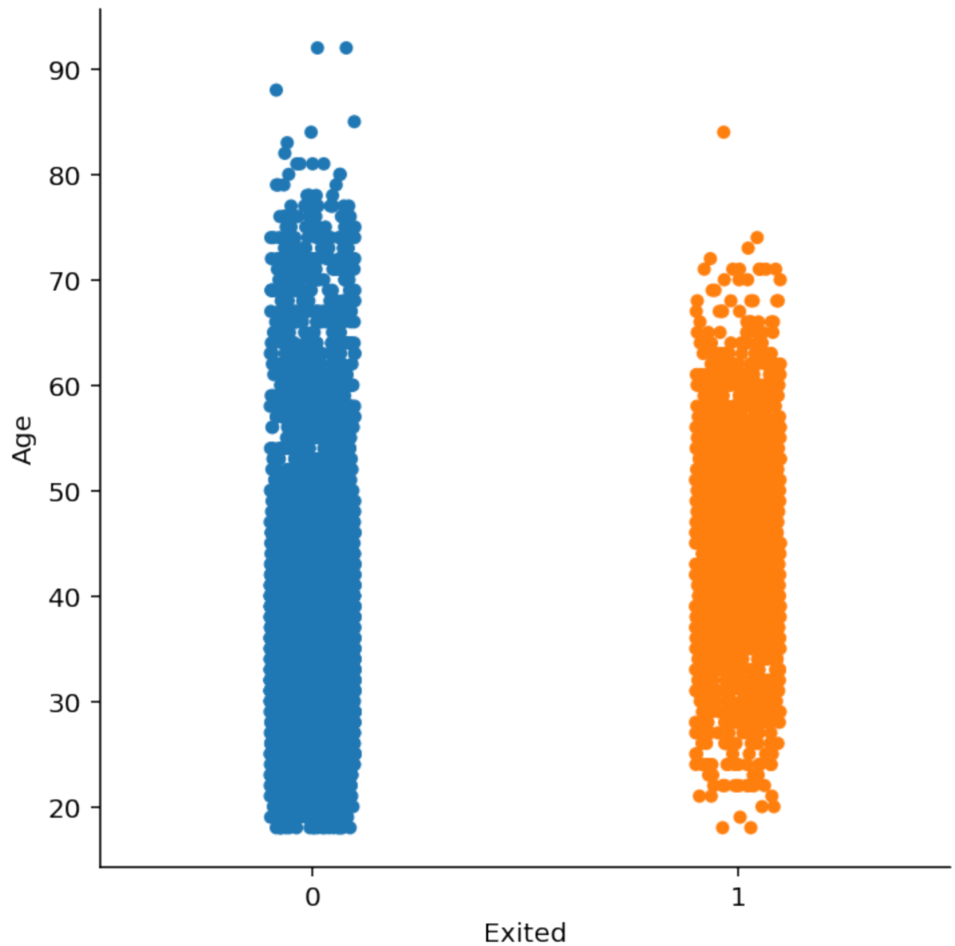
[26]:
```
sns.catplot("Exited", "CreditScore", data = raw_data)
```

[26]:
```
<seaborn.axisgrid.FacetGrid at 0x1a6fb063070>
```

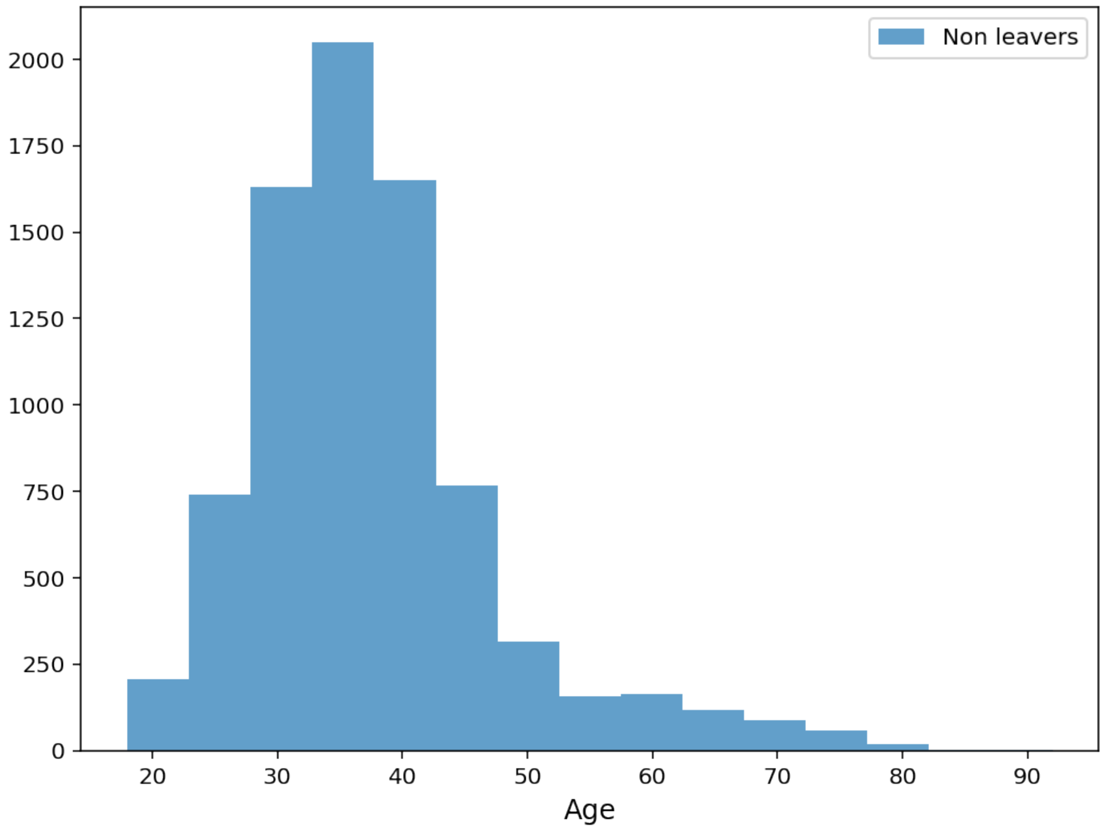



```
[27]: sns.catplot("Exited", "Age", data = raw_data)
```

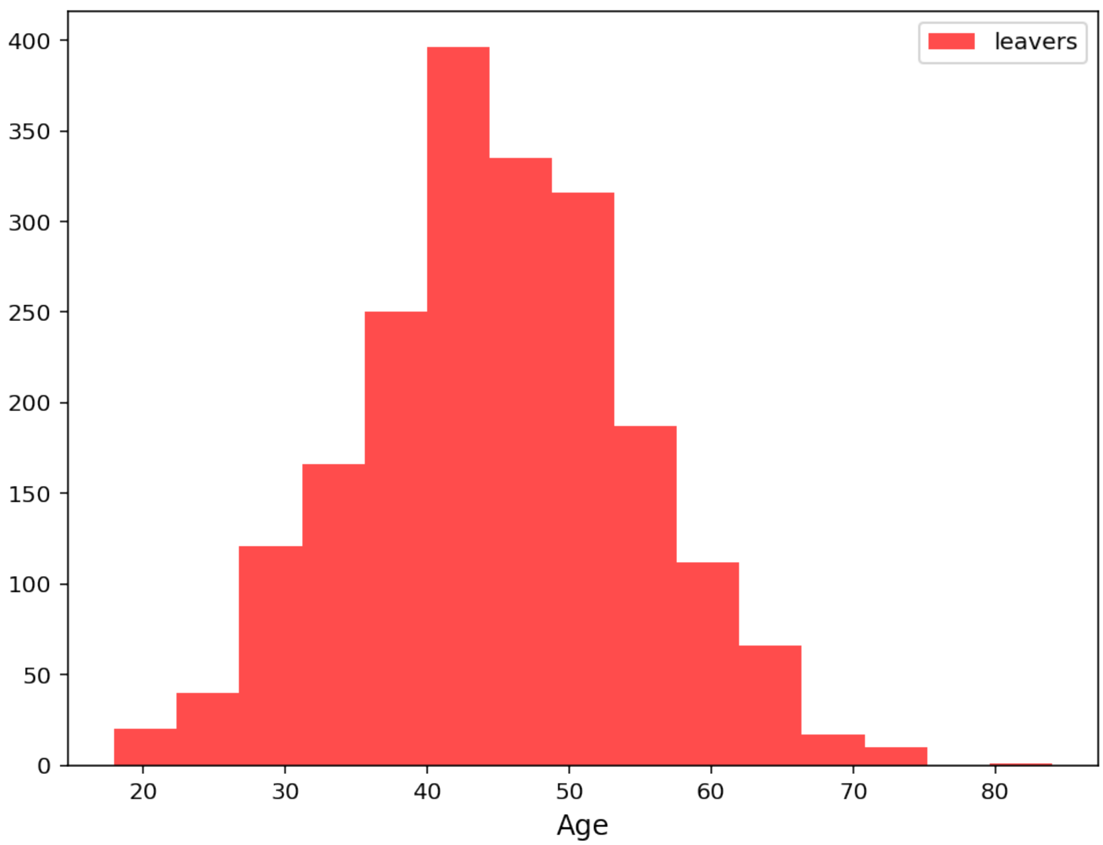
```
[27]: <seaborn.axisgrid.FacetGrid at 0x1a6f7080b20>
```



[28]:
```
# distribution of the Age for customers not left/leaving
pyplot.figure(figsize=(8,6))
pyplot.xlabel('Age',fontsize='large')
pyplot.hist(non_leavers["Age"],bins=15, alpha=0.7, label='Non leavers')
pyplot.legend(loc='upper right')
pyplot.show()
```

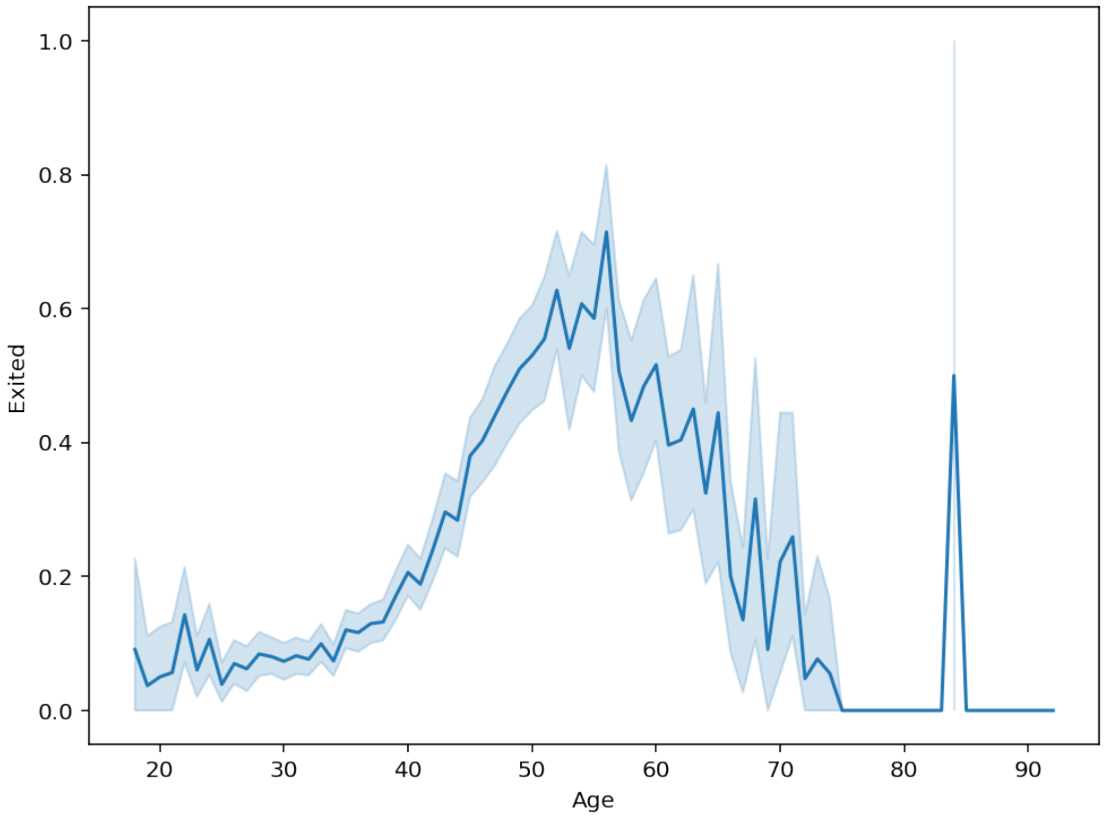


[29]:
```
# Data distribution of the Age for customers left/leaving
pyplot.figure(figsize=(8,6))
pyplot.xlabel('Age',fontsize='large')
pyplot.hist(leavers["Age"],bins=15, alpha=0.7, label='leavers',color='red')
pyplot.legend(loc='upper right')
pyplot.show()
```




```
[30]: plt.figure(figsize=(8,6))
      sns.lineplot(x = "Age", y = "Exited", data = raw_data)
```

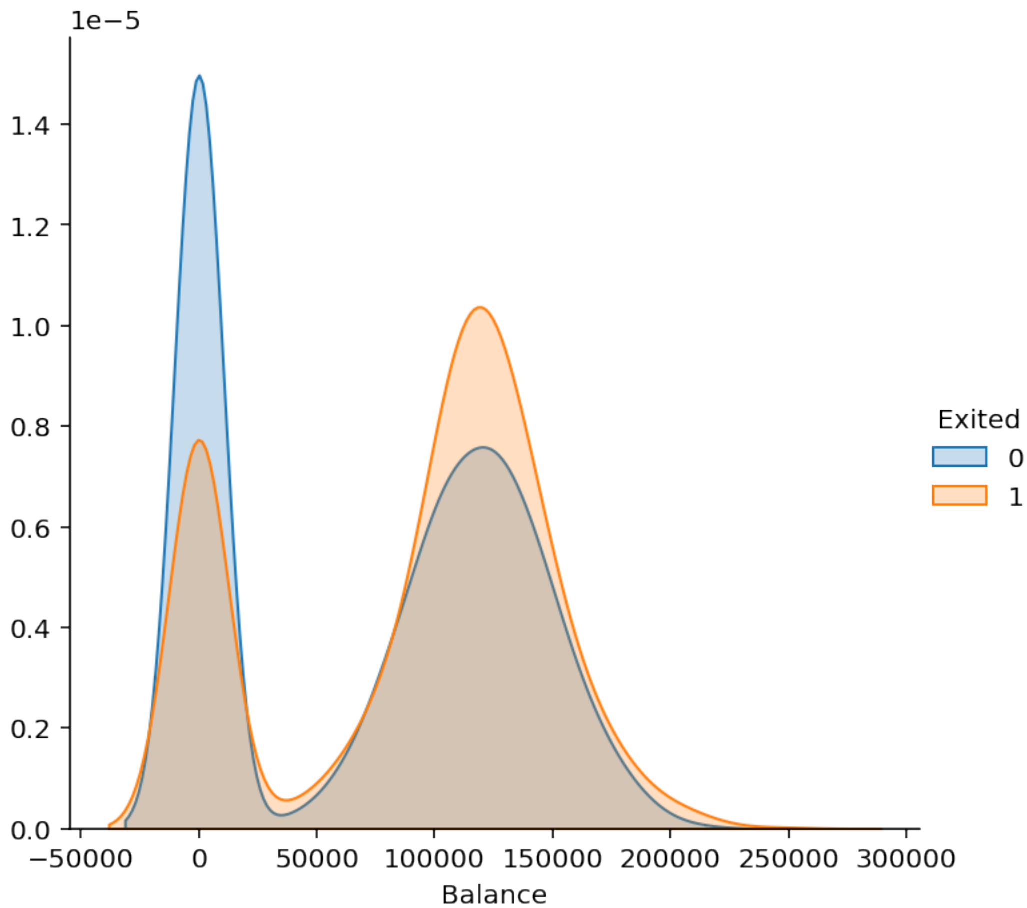
```
[30]: <matplotlib.axes._subplots.AxesSubplot at 0x1a6faee9400>
```



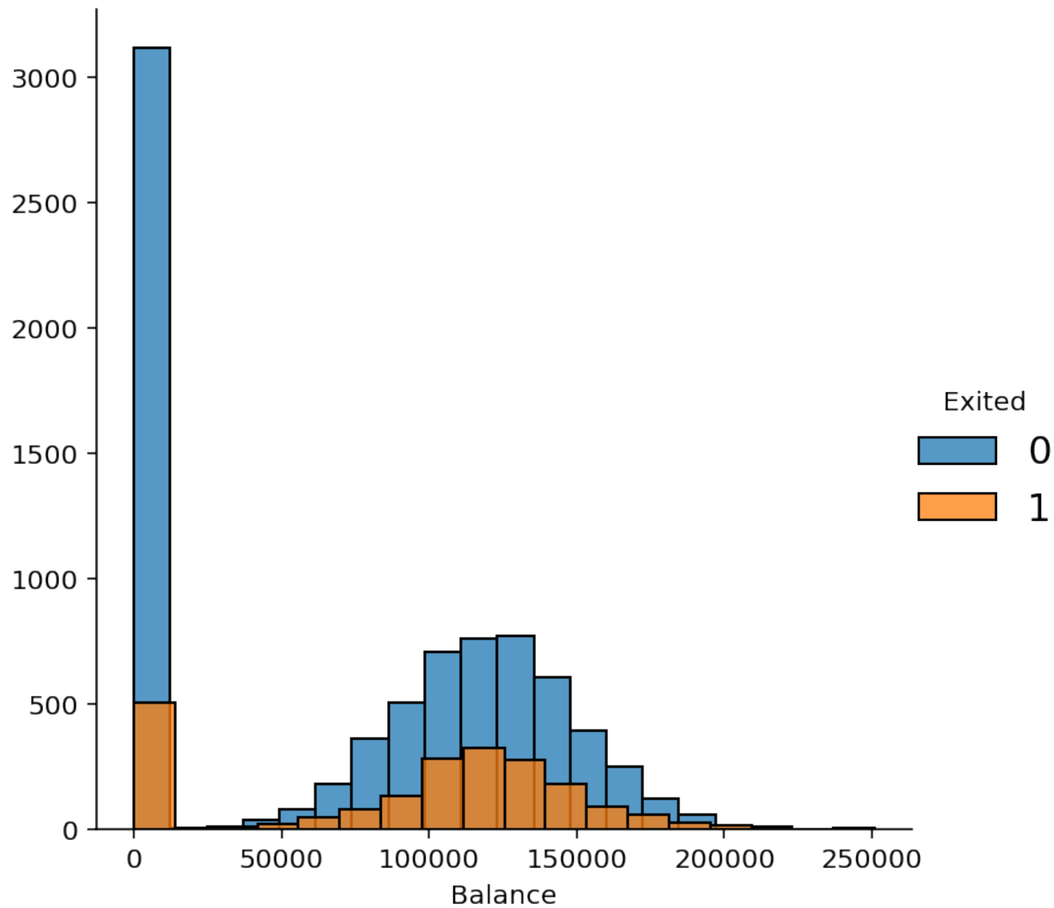
[31]:
```
plt.figure(figsize = (10,8))
sns.FacetGrid(raw_data, hue = "Exited", height = 5).map(sns.kdeplot, "Balance",␣
↪shade= True).add_legend()
```

[31]: <seaborn.axisgrid.FacetGrid at 0x1a6f88dce80>

<Figure size 720x576 with 0 Axes>



[32]:
```
g = sns.FacetGrid(raw_data, hue = "Exited", height = 5)
g.map(sns.histplot, "Balance").add_legend(fontsize ='x-large')
plt.show()
```

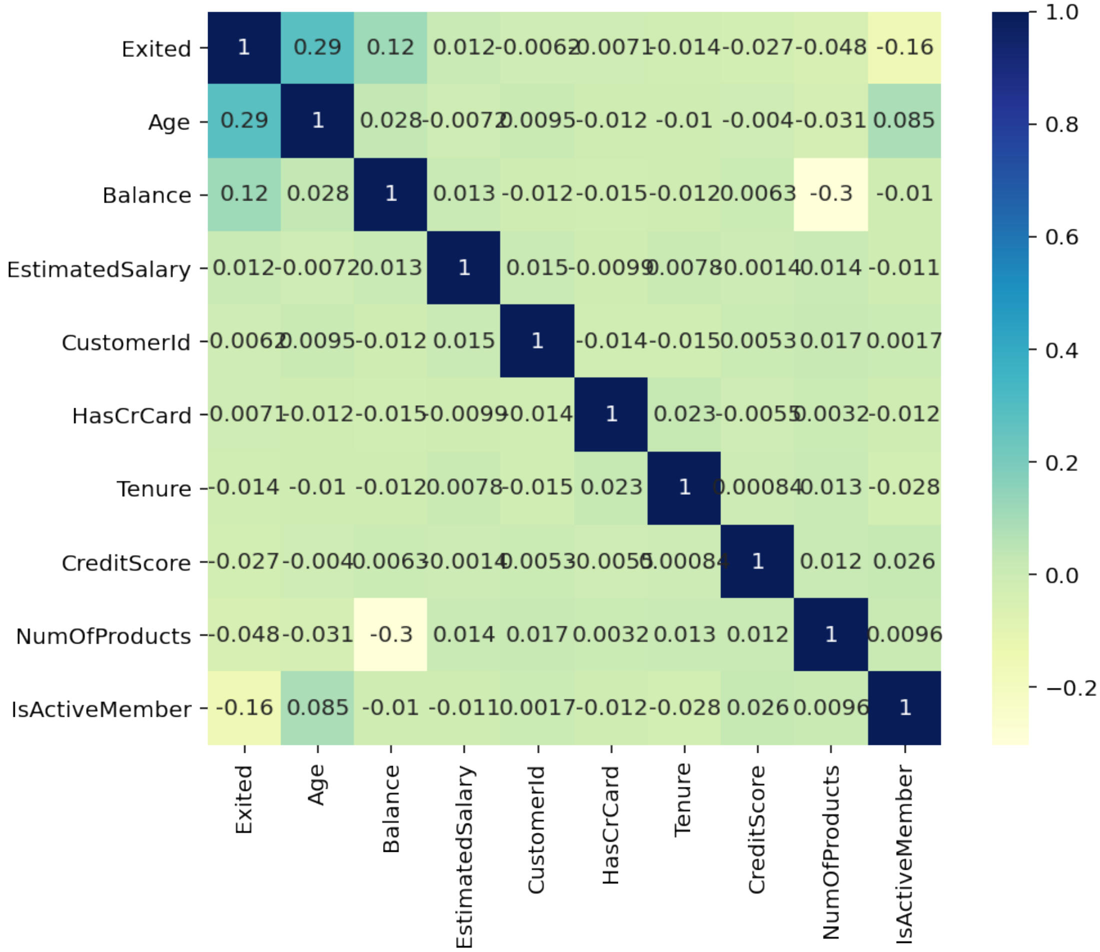


-

1.4.1 *Correlation Matrix - To check how correlated the Independant and Dependant variables are.*

[33]:
```
# Exited correlation matrix
k = 10 #number of variables for heatmap
cols = raw_data.corr().nlargest(k, 'Exited')['Exited'].index
cm = raw_data[cols].corr()
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, cmap = 'YlGnBu',square=True)
```

[33]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a6fb033a00>
```



-

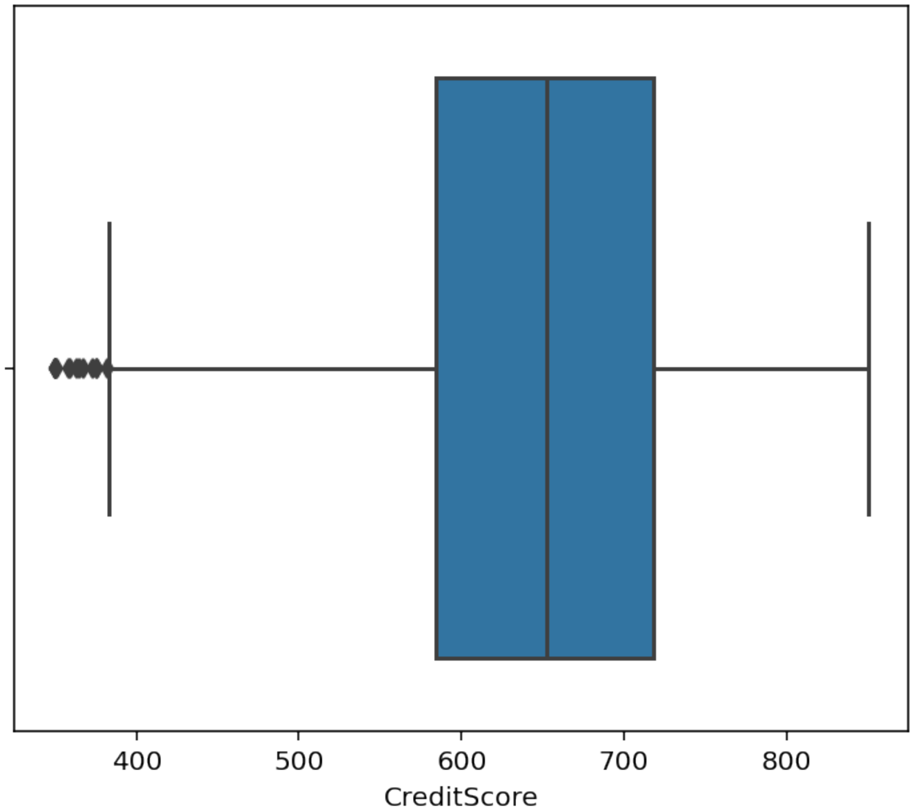
1.4.2 *Using boxplot to look for outliers and defining user define function for code reuse*

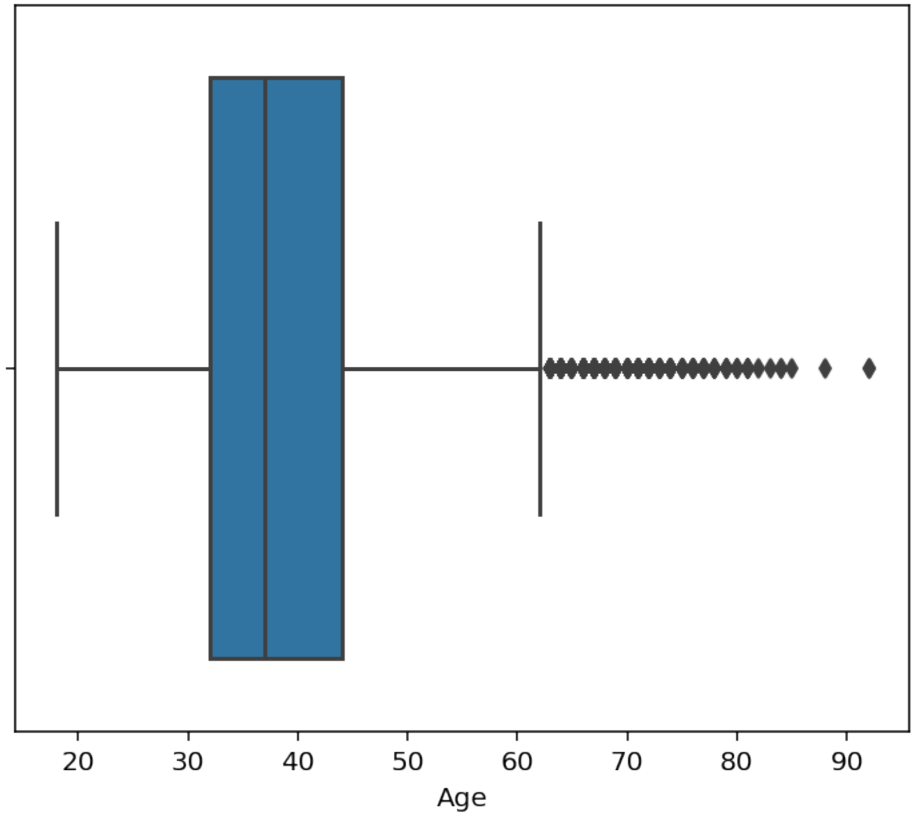
[34]:
```
list_ol =␣
↪["CreditScore","Age","Tenure","Balance","NumOfProducts","EstimatedSalary"]
```

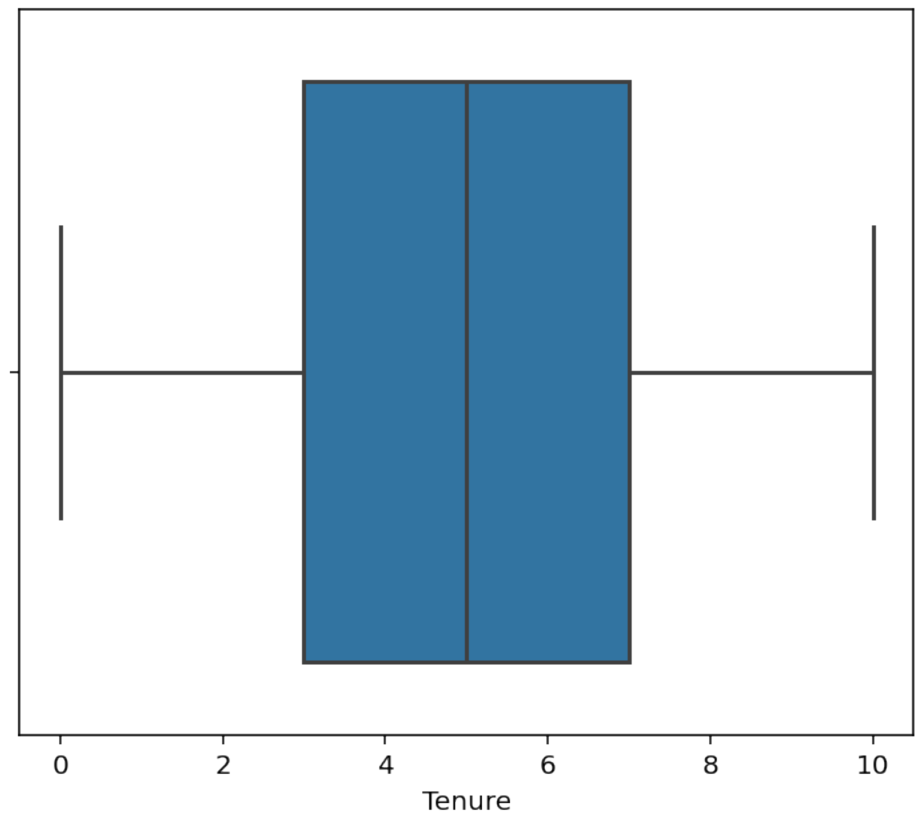
- *Box plot - to look at the data for outliers*

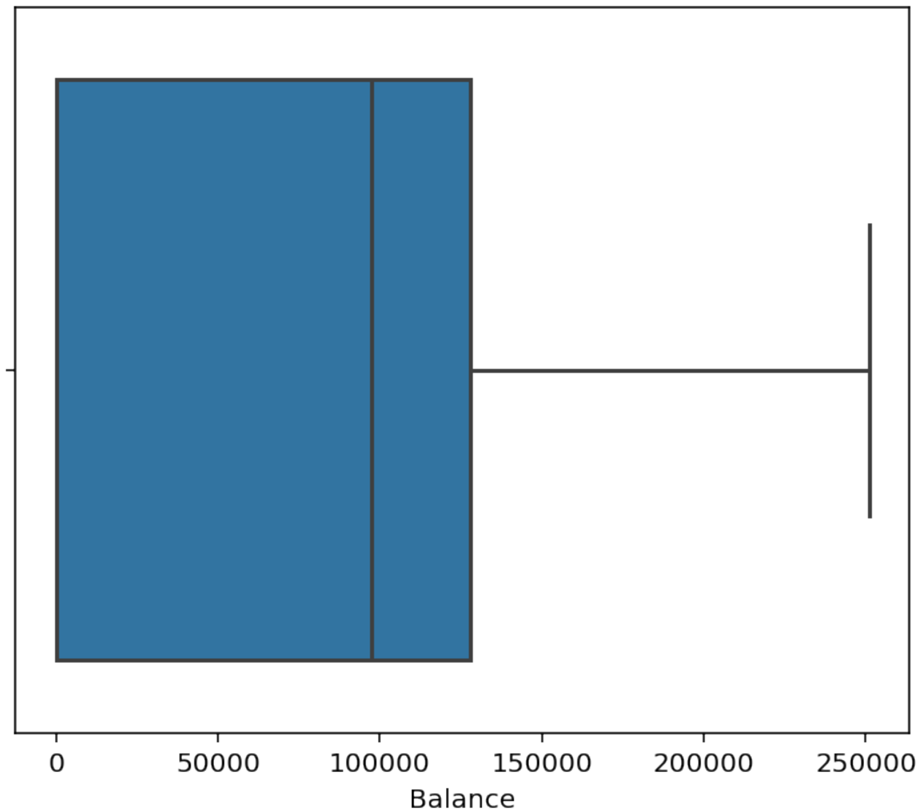
[35]:
```
def outlier_plot(ls):
    for col in ls:
        plt.figure(figsize=(6,5))
        sns.boxplot(raw_data[col])

    plt.show()
```

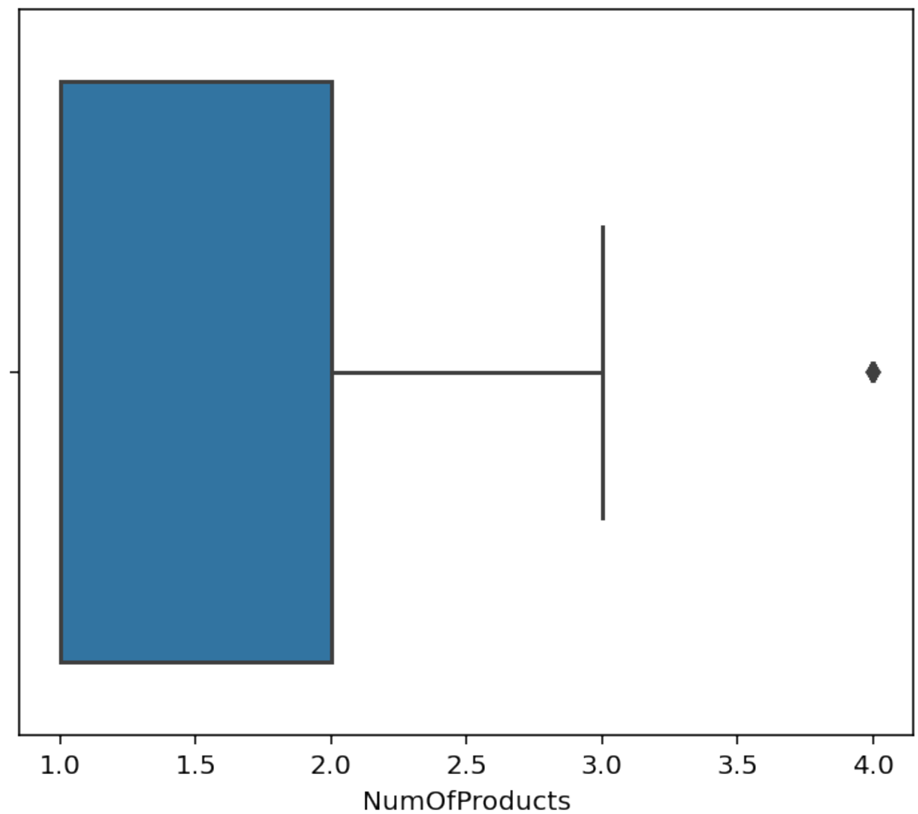
[36]:
```
cols =␣
↪["CreditScore","Age","Tenure","Balance","NumOfProducts","EstimatedSalary"]
outlier_plot(cols)
```

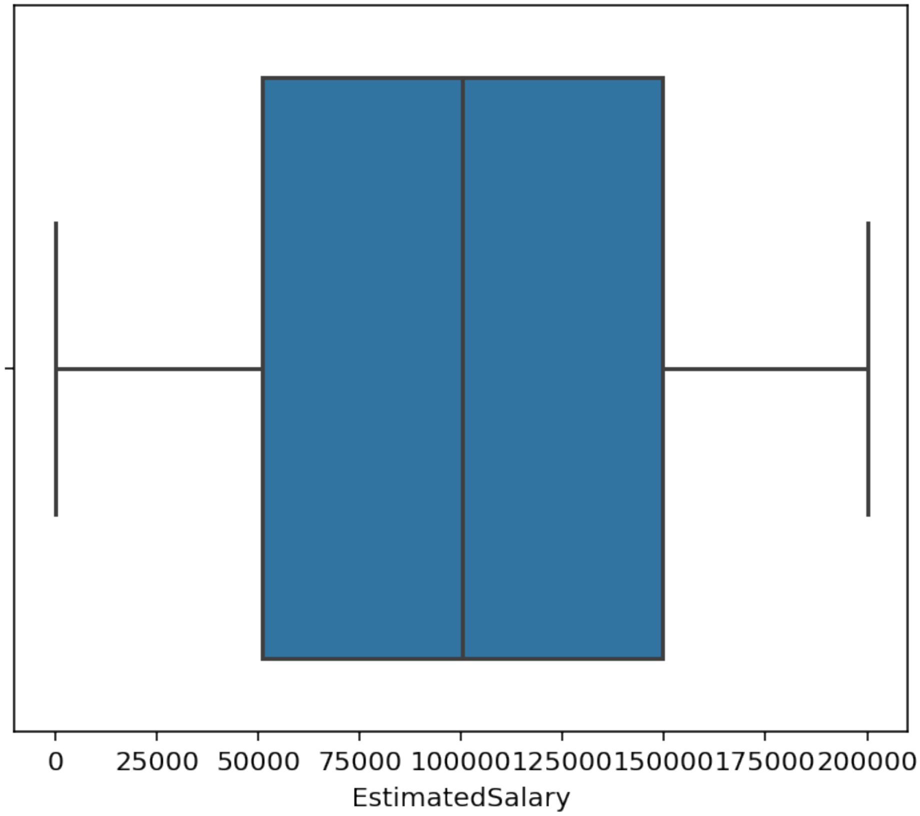



20
30
40
50
60
70
80
90
Age


0
2
4
6
8
10
Tenure


0
50000
100000
150000
200000
250000
Balance


1.0
1.5
2.0
2.5
3.0
3.5
4.0
NumOfProducts



1.5 *Scaling - Using Normalization method*

[37]:
```
raw_data["NumOfProducts"] = raw_data["NumOfProducts"].astype("category")
raw_data["HasCrCard"] = raw_data["HasCrCard"].astype("category")
raw_data["IsActiveMember"] = raw_data["IsActiveMember"].astype("category")

raw_data = pd.get_dummies(raw_data, columns = ["Geography"])
raw_data = pd.get_dummies(raw_data, columns = ["Gender"])
raw_data = pd.get_dummies(raw_data, columns = ["NumOfProducts"])
raw_data = pd.get_dummies(raw_data, columns = ["HasCrCard"])
raw_data = pd.get_dummies(raw_data, columns = ["IsActiveMember"])
```

[38]:
```
raw_data.head()
```

[38]:
```
           CustomerId   Surname  CreditScore  Age  Tenure    Balance  \
RowNumber
1            15634602  Hargrave          619   42       2       0.00
2            15647311      Hill          608   41       1   83807.86
3            15619304      Onio          502   42       8  159660.80
4            15701354      Boni          699   39       1       0.00
```

```
5            15737888  Mitchell          850   43       2  125510.82

           EstimatedSalary  Exited  Geography_France  Geography_Germany  \
RowNumber
1                101348.88       1                 1                  0
2                112542.58       0                 0                  0
3                113931.57       1                 1                  0
4                 93826.63       0                 1                  0
5                 79084.10       0                 0                  0

           Geography_Spain  Gender_Female  Gender_Male  NumOfProducts_1  \
RowNumber
1                        0              1            0                1
2                        1              1            0                1
3                        0              1            0                0
4                        0              1            0                0
5                        1              1            0                1

           NumOfProducts_2  NumOfProducts_3  NumOfProducts_4  HasCrCard_0  \
RowNumber
1                        0                0                0            0
2                        0                0                0            1
3                        0                1                0            0
4                        1                0                0            1
5                        0                0                0            0

           HasCrCard_1  IsActiveMember_0  IsActiveMember_1
RowNumber
1                    1                 0                 1
2                    0                 0                 1
3                    1                 1                 0
4                    0                 1                 0
5                    1                 0                 1
```

- ***Dropping Tenure, CustomerId and Surname as it has no impact on our dependant variable "Exited"***

[39]:
```
raw_data.drop(["Tenure"], axis = 1 , inplace = True)
```

[40]:
```
x_raw_data = raw_data.drop(["Exited","CustomerId","Surname"], axis=1)
y = raw_data["Exited"]

x = (x_raw_data - np.min(x_raw_data)) / (np.max(x_raw_data)-np.min(x_raw_data)).
 ↪values
x.head()
```

[40]:

RowNumber	CreditScore	Age	Balance	EstimatedSalary	Geography_France
1	0.538	0.324324	0.000000	0.506735	1.0
2	0.516	0.310811	0.334031	0.562709	0.0
3	0.304	0.324324	0.636357	0.569654	1.0
4	0.698	0.283784	0.000000	0.469120	1.0
5	1.000	0.337838	0.500246	0.395400	0.0

RowNumber	Geography_Germany	Geography_Spain	Gender_Female	Gender_Male
1	0.0	0.0	1.0	0.0
2	0.0	1.0	1.0	0.0
3	0.0	0.0	1.0	0.0
4	0.0	0.0	1.0	0.0
5	0.0	1.0	1.0	0.0

RowNumber	NumOfProducts_1	NumOfProducts_2	NumOfProducts_3	NumOfProducts_4
1	1.0	0.0	0.0	0.0
2	1.0	0.0	0.0	0.0
3	0.0	0.0	1.0	0.0
4	0.0	1.0	0.0	0.0
5	1.0	0.0	0.0	0.0

RowNumber	HasCrCard_0	HasCrCard_1	IsActiveMember_0	IsActiveMember_1
1	0.0	1.0	0.0	1.0
2	1.0	0.0	0.0	1.0
3	0.0	1.0	1.0	0.0
4	1.0	0.0	1.0	0.0
5	0.0	1.0	0.0	1.0

1.6 *Data Modeling*

- ***Splitting the data into Train & Test sets***

[41]:
```
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size = 0.20,␣
↪random_state = 42)
```

- ***::Logistic Regression::***

  #### Y = 0 + 1X1 +…+ p−1Xp−1 + , y (Exited) our dependent variable, x is the input of our independent variables (a, b, and c are the input coefficients of our independent variables), and e is our constant.

- ***Modelling***

[42]:
```
Model_log = LogisticRegression(solver = "liblinear")
Model_log.fit(x_train,y_train)
```

```
Model_log
```

[42]: LogisticRegression(solver='liblinear')

[43]:
```
Model_log.intercept_ # This is give you value for beta0
```

[43]: array([-0.31925513])

[44]:
```
Model_log.coef_ # This will give you values for all the betas of each␣
↪independant variable
```

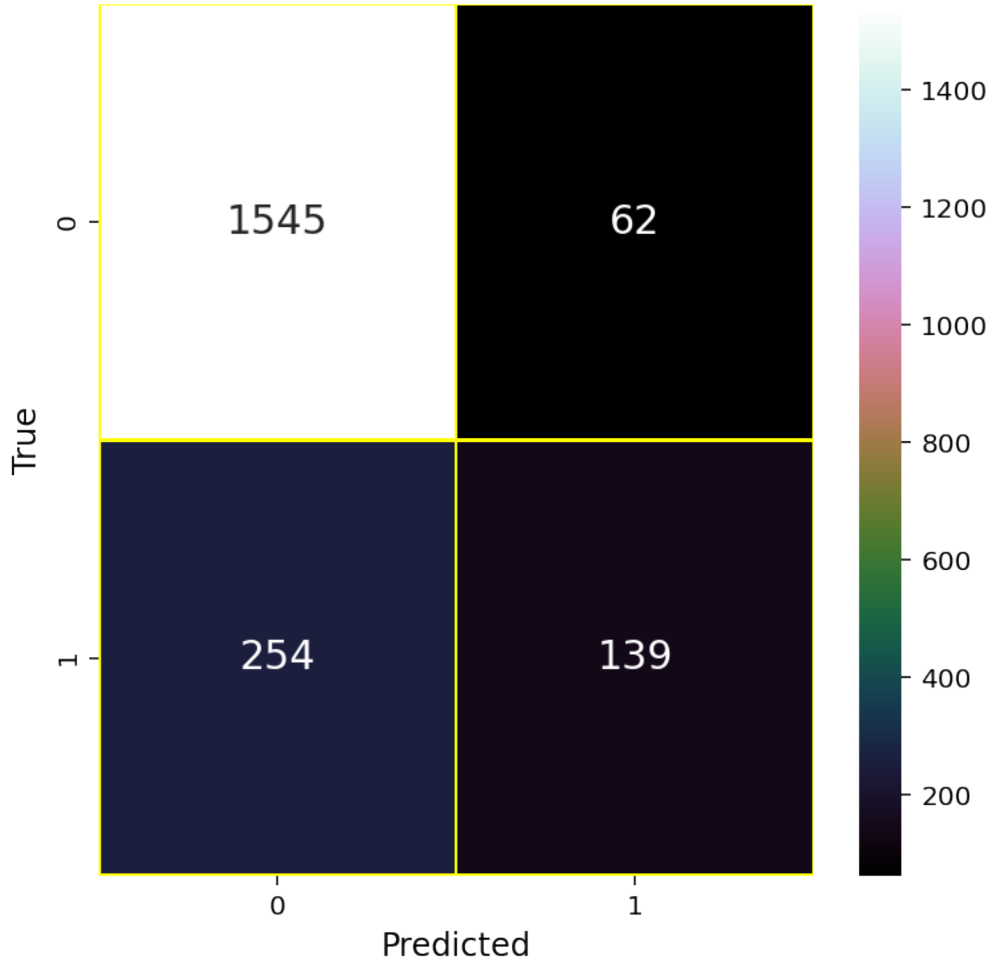
[44]:
```
array([[-0.35498171,  4.89893033, -0.19784012,  0.02217815, -0.4583462 ,
         0.49214686, -0.35305579,  0.1017742 , -0.42102933, -1.28342306,
        -2.8123298 ,  1.39728649,  2.37921124, -0.14205903, -0.1771961 ,
         0.37502271, -0.69427784]])
```

[45]:
```
print("Test accurarcy {}".format(Model_log.score(x_test,y_test)))
```

```
Test accurarcy 0.842
```

- ***Confusion Matrix - to get better understanding on how our model is performing***

[46]:
```
y_pred = Model_log.predict(x_test)
y_actual = y_test

cm =confusion_matrix(y_actual,y_pred)

f, ax = plt.subplots(figsize = (6,6))
sns.heatmap(cm,annot=True,annot_kws={'size': 15},linewidths=0.
↪7,linecolor="yellow",fmt = "g",ax=ax,cmap="cubehelix")
plt.xlabel("Predicted", fontsize = 'large')
plt.ylabel("True", fontsize = 'large')
plt.show()
```




```
[47]: cross_val_score(Model_log, x_test, y_test, cv = 10).mean()
```

```
[47]: 0.8424999999999999
```

- ***::Random Forest::***

```
[48]: model_rf = RandomForestClassifier().fit(x_train, y_train)
```

```
[49]: y_pred = model_rf.predict(x_test)
      print("Test accurarcy {}".format(accuracy_score(y_test, y_pred)))
```

```
Test accurarcy 0.863
```

- ***Tuning Hyperparametre for random forest model***

```
[50]: hyper_rf = {"max_depth": [2,5,8,10],
                  "max_features": [2,5,8],
                  "n_estimators": [10,500,1000],
```

```
                "min_samples_split": [2,5,10]}
```

[51]:
```
model_rf = RandomForestClassifier()

model_cv_rf = GridSearchCV(model_rf,
                           hyper_rf,
                           cv = 10,
                           n_jobs = -1,
                           verbose = 2)
```

[52]:
```
model_cv_rf.fit(x_train, y_train)
```

```
Fitting 10 folds for each of 108 candidates, totalling 1080 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 16 concurrent workers.
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:    1.4s
[Parallel(n_jobs=-1)]: Done 130 tasks      | elapsed:   12.8s
[Parallel(n_jobs=-1)]: Done 333 tasks      | elapsed:   36.3s
[Parallel(n_jobs=-1)]: Done 616 tasks      | elapsed:  1.4min
[Parallel(n_jobs=-1)]: Done 981 tasks      | elapsed:  2.7min
[Parallel(n_jobs=-1)]: Done 1080 out of 1080 | elapsed:  3.3min finished
```

[52]:
```
GridSearchCV(cv=10, estimator=RandomForestClassifier(), n_jobs=-1,
             param_grid={'max_depth': [2, 5, 8, 10], 'max_features': [2, 5, 8],
                         'min_samples_split': [2, 5, 10],
                         'n_estimators': [10, 500, 1000]},
             verbose=2)
```

[53]:
```
print("The best values for the hyperparameters are: " + str(model_cv_rf.
 ↪best_params_))
```

```
The best values for the hyperparameters are: {'max_depth': 10, 'max_features':
8, 'min_samples_split': 10, 'n_estimators': 500}
```

- ***Using best values for hyperparameters of random forest***

[54]:
```
tuned_rf = RandomForestClassifier(max_depth = 10,
                                  max_features = 8,
                                  min_samples_split = 10,
                                  n_estimators = 1000)


tuned_rf.fit(x_train, y_train)
```

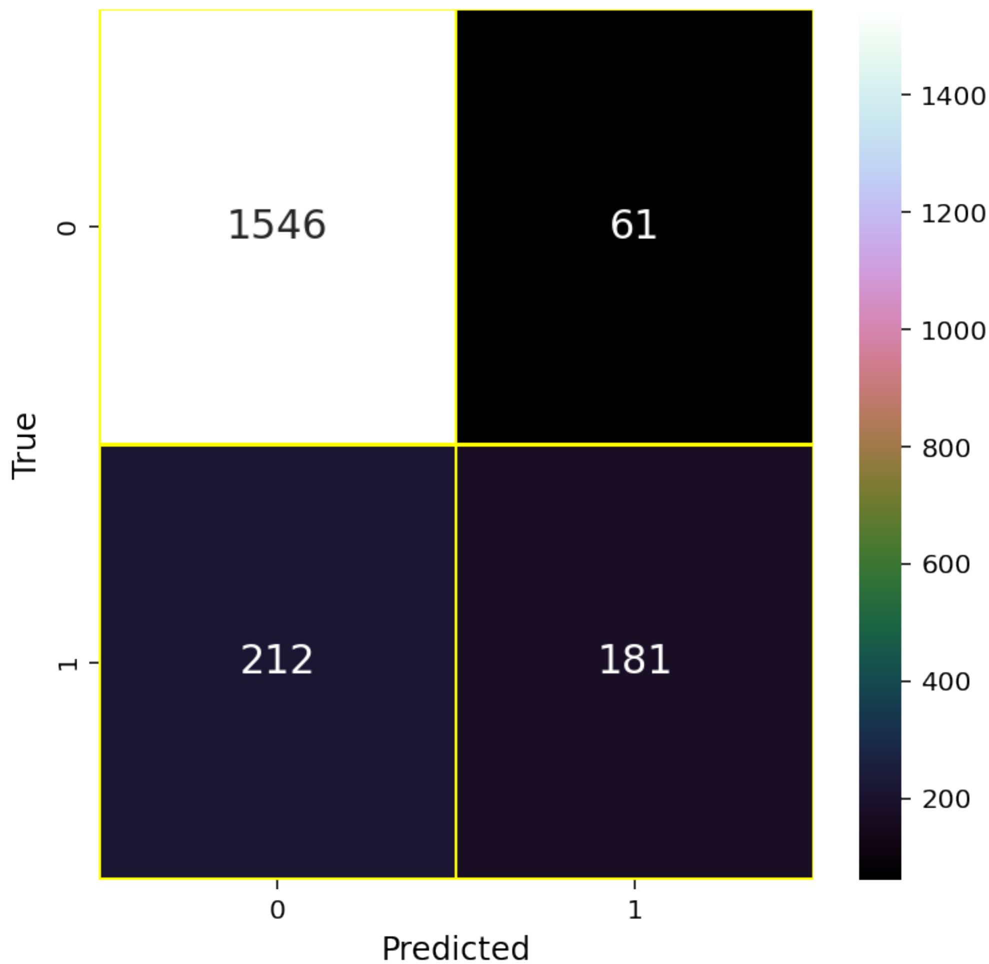
[54]:
```
RandomForestClassifier(max_depth=10, max_features=8, min_samples_split=10,
                       n_estimators=1000)
```

[55]:
```
y_pred = tuned_rf.predict(x_test)
accuracy_score(y_test, y_pred)
```

[55]:
```
0.8635
```

[56]:

```
y_actual = y_test

cm =confusion_matrix(y_actual,y_pred)

f, ax = plt.subplots(figsize = (6,6))
sns.heatmap(cm,annot=True,annot_kws={'size': 15},linewidths=0.
 ↪7,linecolor="yellow",fmt = "g",ax=ax,cmap="cubehelix")
plt.xlabel("Predicted", fontsize = 'large')
plt.ylabel("True", fontsize = 'large')
plt.show()
```



[57]:

```
models = [
    Model_log,
    tuned_rf,
        ]
```

```python
for model in models:
    names = model.__class__.__name__
    y_pred = model.predict(x_test)
    accuracy = accuracy_score(y_test, y_pred)
    print("-"*25)
    print(":: "+names + " ::" )
    print("Accuracy: {:.3%}".format(accuracy))
```

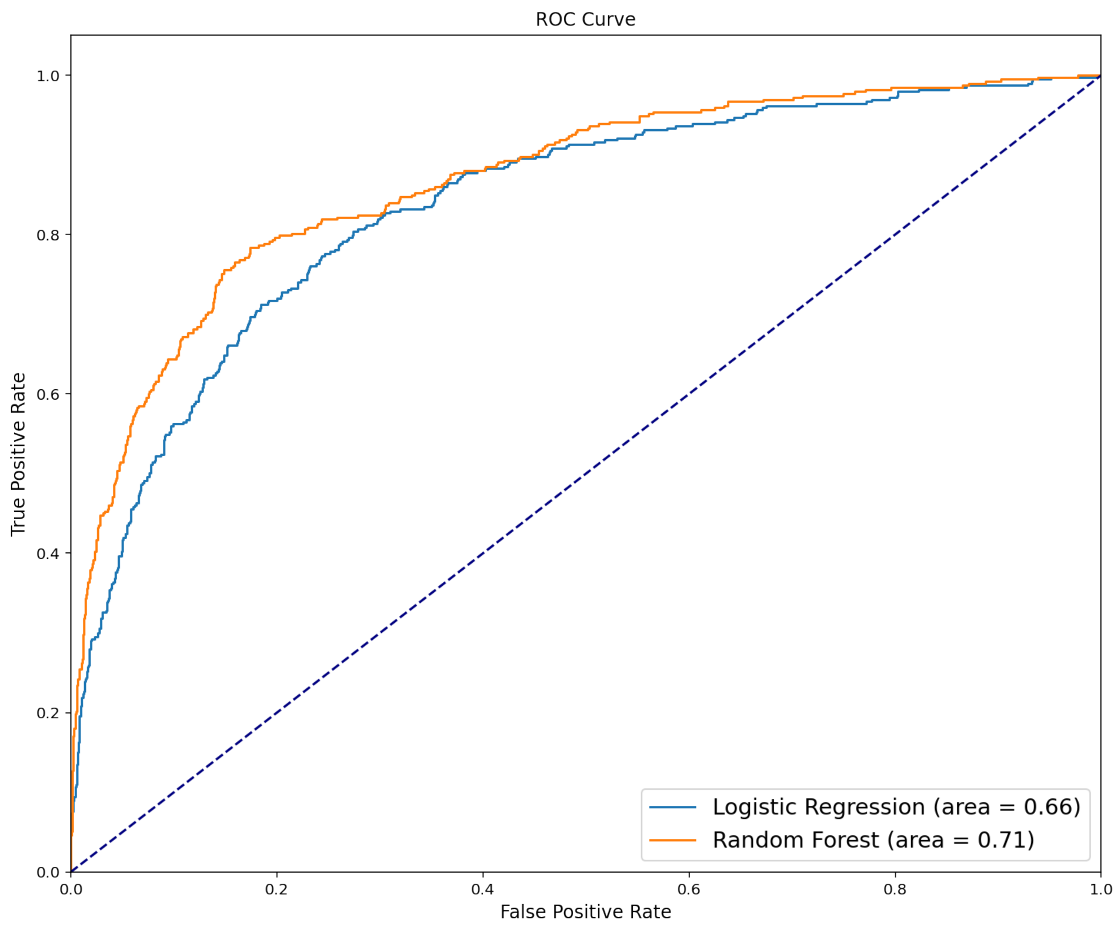
```
-------------------------
:: LogisticRegression ::
Accuracy: 84.200%
-------------------------
:: RandomForestClassifier ::
Accuracy: 86.350%
```

- ***Receiver Operating Characteristic (ROC) Curve***

The ROC curve features true positive on the y-axis and false positive on the x-axis where the ideal point is located at (0,1). This is so that the performance of our classification model can be measured in terms of how close the graph comes to the ideal point (threshold) and how much area they cover under the curve.

[58]:
```python
log_roc = roc_auc_score(y_test, Model_log.predict(x_test))
fpr, tpr, thresholds = roc_curve(y_test, Model_log.predict_proba(x_test)[:,1])
rf_roc = roc_auc_score(y_test, tuned_rf.predict(x_test))
rf_fpr, rf_tpr, rf_thresholds = roc_curve(y_test, tuned_rf.
 ↪predict_proba(x_test)[:,1])
```

[59]:
```python
plt.figure(figsize = (12,10))
plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % log_roc)
plt.plot(rf_fpr, rf_tpr, label='Random Forest (area = %0.2f)' % rf_roc)
plt.plot([0, 1], [0, 1],'r--',color='navy') # random classifier line
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate', fontsize = "large")
plt.ylabel('True Positive Rate', fontsize = "large")
plt.title('ROC Curve')
plt.legend(loc="lower right", fontsize = 'x-large')
plt.show()
```


ROC Curve
1.0
0.8
0.6
0.4
0.2
0.0
True Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate
Logistic Regression (area = 0.66)
Random Forest (area = 0.71)