DSC680_CCPB_ModelSelection_Evaluation

January 30, 2022

[1]:
```
from ipynb.fs.full.DSC680_CCPB_EDA import *
```

```
********* Customer Churn Dataset****************
```

```
<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
```

```
********* number of unique classes of each attributes****************
```

```
RowNumber          10000
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
dtype: int64
```

********** description of the dataset**********

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

             Balance  NumOfProducts    HasCrCard  IsActiveMember  \
count   10000.000000   10000.000000  10000.00000    10000.000000
mean    76485.889288       1.530200      0.70550        0.515100
std     62397.405202       0.581654      0.45584        0.499797
min         0.000000       1.000000      0.00000        0.000000
25%         0.000000       1.000000      0.00000        0.000000
50%     97198.540000       1.000000      1.00000        1.000000
75%    127644.240000       2.000000      1.00000        1.000000
max    250898.090000       4.000000      1.00000        1.000000

       EstimatedSalary        Exited
count     10000.000000  10000.000000
mean     100090.239881      0.203700
std       57510.492818      0.402769
min          11.580000      0.000000
25%       51002.110000      0.000000
50%      100193.915000      0.000000
75%      149388.247500      0.000000
max      199992.480000      1.000000
```

********* Sample Data from file****************

```
   RowNumber  CustomerId   Surname  CreditScore Geography  Gender  Age  \
0          1    15634602  Hargrave          619    France  Female   42
1          2    15647311      Hill          608     Spain  Female   41
2          3    15619304      Onio          502    France  Female   42
3          4    15701354      Boni          699    France  Female   39
4          5    15737888  Mitchell          850     Spain  Female   43

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

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

********* dropped columns [CustomerId,RowNumber,Surname]****************

```
CreditScore          int64
Geography           object
Gender              object
Age                  int64
Tenure               int64
Balance            float64
NumOfProducts        int64
HasCrCard            int64
IsActiveMember       int64
EstimatedSalary    float64
Exited               int64
dtype: object
```

*********Check number of NaN or NULL****************

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

***********boxplot for numerical features******************

***********Analyze correlation among Exited and other Categorical Features*****************

***********Analyze Correlation between numerical feature using Heatmap plot*****************

***********Correlation statistics*****************

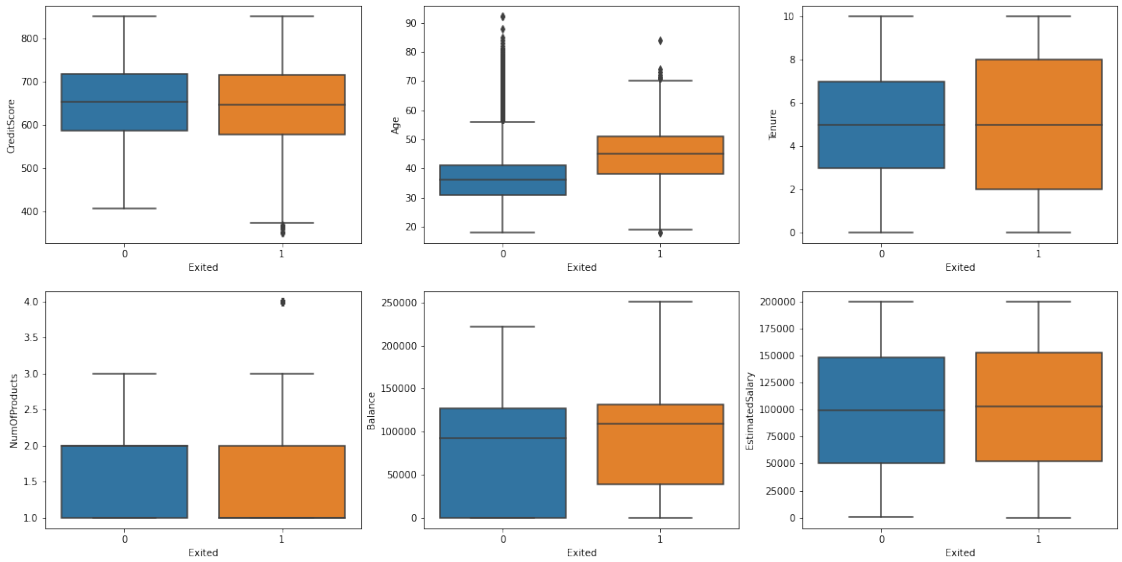
```
                 IsActiveMember  HasCrCard  CreditScore       Age    Tenure  \
IsActiveMember         1.000000  -0.011866     0.025651  0.085472 -0.028362
HasCrCard             -0.011866   1.000000    -0.005458 -0.011721  0.022583
CreditScore            0.025651  -0.005458     1.000000 -0.003965  0.000842
Age                    0.085472  -0.011721    -0.003965  1.000000 -0.009997
Tenure                -0.028362   0.022583     0.000842 -0.009997  1.000000
NumOfProducts          0.009612   0.003183     0.012238 -0.030680  0.013444
Balance               -0.010084  -0.014858     0.006268  0.028308 -0.012254
EstimatedSalary       -0.011421  -0.009933    -0.001384 -0.007201  0.007784

                 NumOfProducts   Balance  EstimatedSalary
```

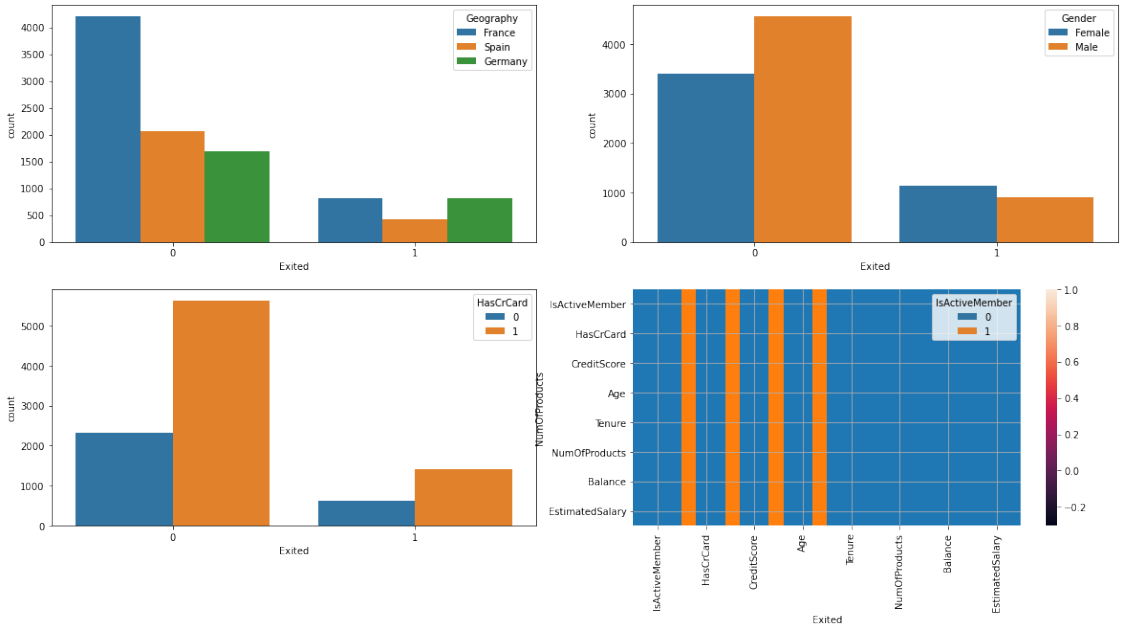
```
IsActiveMember        0.009612 -0.010084        -0.011421
HasCrCard             0.003183 -0.014858        -0.009933
CreditScore           0.012238  0.006268        -0.001384
Age                  -0.030680  0.028308        -0.007201
Tenure                0.013444 -0.012254         0.007784
NumOfProducts         1.000000 -0.304180         0.014204
Balance              -0.304180  1.000000         0.012797
EstimatedSalary       0.014204  0.012797         1.000000
```

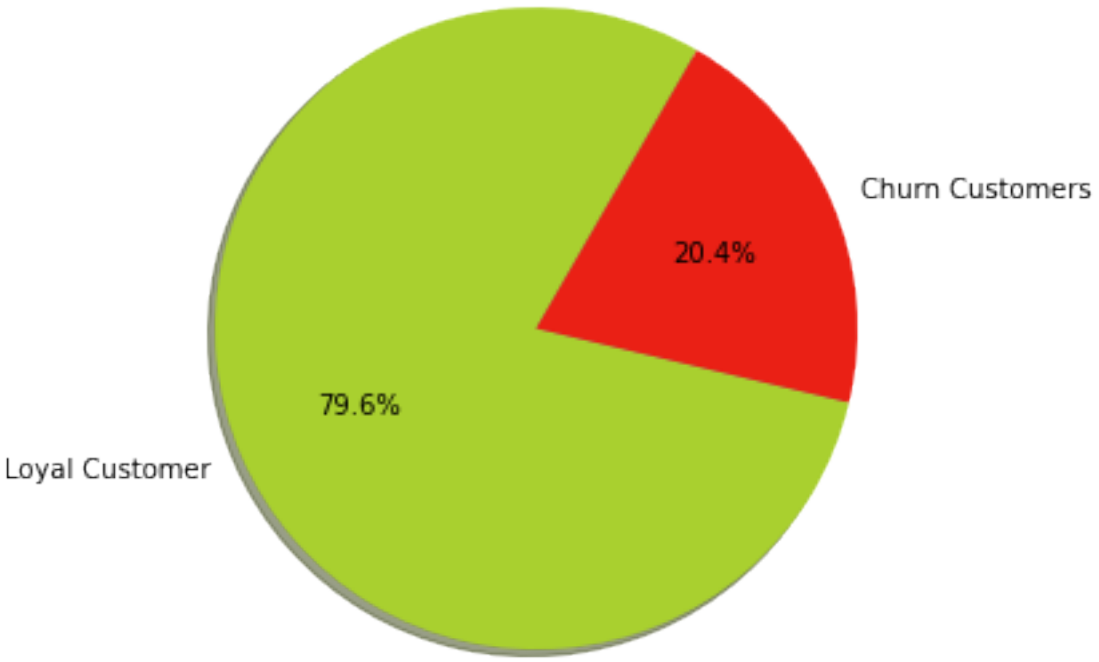
***********Balance Distribution EDA*****************

***********What is the minumum balance of the customers who exited the bank?*****************

***********What is the Percentage of loyal customers vs churn customers?*****************

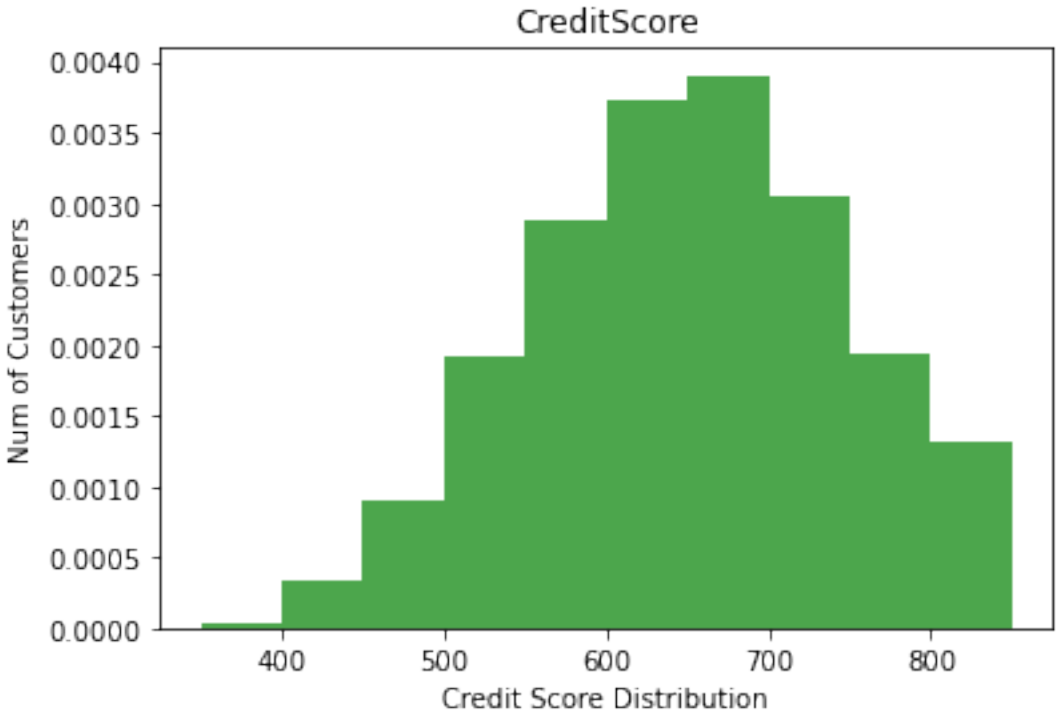


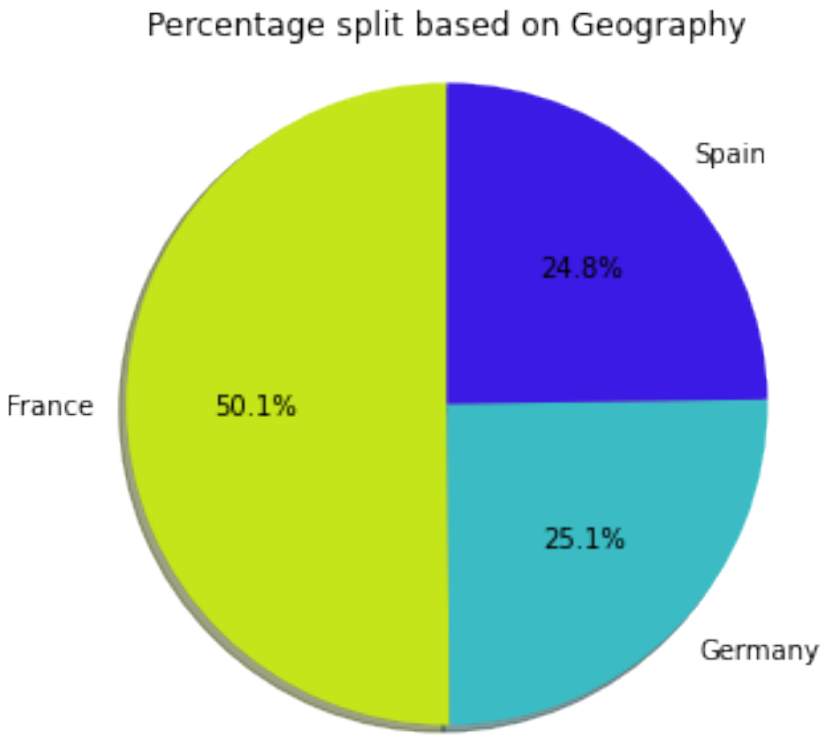




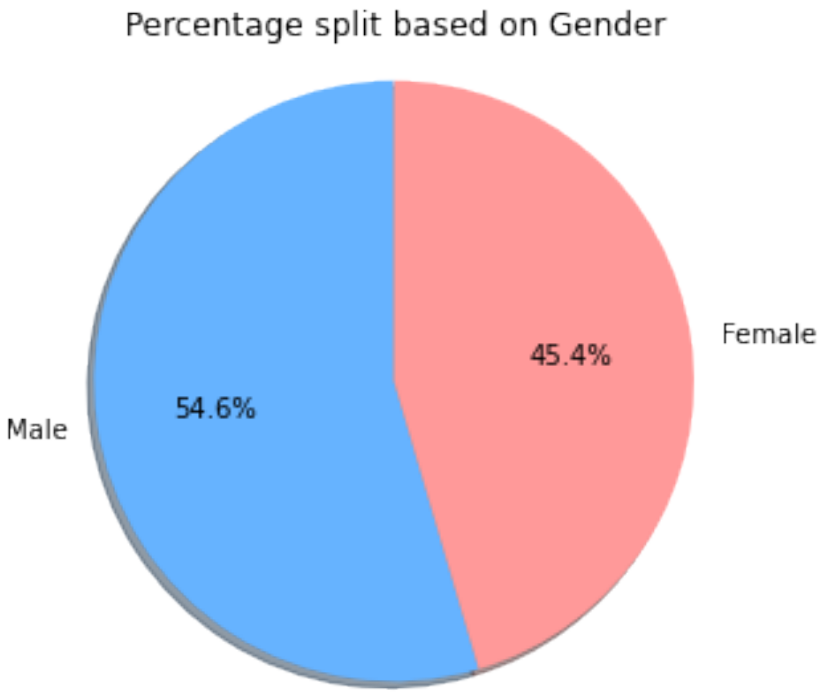
*********Customer's Credit Score Distribution****************

*********Customer churn rate w.r.t geography***************

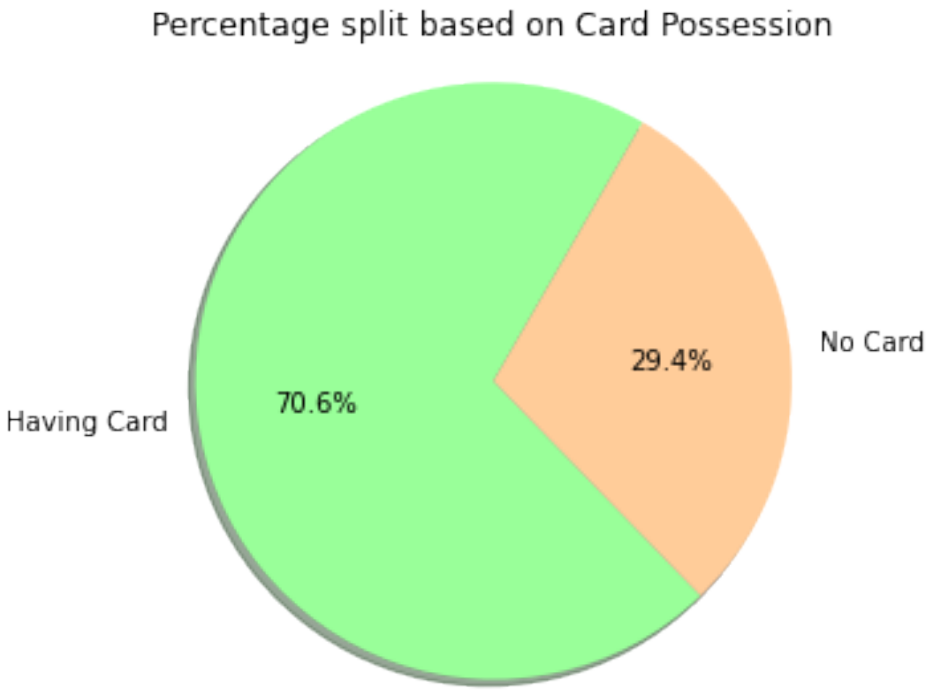




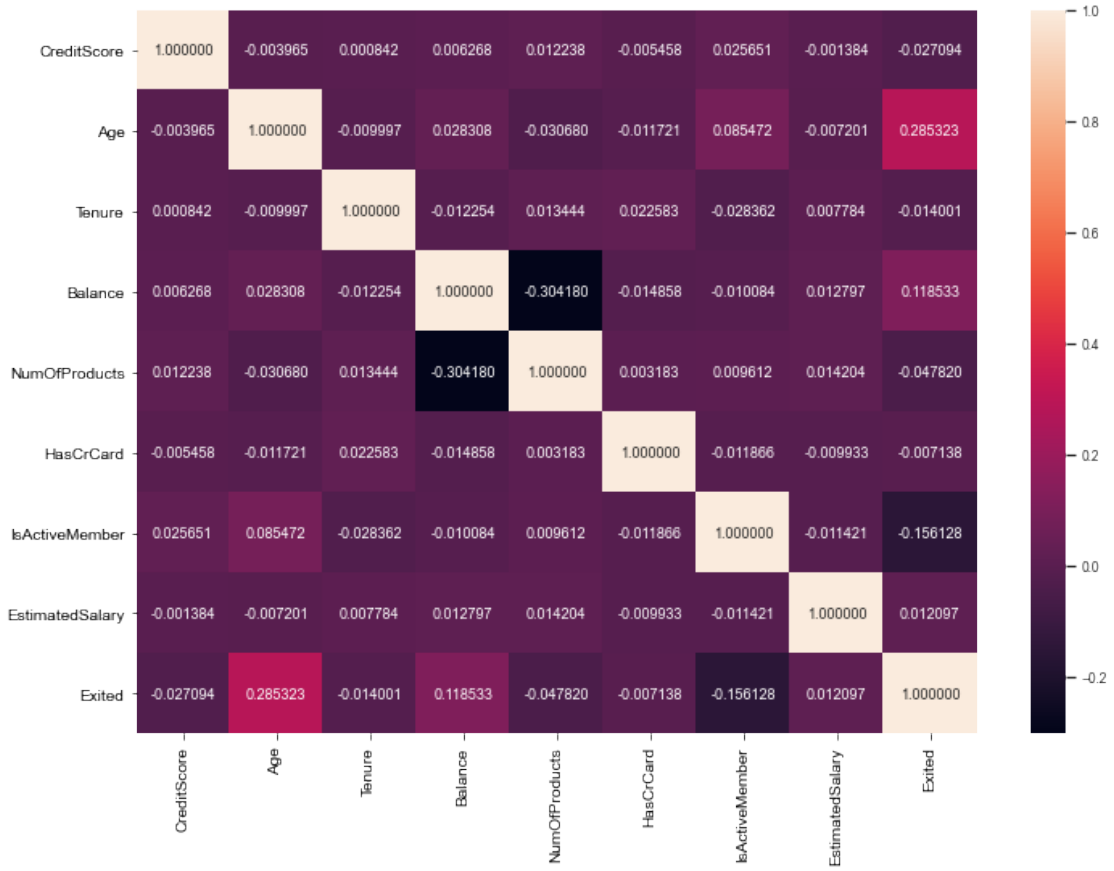

```
*********Customer churn rate w.r.t Gender****************
```



*********Customer churn rate w.r.t CreditCard**********

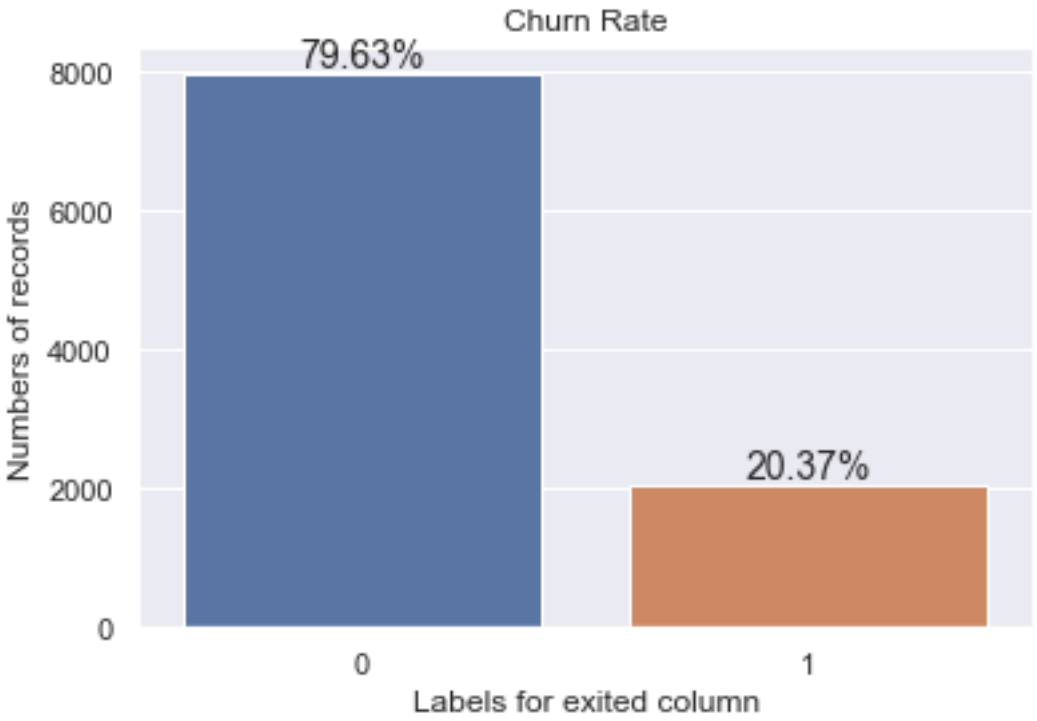



```
*********Correlation Matrix****************
```

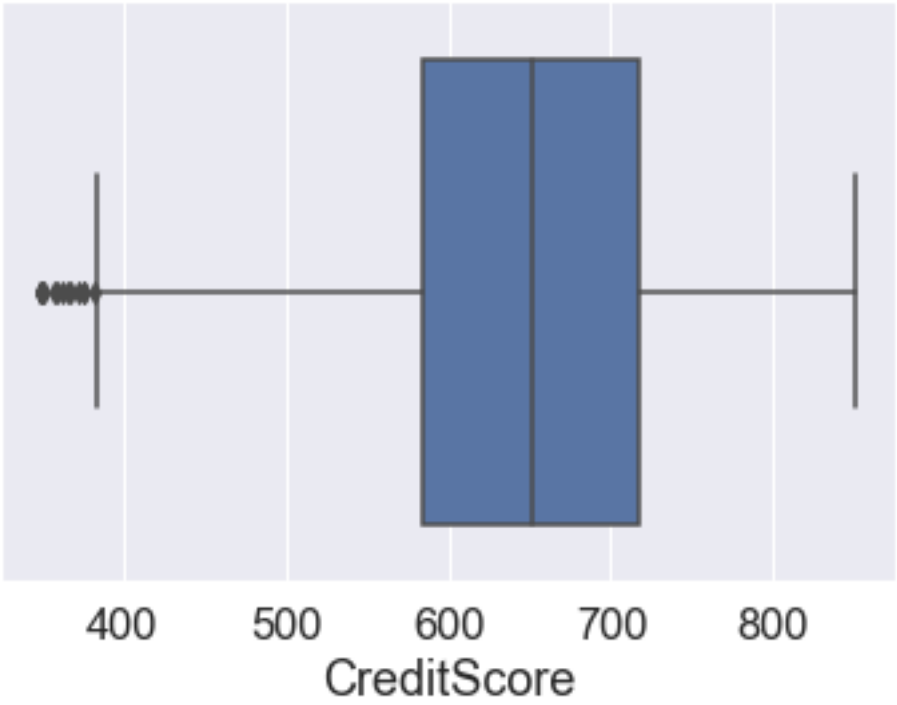


**********Churn Rate w.r.t target label -Exited**********

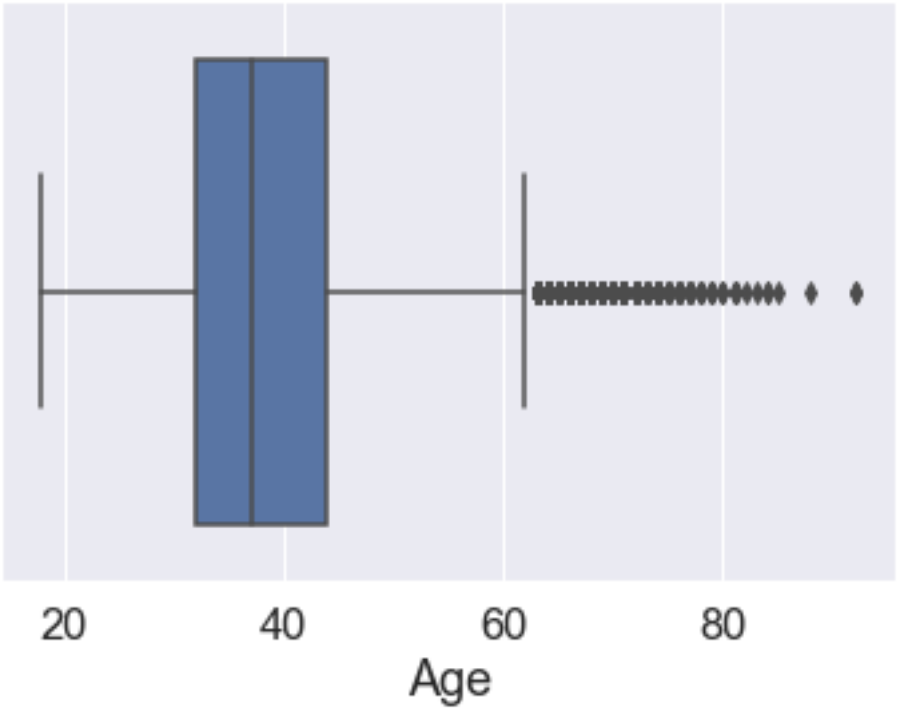
```
C:\Users\aditya.sumbaraju\Anaconda3\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variable as a keyword arg: x. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```



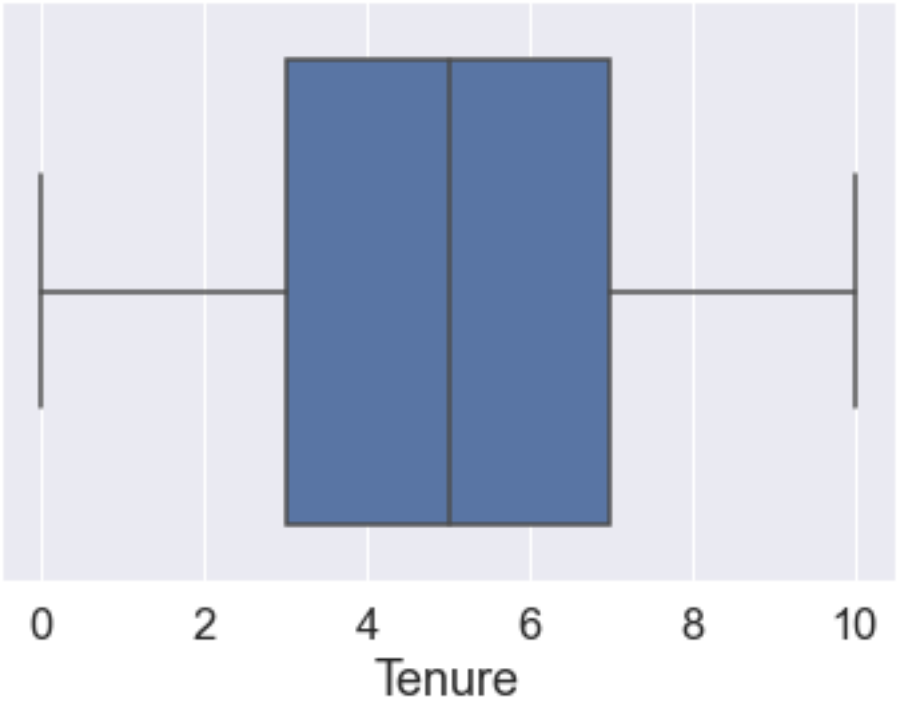

```
*********Outlier w.r.t CreditScore****************


C:\Users\aditya.sumbaraju\Anaconda3\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variable as a keyword arg: x. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```



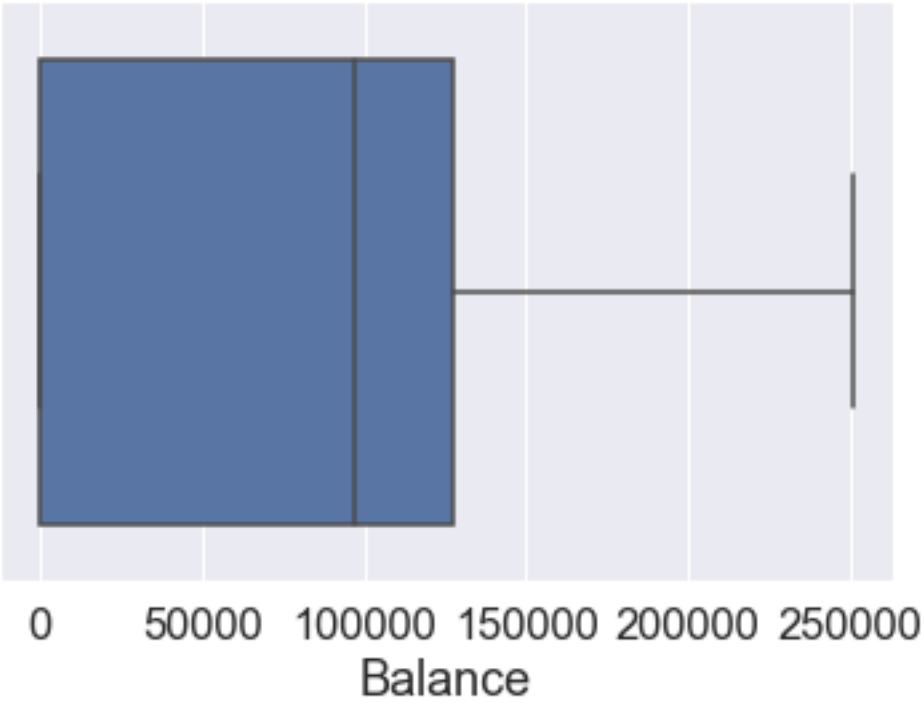

```
C:\Users\aditya.sumbaraju\Anaconda3\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variable as a keyword arg: x. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(


*********Outlier w.r.t Age****************
```



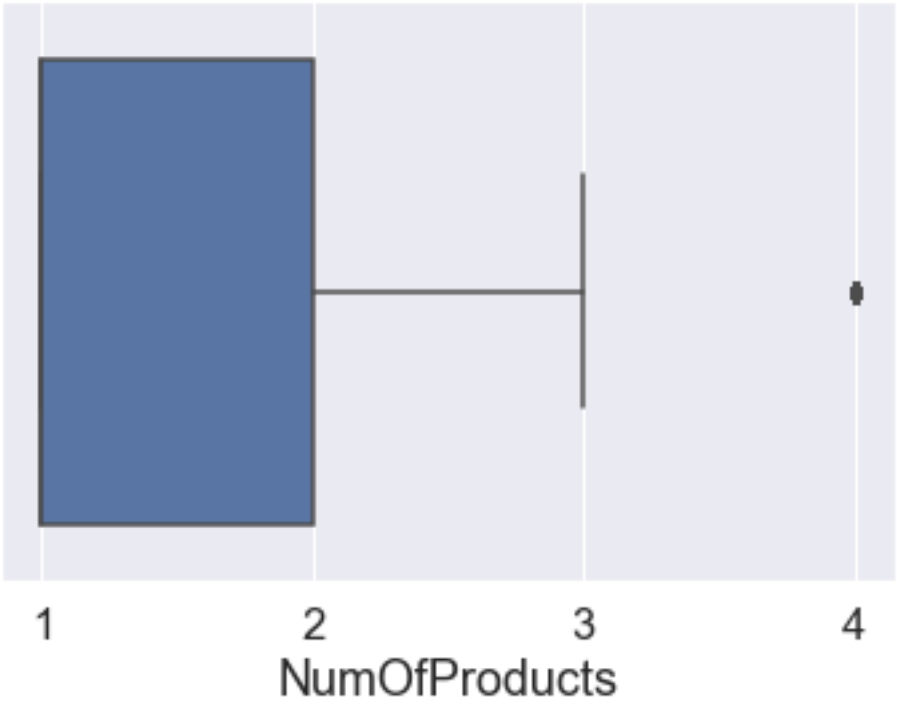

```
*********Outlier w.r.t Tenure****************


C:\Users\aditya.sumbaraju\Anaconda3\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variable as a keyword arg: x. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```



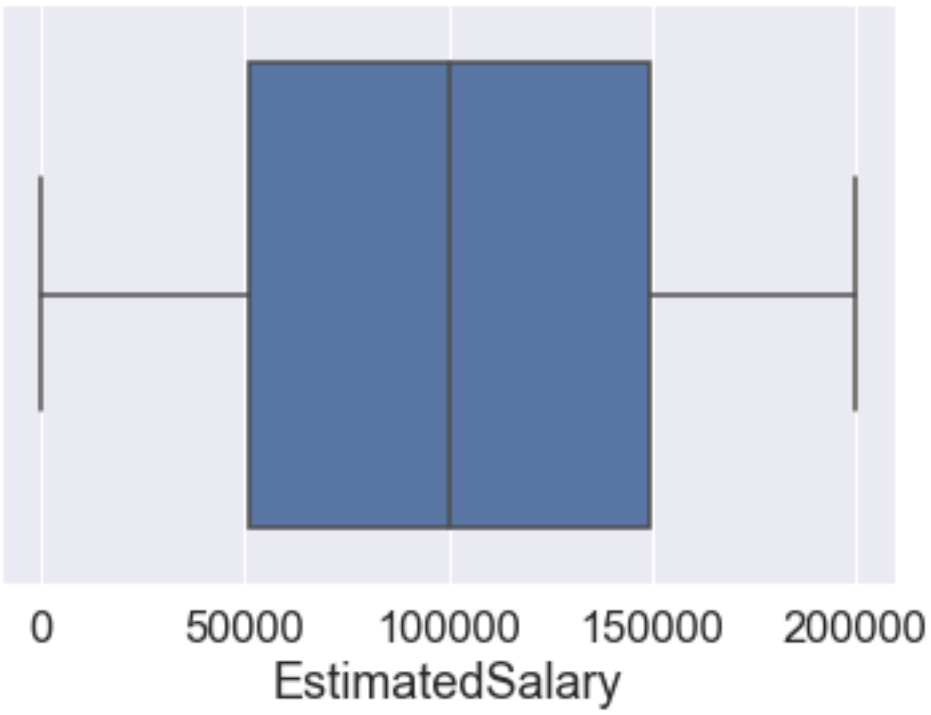

```
C:\Users\aditya.sumbaraju\Anaconda3\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variable as a keyword arg: x. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(


*********Outlier w.r.t Balance****************
```



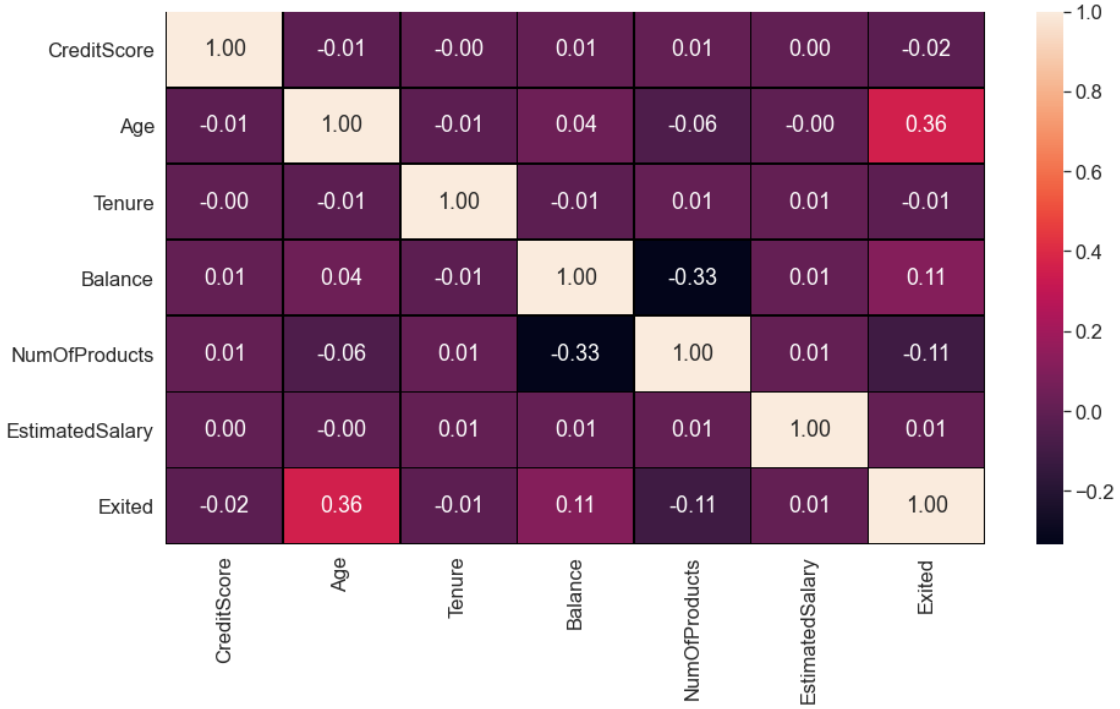

```
*********Outlier w.r.t NumOfProducts****************


C:\Users\aditya.sumbaraju\Anaconda3\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variable as a keyword arg: x. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```




```
C:\Users\aditya.sumbaraju\Anaconda3\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variable as a keyword arg: x. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(


*********Outlier w.r.t EstimatedSalary****************
```




```
(10000, 11)
(9516, 11)
```



[2]:
```
# since geography is a categorical data lets one-hot encode it by using pd.
 ↪get_dummies
ccpb_df_cleaned1 = pd.get_dummies(ccpb_df_cleaned, columns = ['Geography'])

# gender is a categorical data: label encode gender as female = 1 and male = 0
def func(data_cleaned):
    d =[]
    for m in data_cleaned:
        if m =='Female':
            d.append(1)
        else:
            d.append(0)
    return d

ccpb_df_cleaned1['Gender'] = func(ccpb_df_cleaned1['Gender'])
```

[3]:
```
ccpb_df_cleaned1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 9516 entries, 0 to 9999
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   CreditScore        9516 non-null   int64
 1   Gender             9516 non-null   int64
 2   Age                9516 non-null   int64
 3   Tenure             9516 non-null   int64
 4   Balance            9516 non-null   float64
 5   NumOfProducts      9516 non-null   int64
 6   HasCrCard          9516 non-null   int64
 7   IsActiveMember     9516 non-null   int64
 8   EstimatedSalary    9516 non-null   float64
 9   Exited             9516 non-null   int64
 10  Geography_France   9516 non-null   uint8
 11  Geography_Germany  9516 non-null   uint8
 12  Geography_Spain    9516 non-null   uint8
dtypes: float64(2), int64(8), uint8(3)
memory usage: 845.7 KB
```

1 Model Selection

[4]:
```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import plot_roc_curve
from sklearn.model_selection import train_test_split, cross_val_score,␣
 ↪GridSearchCV
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, f1_score, classification_report
```

[5]:
```
x = ccpb_df_cleaned1.drop('Exited', axis = 1)
y = ccpb_df_cleaned1['Exited']
```

[6]:
```
#splitting data into test and train set
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size = 0.
↪3,random_state = None)
```

2 1. Logistic Regression

[42]:
```
sc = StandardScaler()
xstandard_train = sc.fit_transform(x_train)
xstandard_test = sc.transform (x_test)

m1_lr = LogisticRegression()
m1_lr.fit(xstandard_train, y_train)

lr_pred = m1_lr.predict(xstandard_test)

print(classification_report(y_test, lr_pred, digits=2))
lr_score = accuracy_score(y_test, lr_pred)
# evaluate the test data using accuracy score
print("Model Accuracy score: Logistic Regression  ", lr_score)
```

```
              precision    recall  f1-score   support

           0       0.85      0.96      0.90      2294
           1       0.64      0.30      0.41       561

    accuracy                           0.83      2855
   macro avg       0.74      0.63      0.66      2855
weighted avg       0.81      0.83      0.80      2855

Model Accuracy score: Logistic Regression   0.8294220665499125
```

3 2. SVC

[41]:
```
m2_svc = SVC(probability = True)
m2_svc.fit(xstandard_train, y_train)

svc_pred = m2_svc.predict(xstandard_test)

print(classification_report(y_test, svc_pred, digits=2))
svc_score = accuracy_score(y_test, svc_pred)
print("Model Accuracy score: Support Vector Classification ", svc_score)
```

```
              precision    recall  f1-score   support

           0       0.86      0.97      0.91      2294
           1       0.77      0.36      0.49       561

    accuracy                           0.85      2855
   macro avg       0.82      0.67      0.70      2855
weighted avg       0.84      0.85      0.83      2855

Model Accuracy score: Support Vector Classification  0.8539404553415061
```

4 3. Random Forest Classifier

[40]:
```
m3_rfc = RandomForestClassifier(random_state = 42)

m3_rfc.fit(x_train, y_train)
rfc_pred = m3_rfc.predict(x_test)

print(classification_report(y_test, rfc_pred, digits=2))
rfc_score = accuracy_score(y_test, rfc_pred)
print("Model Accuracy score: Random Forest Classifier: ", rfc_score)
```

```
              precision    recall  f1-score   support

           0       0.87      0.96      0.92      2294
           1       0.74      0.42      0.54       561

    accuracy                           0.86      2855
   macro avg       0.81      0.69      0.73      2855
weighted avg       0.85      0.86      0.84      2855

Model Accuracy score: Random Forest Classifier:  0.8574430823117338
```

5 4. KNN

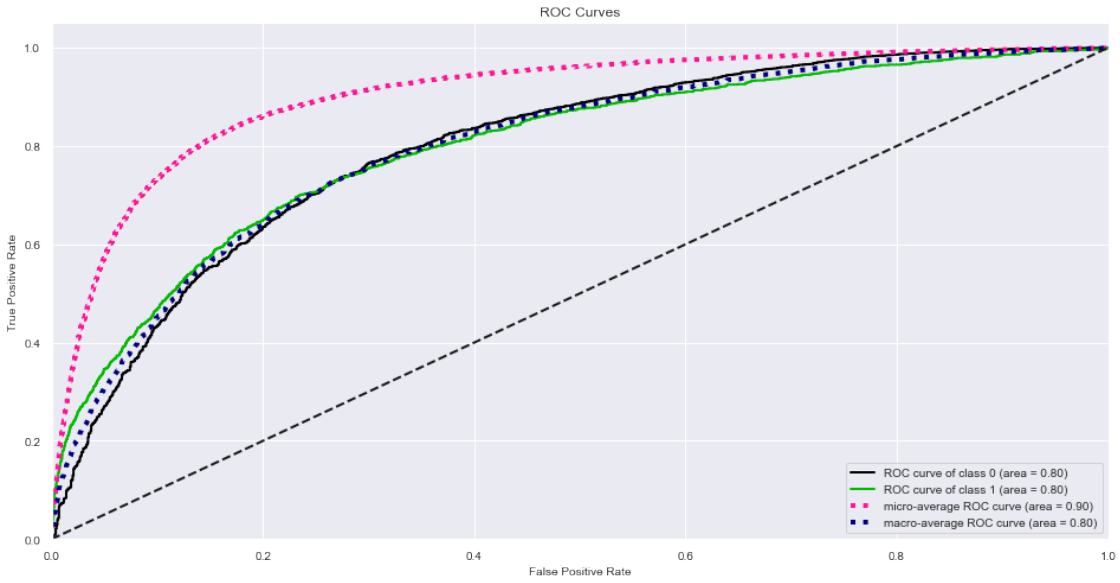
[39]:
```
m4_knn = KNeighborsClassifier(n_neighbors = 5) # algorithm instantiation
m4_knn.fit(xstandard_train, y_train)

knn_pred = m4_knn.predict(xstandard_test)

print(classification_report(y_test, knn_pred, digits=2))

knn_score = accuracy_score(y_test, knn_pred)
print("Model Accuracy score: KNN ", knn_score)
```

```
              precision    recall  f1-score   support

           0       0.85      0.95      0.90      2294
           1       0.61      0.34      0.44       561

    accuracy                           0.83      2855
   macro avg       0.73      0.64      0.67      2855
weighted avg       0.81      0.83      0.81      2855

Model Accuracy score: KNN  0.826970227670753
```

6 Model Evaluation

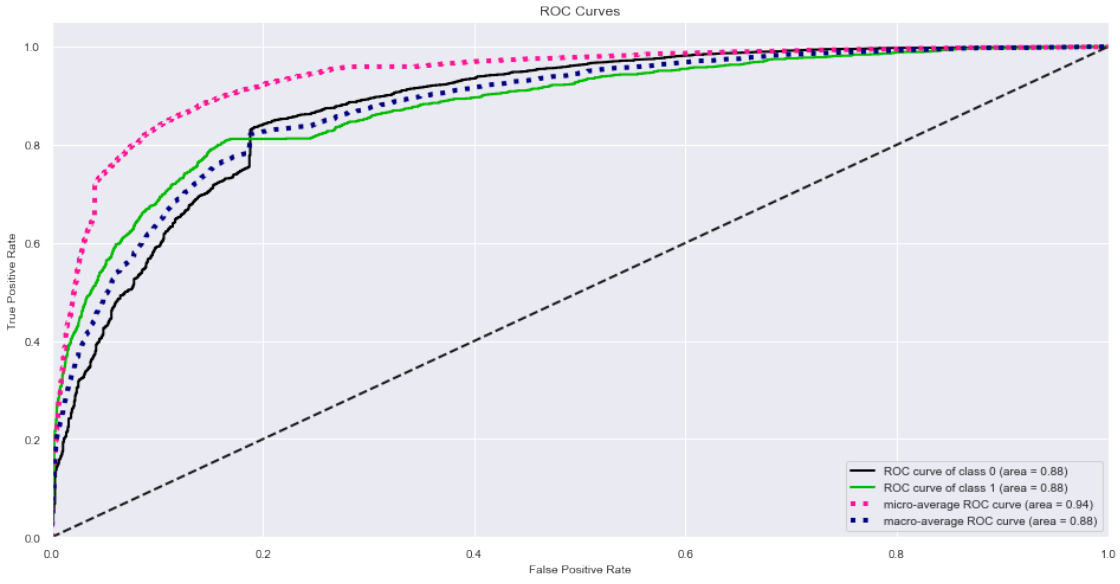
[33]:
```
# Logistic Regression
%pylab inline
pylab.rcParams['figure.figsize'] = (16,8)
import scikitplot as skplt
lr_y_probas = m1_lr.predict_proba(xstandard_train)
skplt.metrics.plot_roc(y_train, lr_y_probas)
plt.show()
```

```
Populating the interactive namespace from numpy and matplotlib
```



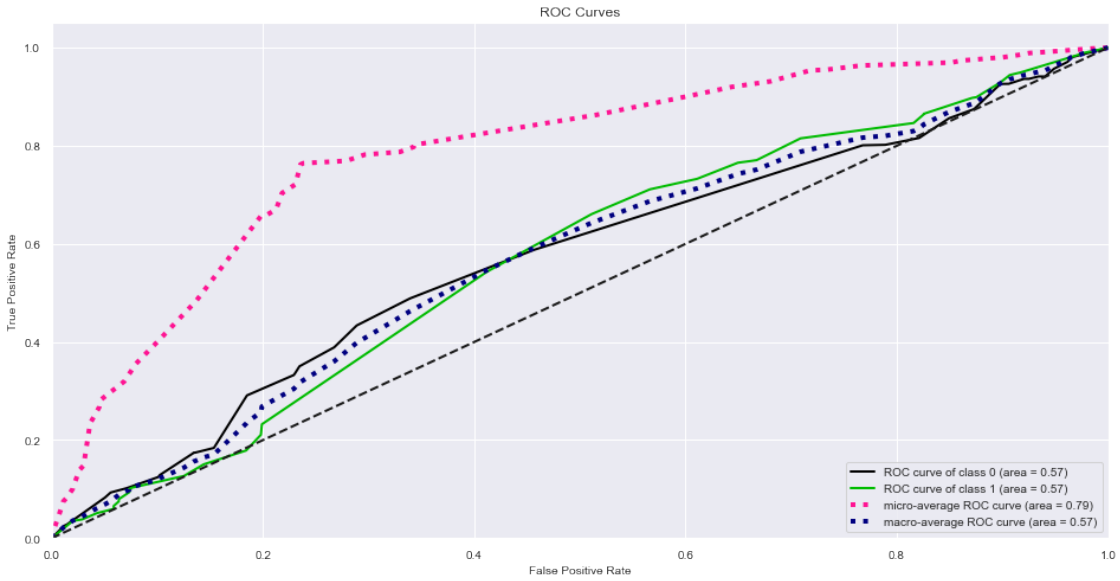
[34]:
```
# SVC
%pylab inline
pylab.rcParams['figure.figsize'] = (16,8)
svc_y_probas = m2_svc.predict_proba(xstandard_train)
skplt.metrics.plot_roc(y_train, svc_y_probas)
plt.show()
```

```
Populating the interactive namespace from numpy and matplotlib
```



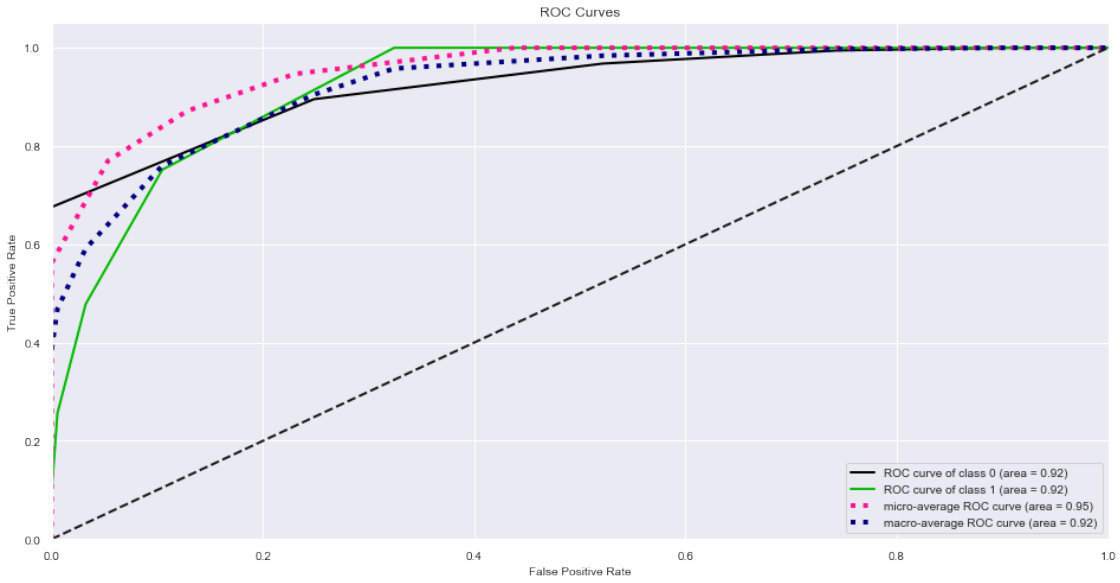
[35]:
```
# Random Forest
%pylab inline
pylab.rcParams['figure.figsize'] = (16,8)
rfc_y_probas = m3_rfc.predict_proba(xstandard_train)
skplt.metrics.plot_roc(y_train, rfc_y_probas)
plt.show()
```

```
Populating the interactive namespace from numpy and matplotlib
```



[32]:
```
# KNN
%pylab inline
pylab.rcParams['figure.figsize'] = (16,8)
knn_y_probas = m4_knn.predict_proba(xstandard_train)
skplt.metrics.plot_roc(y_train, knn_y_probas)
plt.show()
```

```
Populating the interactive namespace from numpy and matplotlib
```



[31]:
```
model_data =[['Random Forest Classifier', rfc_score],
             ['SVM Classifier', svc_score],
             ['Logistic Regression', lr_score],
             ['KNN Classifier', knn_score]]

indexes = [1,2,3,4]
columns_name = ['MODEL', 'ACCURACY_SCORE']
acc_result = pd.DataFrame(data = model_data,index = indexes, columns =␣
 ↪columns_name )
display (acc_result)
```

	MODEL	ACCURACY_SCORE
1	Random Forest Classifier	0.857443
2	SVM Classifier	0.853940
3	Logistic Regression	0.829422
4	KNN Classifier	0.826970

[ ]: