Original Analysis Case Study

Part 1 : Graphics Analysis

Part 2 : Feature Reduction (Extraction/Selection)

Part 3 : Model Selection and Evaluation

Part 1 : Graphics Analysis

In this case study, as part of phase I, we will perform exploratory data analysis by graphing the features in the dataset.

The dataset is composed of 10,000 customer's record at a bank. The dataset has a total of 14 features 13 of which can be considered as independent variables and 1 as the dependent variable. The goal is to build a model that can predict whether a customer is likely to stay or exit the bank. The model will predict the dependent variable 'Exited' using the approrpiate set of independent variables 'CreditScore','Geography','Gender','Age','Tenure','Balance','NumberOfProducts','HasCrCard', and 'IsActiveMember'.

We will perform model selection and model validation exercises and use the model the make the desired prediction. The accuracy and percision of the model will be analyzed in the next phases of the study.

In [101]:
```
# Load Libraries
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import xlrd
```

In [102]:
```
#Step 1:  Load data into a dataframe
DataFile = "Data/BankCustomers.xlsx"

data = pd.read_excel(DataFile)
```

In [103]:
```
# Step 2:  check the dimension of the table
print("The dimension of the table is: ", data.shape)
```

```
The dimension of the table is:  (10000, 14)
```

In [104]:
```python
#Step 3:  Look at the data
print(data.head(5))
```

```
   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
```

In [105]:
```python
#Step 5:  what type of variables are in the table
print("Describe Data")
print(data.describe())
```

```
Describe Data
         RowNumber    CustomerId   CreditScore           Age        Tenur
e  \
count  10000.00000  1.000000e+04  10000.000000  10000.000000  10000.00000
0
mean    5000.50000  1.569094e+07    650.528800     38.921800      5.01280
0
std     2886.89568  7.193619e+04     96.653299     10.487806      2.89217
4
min        1.00000  1.556570e+07    350.000000     18.000000      0.00000
0
25%     2500.75000  1.562853e+07    584.000000     32.000000      3.00000
0
50%     5000.50000  1.569074e+07    652.000000     37.000000      5.00000
0
75%     7500.25000  1.575323e+07    718.000000     44.000000      7.00000
0
max    10000.00000  1.581569e+07    850.000000     92.000000     10.00000
0

             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
```

In [106]:
```python
# Step 6a: Summary of object type data
print("Summarized Data")
print(data.describe(include=['O']))
```

```
Summarized Data
       Surname Geography Gender
count    10000     10000  10000
unique    2932         3      2
top      Smith    France   Male
freq        32      5014   5457
```

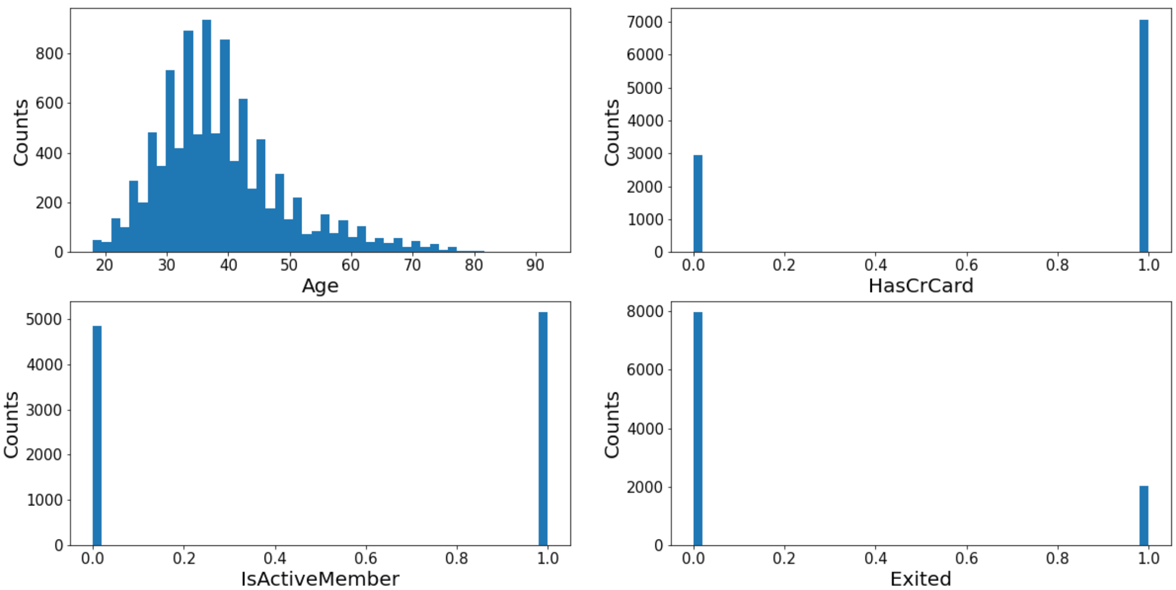
In [107]:
```python
# Step 6b: Summary of numeric type data
print("Summarized Data")
print(data.describe(include=np.number))
```

```
Summarized Data
         RowNumber    CustomerId   CreditScore           Age         Tenur
e  \
count  10000.00000  1.000000e+04  10000.000000  10000.000000  10000.00000
0
mean    5000.50000  1.569094e+07    650.528800     38.921800      5.01280
0
std     2886.89568  7.193619e+04     96.653299     10.487806      2.89217
4
min        1.00000  1.556570e+07    350.000000     18.000000      0.00000
0
25%     2500.75000  1.562853e+07    584.000000     32.000000      3.00000
0
50%     5000.50000  1.569074e+07    652.000000     37.000000      5.00000
0
75%     7500.25000  1.575323e+07    718.000000     44.000000      7.00000
0
max    10000.00000  1.581569e+07    850.000000     92.000000     10.00000
0

             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
```

Histogram of ['Age', 'HasCrCard', 'IsActiveMember', 'Exited']

In [108]:

```
# set up the figure size
plt.rcParams['figure.figsize'] = (20, 10)

# make subplots
fig, axes = plt.subplots(nrows = 2, ncols = 2)

# Specify the features of interest
num_features = ['Age', 'HasCrCard', 'IsActiveMember', 'Exited']
xaxes = num_features
yaxes = ['Counts', 'Counts', 'Counts', 'Counts']

# draw histograms
axes = axes.ravel()
for idx, ax in enumerate(axes):
    ax.hist(data[num_features[idx]].dropna(), bins=50)
    ax.set_xlabel(xaxes[idx], fontsize=20)
    ax.set_ylabel(yaxes[idx], fontsize=20)
    ax.tick_params(axis='both', labelsize=15)
plt.show()
```

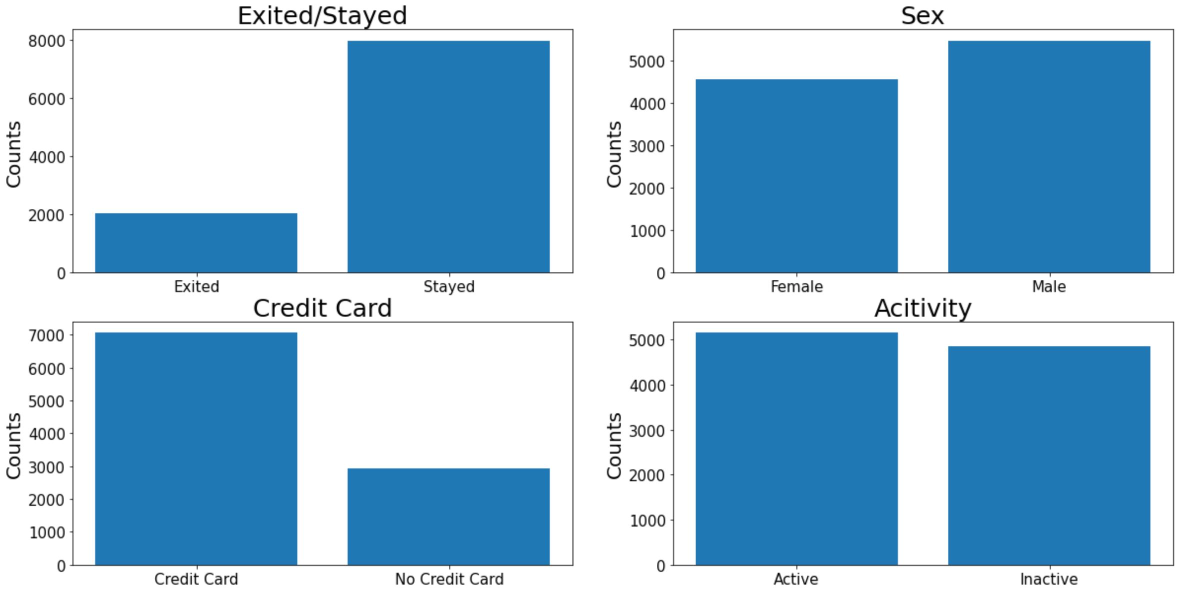


Barchart comparing the number of:

- **Exits vs stays**
- **Males vs. Female**
- **Has credit card vs does not have credit card**
- **active members vs inactive members**

In [109]:

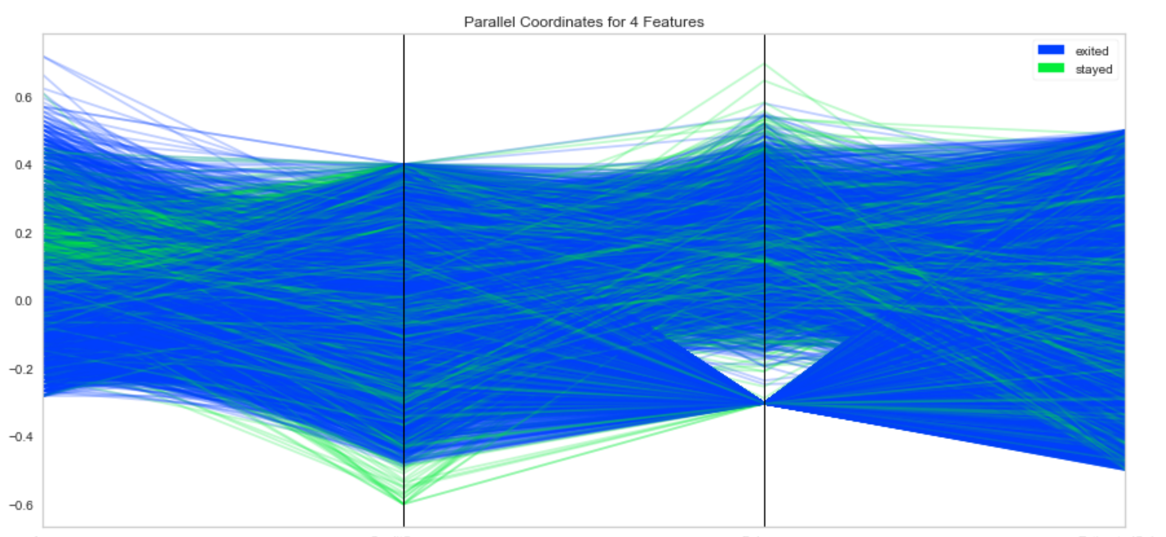
```
# make subplots
fig, axes = plt.subplots(nrows = 2, ncols = 2)

# make the data read to feed into the visulizer
X_Exited = data.replace({'Exited': {1: 'Exited', 0: 'Stayed'}}).groupby(
'Exited').size().reset_index(name='Counts')['Exited']
Y_Exited = data.replace({'Exited': {1: 'Exited', 0: 'Stayed'}}).groupby(
'Exited').size().reset_index(name='Counts')['Counts']
# make the bar plot
axes[0,0].bar(X_Exited, Y_Exited)
axes[0,0].set_title('Exited/Stayed', fontsize=25)
axes[0,0].set_ylabel('Counts', fontsize=20)
axes[0,0].tick_params(axis='both', labelsize=15)

# make the data read to feed into the visulizer
X_Sex = data.groupby('Gender').size().reset_index(name='Counts')['Gender'
]
Y_Sex = data.groupby('Gender').size().reset_index(name='Counts')['Counts'
]
# make the bar plot
axes[0,1].bar(X_Sex, Y_Sex)
axes[0,1].set_title('Sex', fontsize=25)
axes[0,1].set_ylabel('Counts', fontsize=20)
axes[0,1].tick_params(axis='both', labelsize=15)

X_HasCrCard = data.replace({'HasCrCard': {1: 'Credit Card', 0: 'No Credit
Card'}}).groupby('HasCrCard').size().reset_index(name='Counts')['HasCrCar
d']
Y_HasCrCard = data.replace({'HasCrCard': {1: 'Credit Card', 0: 'No Credit
Card'}}).groupby('HasCrCard').size().reset_index(name='Counts')['Counts']
# make the bar plot
axes[1,0].bar(X_HasCrCard, Y_HasCrCard)
axes[1,0].set_title('Credit Card', fontsize=25)
axes[1,0].set_ylabel('Counts', fontsize=20)
axes[1,0].tick_params(axis='both', labelsize=15)

X_IsActive = data.replace({'IsActiveMember': {1: 'Active', 0: 'Inactive'
}}).groupby('IsActiveMember').size().reset_index(name='Counts')['IsActive
Member']
Y_IsActive = data.replace({'IsActiveMember': {1: 'Active', 0: 'Inactive'
}}).groupby('IsActiveMember').size().reset_index(name='Counts')['Counts']
# make the bar plot
axes[1,1].bar(X_IsActive, Y_IsActive)
axes[1,1].set_title('Acitivity', fontsize=25)
axes[1,1].set_ylabel('Counts', fontsize=20)
axes[1,1].tick_params(axis='both', labelsize=15)
```



Parallel Coordinate graphe comparing ['Age', 'CreditScore', 'Balance', 'EstimatedSalary']

In [112]:

```
# Step 9:  Compare variables against those who stayed and those who exite
d
#set up the figure size
%matplotlib inline
plt.rcParams['figure.figsize'] = (15, 7)
plt.rcParams['font.size'] = 50

# setup the color for yellowbrick visulizer
from yellowbrick.style import set_palette
set_palette('sns_bright')

# import packages
from yellowbrick.features import ParallelCoordinates
# Specify the features of interest and the classes of the target
classes = ['exited', 'stayed']
num_features = ['Age', 'CreditScore', 'Balance', 'EstimatedSalary']

# copy data to a new dataframe
data_norm = data.copy()
# normalize data to 0-1 range
for feature in num_features:
    data_norm[feature] = (data[feature] - data[feature].mean(skipna=True
)) / (data[feature].max(skipna=True) - data[feature].min(skipna=True))

# Extract the numpy arrays from the data frame
X = data_norm[num_features].values
y = data.Exited.values

# Instantiate the visualizer
# Instantiate the visualizer
visualizer = ParallelCoordinates(classes=classes, features=num_features)


visualizer.fit(X, y)      # Fit the data to the visualizer
visualizer.transform(X)   # Transform the data
visualizer.poof(outpath="images/pcoords2.png") # Draw/show/poof the data
plt.show();
```



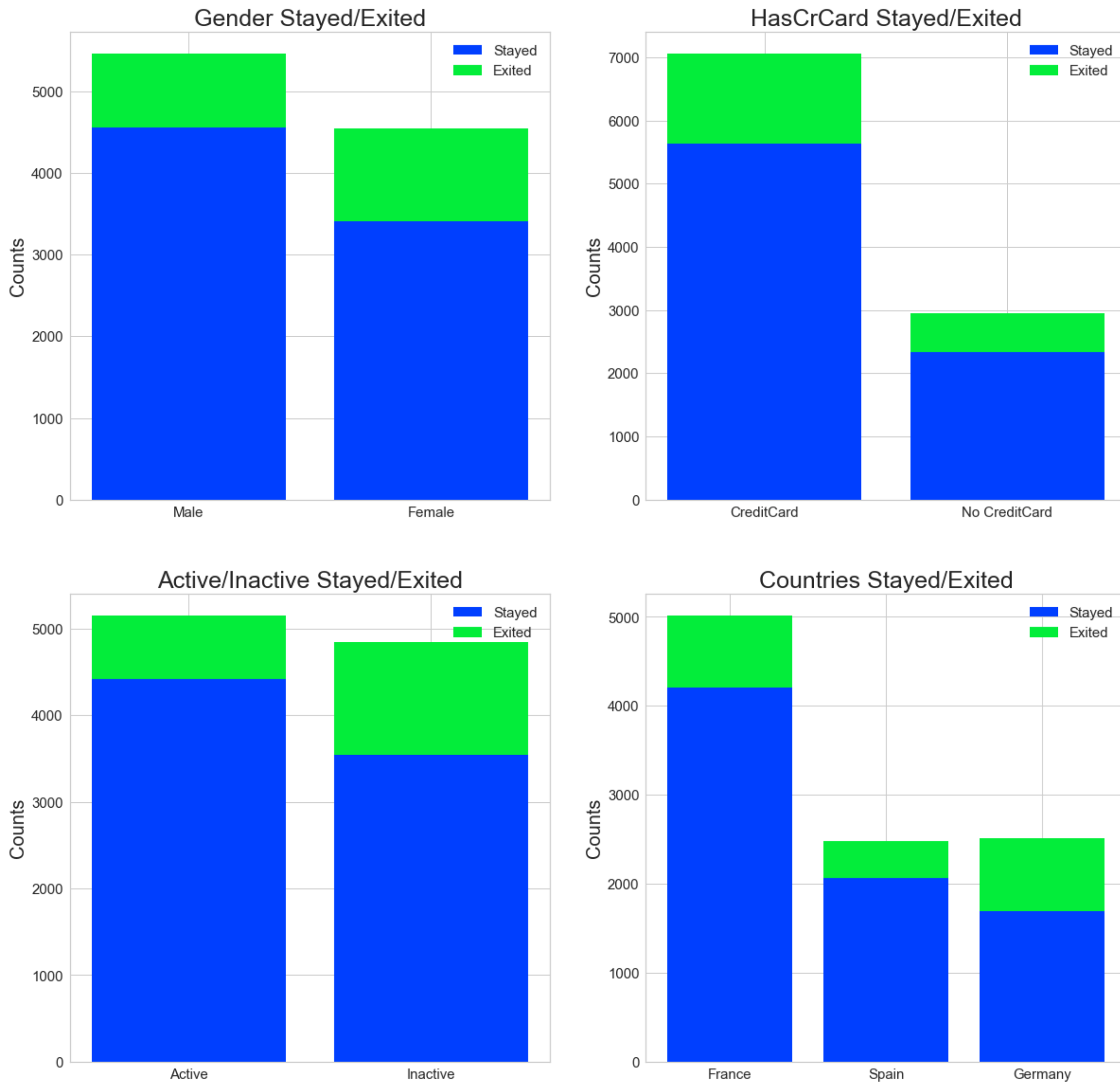
Stacked bar charts showing stays and exits based on:

- **Gender**
- **Has Credit card**
- **banking activity**
- **gegraphic location(Country)**

In [113]:

```
# Step 10 - stacked bar chart to compare Gender exit/stay numbers
#set up the figure size
%matplotlib inline
plt.rcParams['figure.figsize'] = (20, 20)

# make subplots
fig, axes = plt.subplots(nrows = 2, ncols = 2)

# make the data read to feed into the visulizer
Gender_Stayed = data.replace({'Exited': {1: 'Exited', 0: 'Stayed'}})[data
['Exited']==0]['Gender'].value_counts()
Gender_Exited = data.replace({'Exited': {1: 'Exited', 0: 'Stayed'}})[data
['Exited']==1]['Gender'].value_counts()
Gender_Exited = Gender_Exited.reindex(index = Gender_Stayed.index)
# make the bar plot
p1 = axes[0, 0].bar(Gender_Stayed.index, Gender_Stayed.values)
p2 = axes[0, 0].bar(Gender_Exited.index, Gender_Exited.values, bottom=Gen
der_Stayed.values)
axes[0, 0].set_title('Gender Stayed/Exited', fontsize=25)
axes[0, 0].set_ylabel('Counts', fontsize=20)
axes[0, 0].tick_params(axis='both', labelsize=15)
axes[0, 0].legend((p1[0], p2[0]), ('Stayed', 'Exited'), fontsize = 15)

# make the data read to feed into the visulizer
HasCrCard_Stayed = data.replace({'Exited': {1: 'Exited', 0: 'Stayed'}})[d
ata['Exited']==0]
HasCrCard_Stayed = HasCrCard_Stayed.replace({'HasCrCard': {1: 'CreditCar
d', 0: 'No CreditCard'}})['HasCrCard'].value_counts()

HasCrCard_Exited = data.replace({'Exited': {1: 'Exited', 0: 'Stayed'}})[d
ata['Exited']==1]
HasCrCard_Exited = HasCrCard_Exited.replace({'HasCrCard': {1: 'CreditCar
d', 0: 'No CreditCard'}})['HasCrCard'].value_counts()
HasCrCard_Exited = HasCrCard_Exited.reindex(index = HasCrCard_Stayed.inde
x)
# make the bar plot
p3 = axes[0, 1].bar(HasCrCard_Stayed.index, HasCrCard_Stayed.values)
p4 = axes[0, 1].bar(HasCrCard_Exited.index, HasCrCard_Exited.values, bott
om=HasCrCard_Stayed.values)
axes[0, 1].set_title('HasCrCard Stayed/Exited', fontsize=25)
axes[0, 1].set_ylabel('Counts', fontsize=20)
axes[0, 1].tick_params(axis='both', labelsize=15)
axes[0, 1].legend((p3[0], p4[0]), ('Stayed', 'Exited'), fontsize = 15)

# make the data read to feed into the visulizer
IsActive_Stayed = data.replace({'Exited': {1: 'Exited', 0: 'Stayed'}})[da
ta['Exited']==0]
IsActive_Stayed = IsActive_Stayed.replace({'IsActiveMember': {1: 'Active'
, 0: 'Inactive'}})['IsActiveMember'].value_counts()

IsActive_Exited = data.replace({'Exited': {1: 'Exited', 0: 'Stayed'}})[da
ta['Exited']==1]
IsActive_Exited = IsActive_Exited.replace({'IsActiveMember': {1: 'Active'
, 0: 'Inactive'}})['IsActiveMember'].value_counts()
IsActive_Exited = IsActive_Exited.reindex(index = IsActive_Stayed.index)
# make the bar plot
```

```
p4 = axes[1,0].bar(IsActive_Stayed.index, IsActive_Stayed.values)
p5 = axes[1,0].bar(IsActive_Exited.index, IsActive_Exited.values, bottom=
IsActive_Stayed.values)
axes[1,0].set_title('Active/Inactive Stayed/Exited', fontsize=25)
axes[1,0].set_ylabel('Counts', fontsize=20)
axes[1,0].tick_params(axis='both', labelsize=15)
axes[1,0].legend((p4[0], p5[0]), ('Stayed', 'Exited'), fontsize = 15)


# make the data read to feed into the visulizer
Country_Stayed = data.replace({'Exited': {1: 'Exited', 0: 'Stayed'}})[dat
a['Exited']==0]['Geography'].value_counts()

Country_Exited = data.replace({'Exited': {1: 'Exited', 0: 'Stayed'}})[dat
a['Exited']==1]['Geography'].value_counts()
Country_Exited = Country_Exited.reindex(index = Country_Stayed.index)
# make the bar plot
p6 = axes[1,1].bar(Country_Stayed.index, Country_Stayed.values)
p7 = axes[1,1].bar(Country_Exited.index, Country_Exited.values, bottom=Co
untry_Stayed.values)
axes[1,1].set_title('Countries Stayed/Exited', fontsize=25)
axes[1,1].set_ylabel('Counts', fontsize=20)
axes[1,1].tick_params(axis='both', labelsize=15)
axes[1,1].legend((p6[0], p7[0]),('Stayed', 'Exited'), fontsize = 15)
plt.show()
```



Part 2 : Feature Reduction (Extraction/Selection)

In [114]:

```
#Step 1:  Load data into a dataframe
DataFile = "Data/BankCustomers.xlsx"

data = pd.read_excel(DataFile)
```

In [115]:

```
data.columns
```

Out[115]:

```
Index(['RowNumber', 'CustomerId', 'Surname', 'CreditScore', 'Geography',
       'Gender', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCar
d',
       'IsActiveMember', 'EstimatedSalary', 'Exited'],
      dtype='object')
```

In [116]:

```
# Step 11- remove unrelated columns
data = data.drop(['RowNumber','CustomerId','Surname'],axis=1)
```

In [117]:
```python
data.columns
```

Out[117]:
```
Index(['CreditScore', 'Geography', 'Gender', 'Age', 'Tenure', 'Balance',
       'NumOfProducts', 'HasCrCard', 'IsActiveMember', 'EstimatedSalary',
       'Exited'],
      dtype='object')
```

In [118]:
```python
# Step 12 - Onehot code Geography
from sklearn.preprocessing import LabelBinarizer, MultiLabelBinarizer

feature = np.array(data['Geography'])
one_hot = LabelBinarizer()

one_hot.fit_transform(feature)
```

Out[118]:
```
array([[1, 0, 0],
       [0, 0, 1],
       [1, 0, 0],
       ...,
       [1, 0, 0],
       [0, 1, 0],
       [1, 0, 0]])
```

In [119]:
```python
one_hot.classes_
```

Out[119]:
```
array(['France', 'Germany', 'Spain'], dtype='<U7')
```

In [120]:
```python
dummies = pd.get_dummies(feature)
dummies.head()
```

Out[120]:

	France	Germany	Spain
0	1	0	0
1	0	0	1
2	1	0	0
3	1	0	0
4	0	0	1

In [121]:
```python
# Drop Geography column
data = data.drop(['Geography'],axis=1)
```

In [122]:
```
# Add dummies
data[dummies.columns] = dummies
data.head()
```

Out[122]:

	CreditScore	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActi
0	619	Female	42	2	0.00	1	1	
1	608	Female	41	1	83807.86	1	0	
2	502	Female	42	8	159660.80	3	1	
3	699	Female	39	1	0.00	2	0	
4	850	Female	43	2	125510.82	1	1	

In [123]:
```
# one-hot code Gender
feature = np.array(data['Gender'])
one_hot = LabelBinarizer()

one_hot.fit_transform(feature)
dummies = pd.get_dummies(feature)
dummies
```

Out[123]:

	Female	Male
0	1	0
1	1	0
2	1	0
3	1	0
4	1	0
...	...	...
9995	0	1
9996	0	1
9997	1	0
9998	0	1
9999	1	0

10000 rows × 2 columns

In [124]:

```
 #drop Gender and add dummies
data = data.drop(['Gender'],axis=1)
data[dummies.columns] = dummies
data.head()
```

Out[124]:

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMembe
0	619	42	2	0.00	1	1	
1	608	41	1	83807.86	1	0	
2	502	42	8	159660.80	3	1	
3	699	39	1	0.00	2	0	
4	850	43	2	125510.82	1	1	

In [125]:

```
# Drop spain and male to avoid dummy trap
data = data.drop(['Male','Spain'],axis=1)
data.head()
```

Out[125]:

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMembe
0	619	42	2	0.00	1	1	
1	608	41	1	83807.86	1	0	
2	502	42	8	159660.80	3	1	
3	699	39	1	0.00	2	0	
4	850	43	2	125510.82	1	1	

In [126]:

```
# Move the dependent variable column to the last position.

Exited = data.replace({'Exited': {1: 'Existed', 0: 'Stayed'}})['Exited']
```

In [127]:

```
data = data.drop(['Exited'],axis=1)
data['Exited'] = Exited
data.head()
```

Out[127]:

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMembe
0	619	42	2	0.00	1	1	
1	608	41	1	83807.86	1	0	
2	502	42	8	159660.80	3	1	
3	699	39	1	0.00	2	0	
4	850	43	2	125510.82	1	1	

In [128]:
```python
# Step 13 - Set up independet variable and depndent variables and perform feature reduction
Independents = data.iloc[:, :-1].values
print(type(Independents))
Dependent = data.iloc[:,-1].values
print(Dependent)
X = Independents
y = Dependent
```

```
<class 'numpy.ndarray'>
['Existed' 'Stayed' 'Existed' ... 'Existed' 'Existed' 'Stayed']
```

In [129]:
```python
data.shape
```

Out[129]:
```
(10000, 12)
```

In [130]:
```python
X.shape
```

Out[130]:
```
(10000, 11)
```

In [131]:
```python
y.shape
```

Out[131]:
```
(10000,)
```

In [132]:
```python
# Attempt at feature reduction using PCA Before feature scaling
#Load libraries
from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA

# Create a PCA that will retain 99% of variance
pca = PCA(n_components=0.99, whiten=True)

# Conduct PCA
features_pca = pca.fit_transform(X)

# Show results
print("Original number of features:", X.shape[1])
print("Reduced number of features:", features_pca.shape[1])
```

```
Original number of features: 11
Reduced number of features: 2
```

In [133]:
```
# Feature scaling will normalize all variable to the same scale
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X = sc.fit_transform(X)
print(X)
```

```
[[-0.32622142  0.29351742 -1.04175968 ...  0.99720391 -0.57873591
   1.09598752]
 [-0.44003595  0.19816383 -1.38753759 ... -1.00280393 -0.57873591
   1.09598752]
 [-1.53679418  0.29351742  1.03290776 ...  0.99720391 -0.57873591
   1.09598752]
 ...
 [ 0.60498839 -0.27860412  0.68712986 ...  0.99720391 -0.57873591
   1.09598752]
 [ 1.25683526  0.29351742 -0.69598177 ... -1.00280393  1.72790383
  -0.91241915]
 [ 1.46377078 -1.04143285 -0.35020386 ...  0.99720391 -0.57873591
   1.09598752]]
```

In [134]:
```
# Attempt at feature reduction using PCA After feature scaling
#Load libraries
from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA

# Create a PCA that will retain 99% of variance
pca = PCA(n_components=0.99, whiten=True)

# Conduct PCA
features_pca = pca.fit_transform(X)

# Show results
print("Original number of features:", X.shape[1])
print("Reduced number of features:", features_pca.shape[1])
```

```
Original number of features: 11
Reduced number of features: 11
```

Part 3 : Model Selection and Evaluation

Summary of parts1 1 and 2: We have performed feature reduction and scaled the independent variables. The X and y variables are the independent variables dataset and the dependent variables respectively. The value of 0 or 1 for the depended variable has been converted to 'Stayed' and 'Exited" respectively in anticipation of using logistic regression classifier for modeling.

In [135]:
```
import pandas as pd
import yellowbrick

import warnings
warnings.filterwarnings("ignore")
```

In [136]:
```
print("Indpendent variables matrix:\n")
print(X)
```

```
Indpendent variables matrix:

[[-0.32622142  0.29351742 -1.04175968 ...  0.99720391 -0.57873591
   1.09598752]
 [-0.44003595  0.19816383 -1.38753759 ... -1.00280393 -0.57873591
   1.09598752]
 [-1.53679418  0.29351742  1.03290776 ...  0.99720391 -0.57873591
   1.09598752]
 ...
 [ 0.60498839 -0.27860412  0.68712986 ...  0.99720391 -0.57873591
   1.09598752]
 [ 1.25683526  0.29351742 -0.69598177 ... -1.00280393  1.72790383
  -0.91241915]
 [ 1.46377078 -1.04143285 -0.35020386 ...  0.99720391 -0.57873591
   1.09598752]]
```

In [137]:
```
print("Dependent variable array:\n")
print(y)
```

```
Dependent variable array:

['Existed' 'Stayed' 'Existed' ... 'Existed' 'Existed' 'Stayed']
```

Step 14 - Split the dataset to 30% test set and 70% training dataset

In [138]:
```
from sklearn.model_selection import train_test_split



X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =0.3,
random_state=0)

# number of samples in each set
print("Total sample in dataset: ", X.shape[0])

print("No. of samples in training set: ", X_train.shape[0])
print("No. of samples in validation set:", X_test.shape[0])
```

```
Total sample in dataset:  10000
No. of samples in training set:  7000
No. of samples in validation set: 3000
```

In [139]:
```
print(y_train.shape)
```

```
(7000,)
```

In [145]:
```
print(y_test.shape)
```

```
(3000,)
```

In [146]:
```
# stayed and exited
print('\n')
print('No. of customer who stayed and exited in the training set:')
print(pd.Series(y_train).value_counts())
```

```

No. of customer who stayed and exited in the training set:
Stayed     5584
Existed    1416
dtype: int64
```

In [147]:
```
print('\n')
print('No. of customer who stayed and exited  in the validation set:')
print(pd.Series(y_test).value_counts())
```

```

No. of customer who stayed and exited  in the validation set:
Stayed     2379
Existed     621
dtype: int64
```

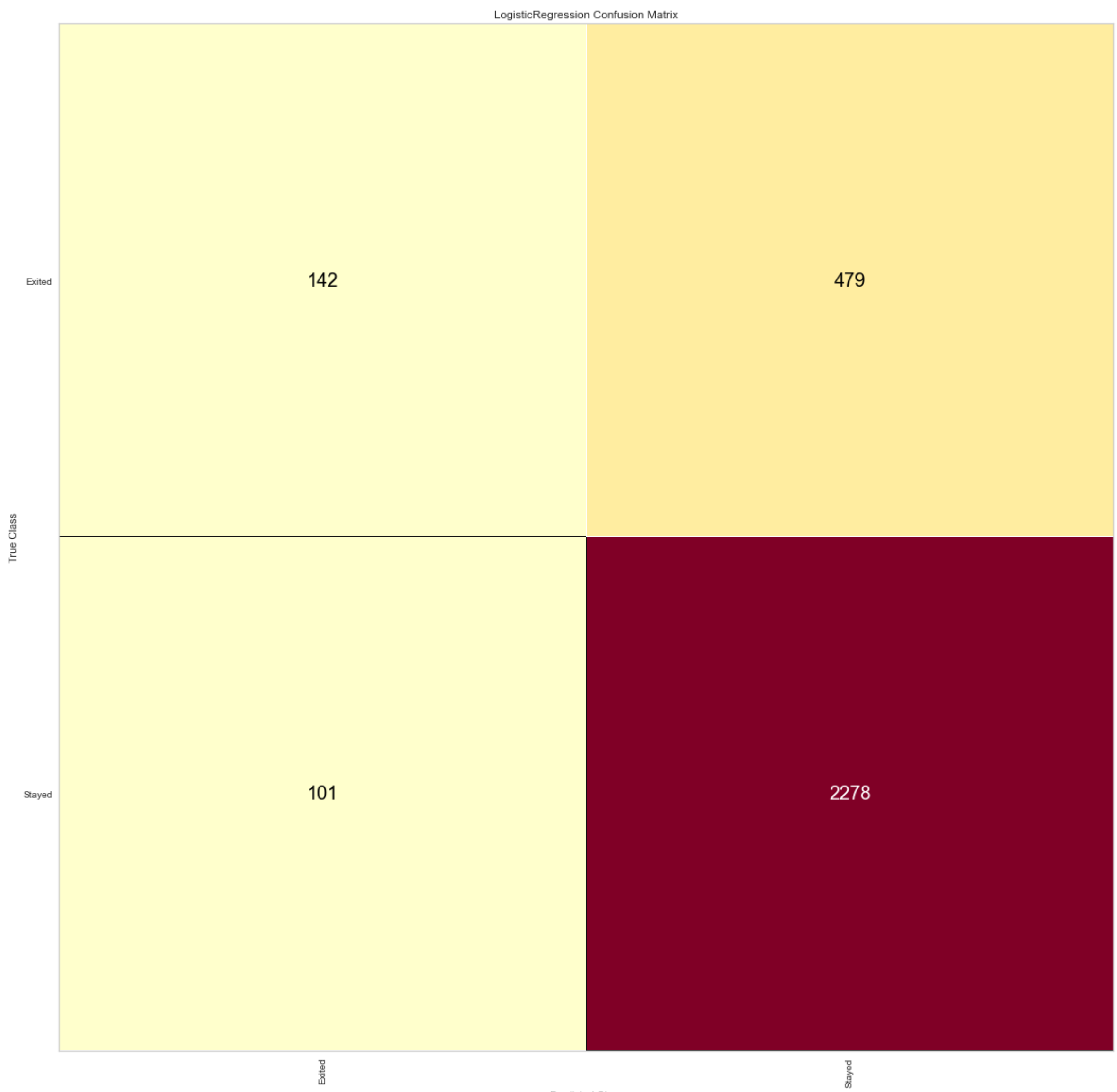
Step 15 - Model evaluation and metrics

Create a logistics regression model

In [148]:
```
from sklearn.linear_model import LogisticRegression

from yellowbrick.classifier import ConfusionMatrix
from yellowbrick.classifier import ClassificationReport
from yellowbrick.classifier import ROCAUC

# Instantiate the classification model
model = LogisticRegression()
```

Define class for 'Exited' and 'stayed' to create confusion metrix and fit it into the trainign sets. Then display the confusion metric

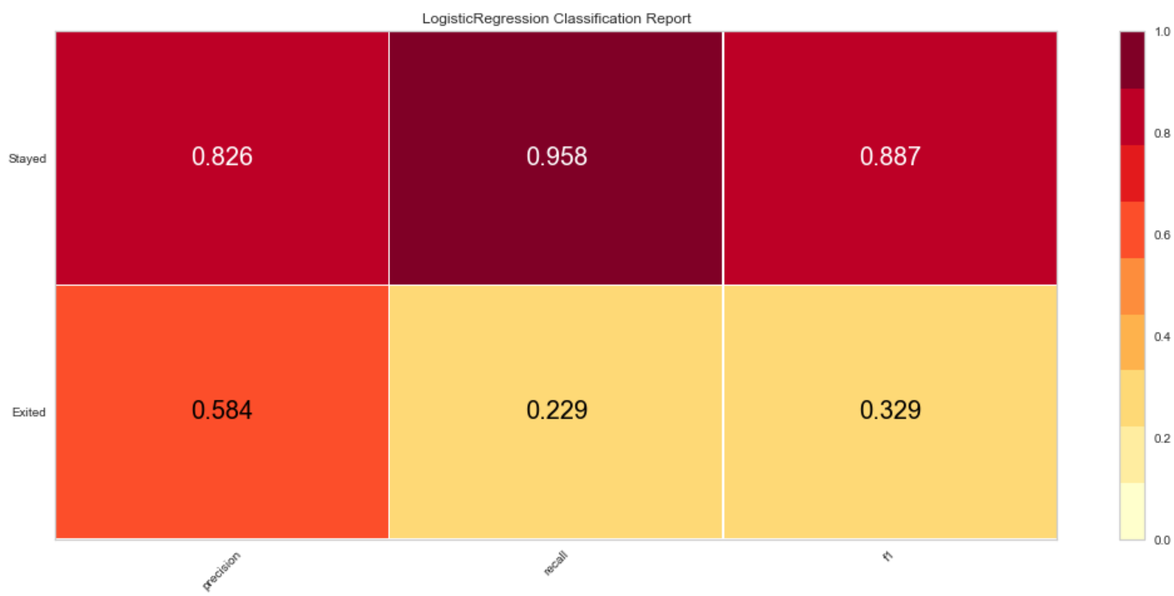
In [150]:
```python
#The ConfusionMatrix visualizer taxes a model
classes = ['Exited','Stayed']
cm = ConfusionMatrix(model, classes=classes, percent=False)

#Fit fits the passed model. This is unnecessary if you pass the visualize
r a pre-fitted model
cm.fit(X_train, y_train)

#To create the ConfusionMatrix, we need some test data. Score runs predic
t() on the data
#and then creates the confusion_matrix from scikit learn.
cm.score(X_test, y_test)

# change fontsize of the labels in the figure
for label in cm.ax.texts:
    label.set_size(20)

#How did we do?
cm.poof()
```

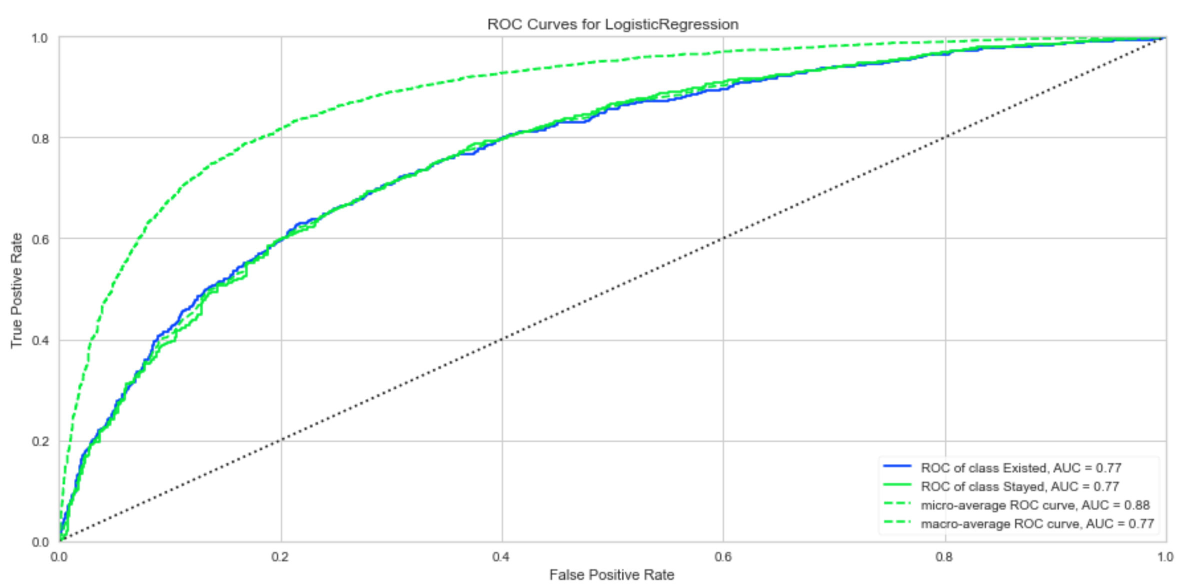


Out[150]: <matplotlib.axes._subplots.AxesSubplot at 0x287cb17bfc8>

Precision, Recall, and F1 Score metrics:

In [154]:
```
#%matplotlib inline
plt.rcParams['figure.figsize'] = (15, 7)
plt.rcParams['font.size'] = 20

# Instantiate the visualizer
visualizer = ClassificationReport(model, classes=classes)

visualizer.fit(X_train, y_train)  # Fit the training data to the visualiz
er
visualizer.score(X_test, y_test)  # Evaluate the model on the test data
g = visualizer.poof()

# ROC and AUC
#Instantiate the visualizer
visualizer = ROCAUC(model)

visualizer.fit(X_train, y_train)  # Fit the training data to the visualiz
er
visualizer.score(X_test, y_test)  # Evaluate the model on the test data
g = visualizer.poof()
```





Artificial Neural Network

In [155]:
```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

#Step 1:  Load data into a dataframe
DataFile = "Data/BankCustomers.xlsx"

dataset = pd.read_excel(DataFile)
print(dataset.shape)
X = dataset.iloc[:, 3:13].values
y = dataset.iloc[:, 13].values
```

```
(10000, 14)
```

In [156]:
```
X[:10]
```

Out[156]:
```
array([[619, 'France', 'Female', 42, 2, 0.0, 1, 1, 1, 101348.88],
       [608, 'Spain', 'Female', 41, 1, 83807.86, 1, 0, 1, 112542.58],
       [502, 'France', 'Female', 42, 8, 159660.8, 3, 1, 0, 113931.57],
       [699, 'France', 'Female', 39, 1, 0.0, 2, 0, 0, 93826.63],
       [850, 'Spain', 'Female', 43, 2, 125510.82, 1, 1, 1, 79084.1],
       [645, 'Spain', 'Male', 44, 8, 113755.78, 2, 1, 0, 149756.71],
       [822, 'France', 'Male', 50, 7, 0.0, 2, 1, 1, 10062.8],
       [376, 'Germany', 'Female', 29, 4, 115046.74, 4, 1, 0, 119346.88],
       [501, 'France', 'Male', 44, 4, 142051.07, 2, 0, 1, 74940.5],
       [684, 'France', 'Male', 27, 2, 134603.88, 1, 1, 1, 71725.73]],
      dtype=object)
```

In [157]:
```
y[:10]
```

Out[157]:
```
array([1, 0, 1, 0, 0, 1, 0, 1, 0, 0], dtype=int64)
```

In [158]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
X[:, 2] = le.fit_transform(X[:, 2])

print(X)
```

```
[[619 'France' 0 ... 1 1 101348.88]
 [608 'Spain' 0 ... 0 1 112542.58]
 [502 'France' 0 ... 1 0 113931.57]
 ...
 [709 'France' 0 ... 0 1 42085.58]
 [772 'Germany' 1 ... 1 0 92888.52]
 [792 'France' 0 ... 1 0 38190.78]]
```

In [159]:
```python
# One Hot Encoding the "Geography" column
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [1])],
remainder='passthrough')
X = np.array(ct.fit_transform(X))
print(X)
```

```
[[1.0 0.0 0.0 ... 1 1 101348.88]
 [0.0 0.0 1.0 ... 0 1 112542.58]
 [1.0 0.0 0.0 ... 1 0 113931.57]
 ...
 [1.0 0.0 0.0 ... 0 1 42085.58]
 [0.0 1.0 0.0 ... 1 0 92888.52]
 [1.0 0.0 0.0 ... 1 0 38190.78]]
```

In [160]:
```python
# Feature scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X = sc.fit_transform(X)
print(X)
```

```
[[ 0.99720391 -0.57873591 -0.57380915 ...  0.64609167  0.97024255
   0.02188649]
 [-1.00280393 -0.57873591  1.74273971 ... -1.54776799  0.97024255
   0.21653375]
 [ 0.99720391 -0.57873591 -0.57380915 ...  0.64609167 -1.03067011
   0.2406869 ]
 ...
 [ 0.99720391 -0.57873591 -0.57380915 ... -1.54776799  0.97024255
  -1.00864308]
 [-1.00280393  1.72790383 -0.57380915 ...  0.64609167 -1.03067011
  -0.12523071]
 [ 0.99720391 -0.57873591 -0.57380915 ...  0.64609167 -1.03067011
  -1.07636976]]
```

In [161]:
```python
# Create trainig and test sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2
, random_state = 0)
```

In [162]:
```python
# initislize an ANN
import tensorflow as tf
ann = tf.keras.models.Sequential()
```

In [163]:
```python
# Add the input layer and the first hidden layer
ann.add(tf.keras.layers.Dense(units=6, activation='relu'))
```

In [164]:
```python
# Add output layer
ann.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))
```

In [166]:

```
# Compile
ann.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = [
'accuracy'])
```

In [167]:
```
# Train on training set
ann.fit(X_train, y_train, batch_size = 32, epochs = 100)
```

```
Train on 8000 samples
Epoch 1/100
8000/8000 [==============================] - 0s 28us/sample - loss: 0.849
9 - acc: 0.3631
Epoch 2/100
8000/8000 [==============================] - 0s 20us/sample - loss: 0.591
5 - acc: 0.7391
Epoch 3/100
8000/8000 [==============================] - 0s 20us/sample - loss: 0.509
9 - acc: 0.7912
Epoch 4/100
8000/8000 [==============================] - 0s 26us/sample - loss: 0.468
9 - acc: 0.7979
Epoch 5/100
8000/8000 [==============================] - 0s 22us/sample - loss: 0.444
0 - acc: 0.8011
Epoch 6/100
8000/8000 [==============================] - ETA: 0s - loss: 0.4307 - ac
c: 0.806 - 0s 21us/sample - loss: 0.4281 - acc: 0.8092
Epoch 7/100
8000/8000 [==============================] - 0s 21us/sample - loss: 0.417
3 - acc: 0.8158
Epoch 8/100
8000/8000 [==============================] - 0s 21us/sample - loss: 0.408
0 - acc: 0.8250
Epoch 9/100
8000/8000 [==============================] - 0s 27us/sample - loss: 0.399
3 - acc: 0.8311
Epoch 10/100
8000/8000 [==============================] - 0s 37us/sample - loss: 0.391
3 - acc: 0.8329
Epoch 11/100
8000/8000 [==============================] - 0s 30us/sample - loss: 0.384
2 - acc: 0.8381
Epoch 12/100
8000/8000 [==============================] - 0s 30us/sample - loss: 0.377
9 - acc: 0.8393
Epoch 13/100
8000/8000 [==============================] - 0s 30us/sample - loss: 0.372
9 - acc: 0.8428
Epoch 14/100
8000/8000 [==============================] - 0s 27us/sample - loss: 0.368
6 - acc: 0.8454
Epoch 15/100
8000/8000 [==============================] - 0s 30us/sample - loss: 0.364
7 - acc: 0.8476
Epoch 16/100
8000/8000 [==============================] - 0s 23us/sample - loss: 0.361
8 - acc: 0.8486
Epoch 17/100
8000/8000 [==============================] - 0s 22us/sample - loss: 0.359
3 - acc: 0.8508
Epoch 18/100
8000/8000 [==============================] - 0s 22us/sample - loss: 0.356
8 - acc: 0.8537
Epoch 19/100
8000/8000 [==============================] - 0s 25us/sample - loss: 0.354
```

```
7 - acc: 0.8547
Epoch 20/100
8000/8000 [==============================] - 0s 30us/sample - loss: 0.352
5 - acc: 0.8553
Epoch 21/100
8000/8000 [==============================] - 0s 24us/sample - loss: 0.350
2 - acc: 0.8565
Epoch 22/100
8000/8000 [==============================] - 0s 23us/sample - loss: 0.348
4 - acc: 0.8571
Epoch 23/100
8000/8000 [==============================] - 0s 23us/sample - loss: 0.346
8 - acc: 0.8591
Epoch 24/100
8000/8000 [==============================] - 0s 23us/sample - loss: 0.345
6 - acc: 0.8609
Epoch 25/100
8000/8000 [==============================] - 0s 23us/sample - loss: 0.344
4 - acc: 0.8608
Epoch 26/100
8000/8000 [==============================] - 0s 28us/sample - loss: 0.343
5 - acc: 0.8580
Epoch 27/100
8000/8000 [==============================] - 0s 24us/sample - loss: 0.342
8 - acc: 0.8602
Epoch 28/100
8000/8000 [==============================] - 0s 24us/sample - loss: 0.342
1 - acc: 0.8622
Epoch 29/100
8000/8000 [==============================] - 0s 27us/sample - loss: 0.341
6 - acc: 0.8611
Epoch 30/100
8000/8000 [==============================] - 0s 23us/sample - loss: 0.341
2 - acc: 0.8596
Epoch 31/100
8000/8000 [==============================] - 0s 26us/sample - loss: 0.340
7 - acc: 0.8610
Epoch 32/100
8000/8000 [==============================] - 0s 27us/sample - loss: 0.340
7 - acc: 0.8610
Epoch 33/100
8000/8000 [==============================] - 0s 24us/sample - loss: 0.340
2 - acc: 0.8600
Epoch 34/100
8000/8000 [==============================] - 0s 25us/sample - loss: 0.339
9 - acc: 0.86240s - loss: 0.3405 - acc: 0.862
Epoch 35/100
8000/8000 [==============================] - 0s 26us/sample - loss: 0.339
8 - acc: 0.8605
Epoch 36/100
8000/8000 [==============================] - 0s 27us/sample - loss: 0.339
6 - acc: 0.8608
Epoch 37/100
8000/8000 [==============================] - 0s 30us/sample - loss: 0.339
6 - acc: 0.8612
Epoch 38/100
8000/8000 [==============================] - 0s 24us/sample - loss: 0.339
```

```
1 - acc: 0.8612
Epoch 39/100
8000/8000 [==============================] - 0s 23us/sample - loss: 0.339
0 - acc: 0.8620
Epoch 40/100
8000/8000 [==============================] - 0s 23us/sample - loss: 0.339
0 - acc: 0.8612
Epoch 41/100
8000/8000 [==============================] - 0s 23us/sample - loss: 0.338
8 - acc: 0.8620
Epoch 42/100
8000/8000 [==============================] - 0s 29us/sample - loss: 0.338
5 - acc: 0.8612
Epoch 43/100
8000/8000 [==============================] - 0s 27us/sample - loss: 0.338
3 - acc: 0.8605
Epoch 44/100
8000/8000 [==============================] - 0s 25us/sample - loss: 0.338
2 - acc: 0.8600
Epoch 45/100
8000/8000 [==============================] - 0s 25us/sample - loss: 0.338
0 - acc: 0.8611
Epoch 46/100
8000/8000 [==============================] - 0s 24us/sample - loss: 0.338
1 - acc: 0.8620
Epoch 47/100
8000/8000 [==============================] - 0s 25us/sample - loss: 0.338
0 - acc: 0.8609
Epoch 48/100
8000/8000 [==============================] - 0s 31us/sample - loss: 0.338
0 - acc: 0.8593
Epoch 49/100
8000/8000 [==============================] - 0s 24us/sample - loss: 0.337
8 - acc: 0.8608
Epoch 50/100
8000/8000 [==============================] - 0s 24us/sample - loss: 0.337
5 - acc: 0.8619
Epoch 51/100
8000/8000 [==============================] - 0s 25us/sample - loss: 0.337
5 - acc: 0.8612
Epoch 52/100
8000/8000 [==============================] - 0s 24us/sample - loss: 0.337
4 - acc: 0.8618
Epoch 53/100
8000/8000 [==============================] - 0s 30us/sample - loss: 0.337
3 - acc: 0.8606
Epoch 54/100
8000/8000 [==============================] - 0s 24us/sample - loss: 0.337
1 - acc: 0.8604
Epoch 55/100
8000/8000 [==============================] - 0s 24us/sample - loss: 0.337
0 - acc: 0.8611
Epoch 56/100
8000/8000 [==============================] - 0s 24us/sample - loss: 0.337
3 - acc: 0.8609
Epoch 57/100
8000/8000 [==============================] - 0s 24us/sample - loss: 0.337
```

```
1 - acc: 0.8609
Epoch 58/100
8000/8000 [==============================] - 0s 27us/sample - loss: 0.337
1 - acc: 0.8605
Epoch 59/100
8000/8000 [==============================] - 0s 29us/sample - loss: 0.337
1 - acc: 0.8622
Epoch 60/100
8000/8000 [==============================] - 0s 24us/sample - loss: 0.337
0 - acc: 0.8597
Epoch 61/100
8000/8000 [==============================] - 0s 24us/sample - loss: 0.337
3 - acc: 0.8608
Epoch 62/100
8000/8000 [==============================] - 0s 24us/sample - loss: 0.337
0 - acc: 0.8610
Epoch 63/100
8000/8000 [==============================] - 0s 24us/sample - loss: 0.336
9 - acc: 0.8614
Epoch 64/100
8000/8000 [==============================] - 0s 29us/sample - loss: 0.336
9 - acc: 0.8619
Epoch 65/100
8000/8000 [==============================] - 0s 24us/sample - loss: 0.336
8 - acc: 0.8609
Epoch 66/100
8000/8000 [==============================] - 0s 28us/sample - loss: 0.336
8 - acc: 0.8601
Epoch 67/100
8000/8000 [==============================] - 0s 26us/sample - loss: 0.336
9 - acc: 0.8620
Epoch 68/100
8000/8000 [==============================] - 0s 24us/sample - loss: 0.336
8 - acc: 0.8612
Epoch 69/100
8000/8000 [==============================] - 0s 30us/sample - loss: 0.337
0 - acc: 0.8614
Epoch 70/100
8000/8000 [==============================] - 0s 25us/sample - loss: 0.336
7 - acc: 0.8609
Epoch 71/100
8000/8000 [==============================] - 0s 26us/sample - loss: 0.336
8 - acc: 0.8612
Epoch 72/100
8000/8000 [==============================] - 0s 32us/sample - loss: 0.336
9 - acc: 0.8619
Epoch 73/100
8000/8000 [==============================] - 0s 25us/sample - loss: 0.336
4 - acc: 0.8612
Epoch 74/100
8000/8000 [==============================] - 0s 29us/sample - loss: 0.336
6 - acc: 0.8612
Epoch 75/100
8000/8000 [==============================] - 0s 26us/sample - loss: 0.336
6 - acc: 0.8620
Epoch 76/100
8000/8000 [==============================] - 0s 25us/sample - loss: 0.336
```

```
4 - acc: 0.8609
Epoch 77/100
8000/8000 [==============================] - 0s 26us/sample - loss: 0.336
4 - acc: 0.8630
Epoch 78/100
8000/8000 [==============================] - 0s 28us/sample - loss: 0.336
5 - acc: 0.8620
Epoch 79/100
8000/8000 [==============================] - 0s 30us/sample - loss: 0.336
4 - acc: 0.8619
Epoch 80/100
8000/8000 [==============================] - 0s 27us/sample - loss: 0.336
6 - acc: 0.8622
Epoch 81/100
8000/8000 [==============================] - 0s 24us/sample - loss: 0.336
4 - acc: 0.8615
Epoch 82/100
8000/8000 [==============================] - 0s 27us/sample - loss: 0.336
2 - acc: 0.8618
Epoch 83/100
8000/8000 [==============================] - 0s 24us/sample - loss: 0.336
4 - acc: 0.8625
Epoch 84/100
8000/8000 [==============================] - 0s 24us/sample - loss: 0.336
2 - acc: 0.86190s - loss: 0.3359 - acc: 0.863
Epoch 85/100
8000/8000 [==============================] - 0s 31us/sample - loss: 0.336
2 - acc: 0.8619
Epoch 86/100
8000/8000 [==============================] - 0s 26us/sample - loss: 0.336
2 - acc: 0.8614
Epoch 87/100
8000/8000 [==============================] - 0s 26us/sample - loss: 0.336
1 - acc: 0.8609
Epoch 88/100
8000/8000 [==============================] - 0s 24us/sample - loss: 0.335
9 - acc: 0.8619
Epoch 89/100
8000/8000 [==============================] - 0s 24us/sample - loss: 0.336
0 - acc: 0.8621
Epoch 90/100
8000/8000 [==============================] - 0s 30us/sample - loss: 0.336
2 - acc: 0.8609
Epoch 91/100
8000/8000 [==============================] - 0s 25us/sample - loss: 0.336
0 - acc: 0.8619
Epoch 92/100
8000/8000 [==============================] - 0s 31us/sample - loss: 0.335
8 - acc: 0.8644
Epoch 93/100
8000/8000 [==============================] - 0s 24us/sample - loss: 0.335
9 - acc: 0.8630
Epoch 94/100
8000/8000 [==============================] - 0s 26us/sample - loss: 0.335
9 - acc: 0.8619
Epoch 95/100
8000/8000 [==============================] - 0s 29us/sample - loss: 0.335
```

```
9 - acc: 0.8618
Epoch 96/100
8000/8000 [==============================] - 0s 26us/sample - loss: 0.335
8 - acc: 0.8645
Epoch 97/100
8000/8000 [==============================] - 0s 24us/sample - loss: 0.335
9 - acc: 0.8620
Epoch 98/100
8000/8000 [==============================] - 0s 25us/sample - loss: 0.335
9 - acc: 0.8631
Epoch 99/100
8000/8000 [==============================] - 0s 25us/sample - loss: 0.335
6 - acc: 0.8622
Epoch 100/100
8000/8000 [==============================] - 0s 28us/sample - loss: 0.335
7 - acc: 0.8621
```

Out[167]:

```
<tensorflow.python.keras.callbacks.History at 0x287cc0a1848>
```

In [168]:

```python
print(X_train)
```

```
[[-1.00280393 -0.57873591  1.74273971 ...  0.64609167 -1.03067011
   1.10838187]
 [-1.00280393  1.72790383 -0.57380915 ...  0.64609167  0.97024255
  -0.74759209]
 [ 0.99720391 -0.57873591 -0.57380915 ...  0.64609167 -1.03067011
   1.48746417]
 ...
 [ 0.99720391 -0.57873591 -0.57380915 ...  0.64609167 -1.03067011
   1.41441489]
 [-1.00280393 -0.57873591  1.74273971 ...  0.64609167  0.97024255
   0.84614739]
 [-1.00280393  1.72790383 -0.57380915 ...  0.64609167 -1.03067011
   0.32630495]]
```

In [169]:

```python
# Making prediction using test set
y_pred = ann.predict(X_test)
y_pred = (y_pred > 0.5)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y
_test),1)),1))
```

```
[[0 0]
 [0 1]
 [0 0]
 ...
 [0 0]
 [0 0]
 [0 0]]
```

In [173]:

```python
# making confusion matrix to detrmine accuracy
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[1518   77]
 [ 190  215]]
```

From CF:

1- TP = 1518

1. TN = 215
2. FN = 77
3. FP = 190

Accuracy = (TP + TN)/(TP + TN + FP + FN) = (1518 + 215)/(1518 + 215 + 190 + 77) = 1733/2000 = .8665 = 86.65%

Given the correlation between independent variables, 86.65 accuracy is adequate.

Recall = TP/(TP + FN) = 1518/(1518+77) = .9117 = 91.17%

Precision = TP/(TP + FN) = 1518/(1518 + 190) = .88875 = 88%

```
In [ ]:
```

XGBoost

Importing the libraries

In [21]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import xlrd
```

Importing the dataset

In [40]:
```
#Step 1:  Load data into a dataframe
DataFile = "Data/BankCustomers.xlsx"

dataset = pd.read_excel(DataFile)
print(dataset.shape)
```

```
(10000, 14)
```

In [41]:
```python
print(dataset.iloc[:, 3:-1])
print(dataset.iloc[:, -1])
```

```
      CreditScore Geography  Gender  Age  Tenure    Balance  NumOfProduct
s  \
0             619    France  Female   42       2       0.00
1
1             608     Spain  Female   41       1   83807.86
1
2             502    France  Female   42       8  159660.80
3
3             699    France  Female   39       1       0.00
2
4             850     Spain  Female   43       2  125510.82
1
...           ...       ...     ...  ...     ...        ...
...
9995          771    France    Male   39       5       0.00
2
9996          516    France    Male   35      10   57369.61
1
9997          709    France  Female   36       7       0.00
1
9998          772   Germany    Male   42       3   75075.31
2
9999          792    France  Female   28       4  130142.79
1

      HasCrCard  IsActiveMember  EstimatedSalary
0             1               1        101348.88
1             0               1        112542.58
2             1               0        113931.57
3             0               0         93826.63
4             1               1         79084.10
...         ...             ...              ...
9995          1               0         96270.64
9996          1               1        101699.77
9997          0               1         42085.58
9998          1               0         92888.52
9999          1               0         38190.78

[10000 rows x 10 columns]
0       1
1       0
2       1
3       0
4       0
       ..
9995    0
9996    0
9997    1
9998    1
9999    0
Name: Exited, Length: 10000, dtype: int64
```

In [44]:
```
X = dataset.iloc[:, 3:-1].values
y = dataset.iloc[:, -1].values
```

In [45]:
```
print(X)
```

```
[[619 'France' 'Female' ... 1 1 101348.88]
 [608 'Spain' 'Female' ... 0 1 112542.58]
 [502 'France' 'Female' ... 1 0 113931.57]
 ...
 [709 'France' 'Female' ... 0 1 42085.58]
 [772 'Germany' 'Male' ... 1 0 92888.52]
 [792 'France' 'Female' ... 1 0 38190.78]]
```

In [46]:
```
X = dataset.iloc[:, 3:-1].values
y = dataset.iloc[:, -1].values
```

In [47]:
```
print(X)
```

```
[[619 'France' 'Female' ... 1 1 101348.88]
 [608 'Spain' 'Female' ... 0 1 112542.58]
 [502 'France' 'Female' ... 1 0 113931.57]
 ...
 [709 'France' 'Female' ... 0 1 42085.58]
 [772 'Germany' 'Male' ... 1 0 92888.52]
 [792 'France' 'Female' ... 1 0 38190.78]]
```

In [48]:
```
print(y)
```

```
[1 0 1 ... 1 1 0]
```

Encoding categorical data

Label Encoding the "Gender" column

In [49]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
X[:, 2] = le.fit_transform(X[:, 2])
```

In [50]:
```
print(X)
```

```
[[619 'France' 0 ... 1 1 101348.88]
 [608 'Spain' 0 ... 0 1 112542.58]
 [502 'France' 0 ... 1 0 113931.57]
 ...
 [709 'France' 0 ... 0 1 42085.58]
 [772 'Germany' 1 ... 1 0 92888.52]
 [792 'France' 0 ... 1 0 38190.78]]
```

One Hot Encoding the "Geography" column

In [51]:
```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [1])],
remainder='passthrough')
X = np.array(ct.fit_transform(X))
```

In [52]:
```
print(X)
```

```
[[1.0 0.0 0.0 ... 1 1 101348.88]
 [0.0 0.0 1.0 ... 0 1 112542.58]
 [1.0 0.0 0.0 ... 1 0 113931.57]
 ...
 [1.0 0.0 0.0 ... 0 1 42085.58]
 [0.0 1.0 0.0 ... 1 0 92888.52]
 [1.0 0.0 0.0 ... 1 0 38190.78]]
```

Splitting the dataset into the Training set and Test set

In [53]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2
, random_state = 0)
print(X_train,y_train)
```

```
[[0.0 0.0 1.0 ... 1 0 163830.64]
 [0.0 1.0 0.0 ... 1 1 57098.0]
 [1.0 0.0 0.0 ... 1 0 185630.76]
 ...
 [1.0 0.0 0.0 ... 1 0 181429.87]
 [0.0 0.0 1.0 ... 1 1 148750.16]
 [0.0 1.0 0.0 ... 1 0 118855.26]] [0 0 0 ... 0 0 1]
```

Training XGBoost on the Training set

In [54]:
```
from xgboost import XGBClassifier
classifier = XGBClassifier()
classifier.fit(X_train, y_train)
```

Out[54]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.300000012, max_delta_step=0, max_depth=6,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=100, n_jobs=0, num_parallel_tree=1, random_sta
te=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

Predicting the Test set results

In [55]:
```
y_pred = classifier.predict(X_test)
```

Making the Confusion Matrix

In [56]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[1497   98]
 [ 196  209]]
```

From CF:

1- TP = 1497

1. TN = 209
2. FN = 98
3. FP = 196

Accuracy = (TP + TN)/(TP + TN + FP + FN) = (1497 + 209)/(1497 + 209 + 196 + 98) = 1706/2000 = .853 = 85.3%%

Not much dfference with ANN

Recall = TP/(TP + FN) = 1497/(1497+98) = .9385 = 93.85%

Precision = TP/(TP + FN) = 1497/(1497 + 196) = .8842 = 88.52%

Applying k-Fold Cross Validation

In [57]:
```
from sklearn.model_selection import cross_val_score
accuracies = cross_val_score(estimator = classifier, X = X_train, y = y_t
rain, cv = 10)
print("Accuracy: {:.2f} %".format(accuracies.mean()*100))
print("Standard Deviation: {:.2f} %".format(accuracies.std()*100))
```

```
Accuracy: 85.25 %
Standard Deviation: 1.39 %
```