Original Analysis Case Study

Part 1 : Graphics Analysis

Part 2 : Feature Reduction (Extraction/Selection)

Part 3 : Filling in Missing Values

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 [3]:
```
# Load Libraries
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import xlrd
```

In [5]:
```
#Step 1:  Load data into a dataframe
DataFile = "Data/BankCustomers.xlsx"

data = pd.read_excel(DataFile)
```

In [6]:
```
# 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 [7]:
```
#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 [8]:
```
#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 [9]:
```
# Step 6a: Summary of object type data
print("Summarized Data")
print(data.describe(include=['O']))
```

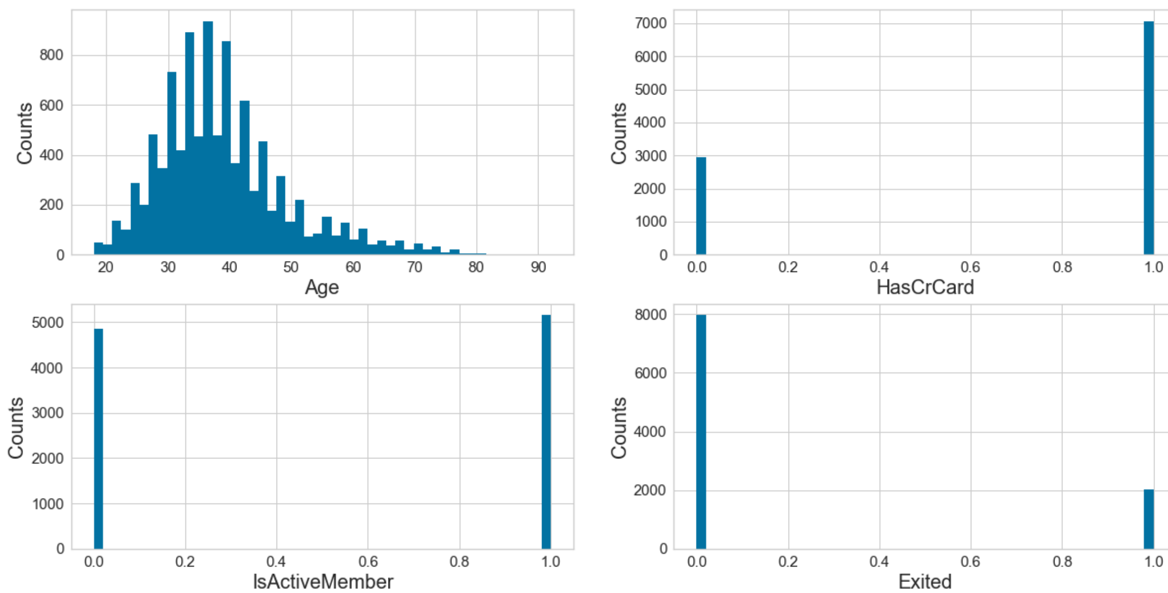
```
Summarized Data
       Surname Geography Gender
count    10000     10000  10000
unique    2932         3      2
top      Smith    France   Male
freq        32      5014   5457
```

In [10]:
```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 [11]:
```
# 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()
```

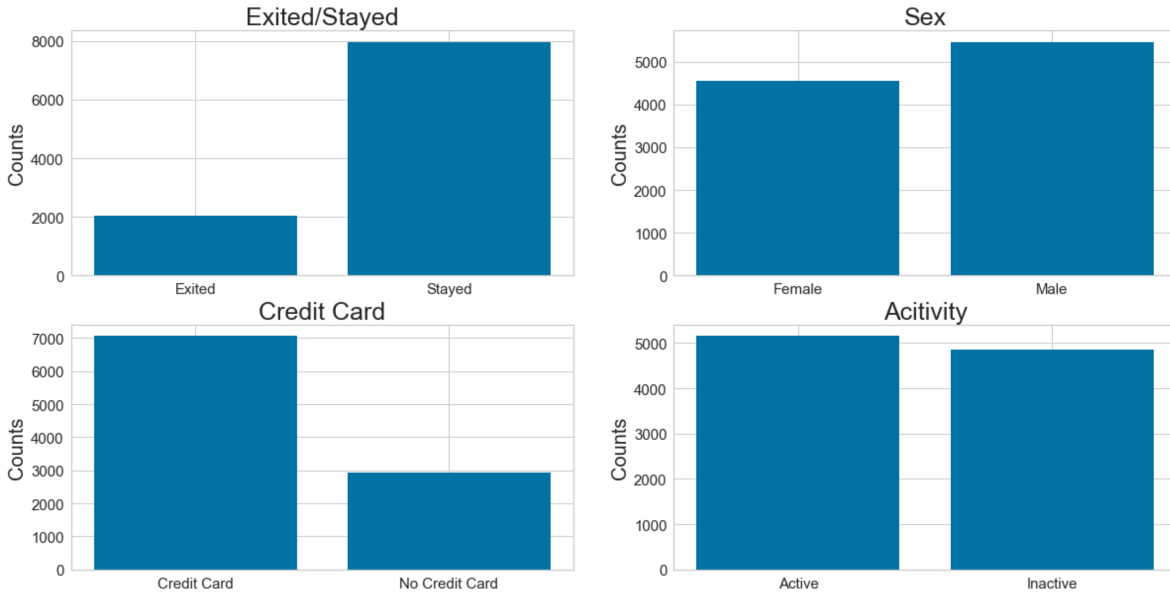


Barchart comparing the number of:

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

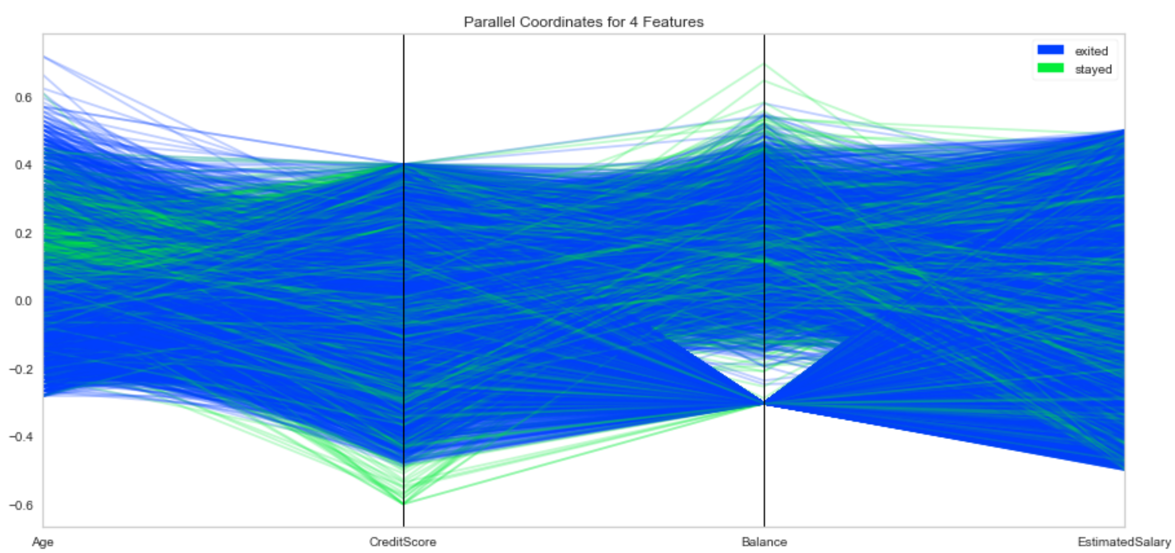
In [12]:

```
# 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 [13]:
```
# 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();
```



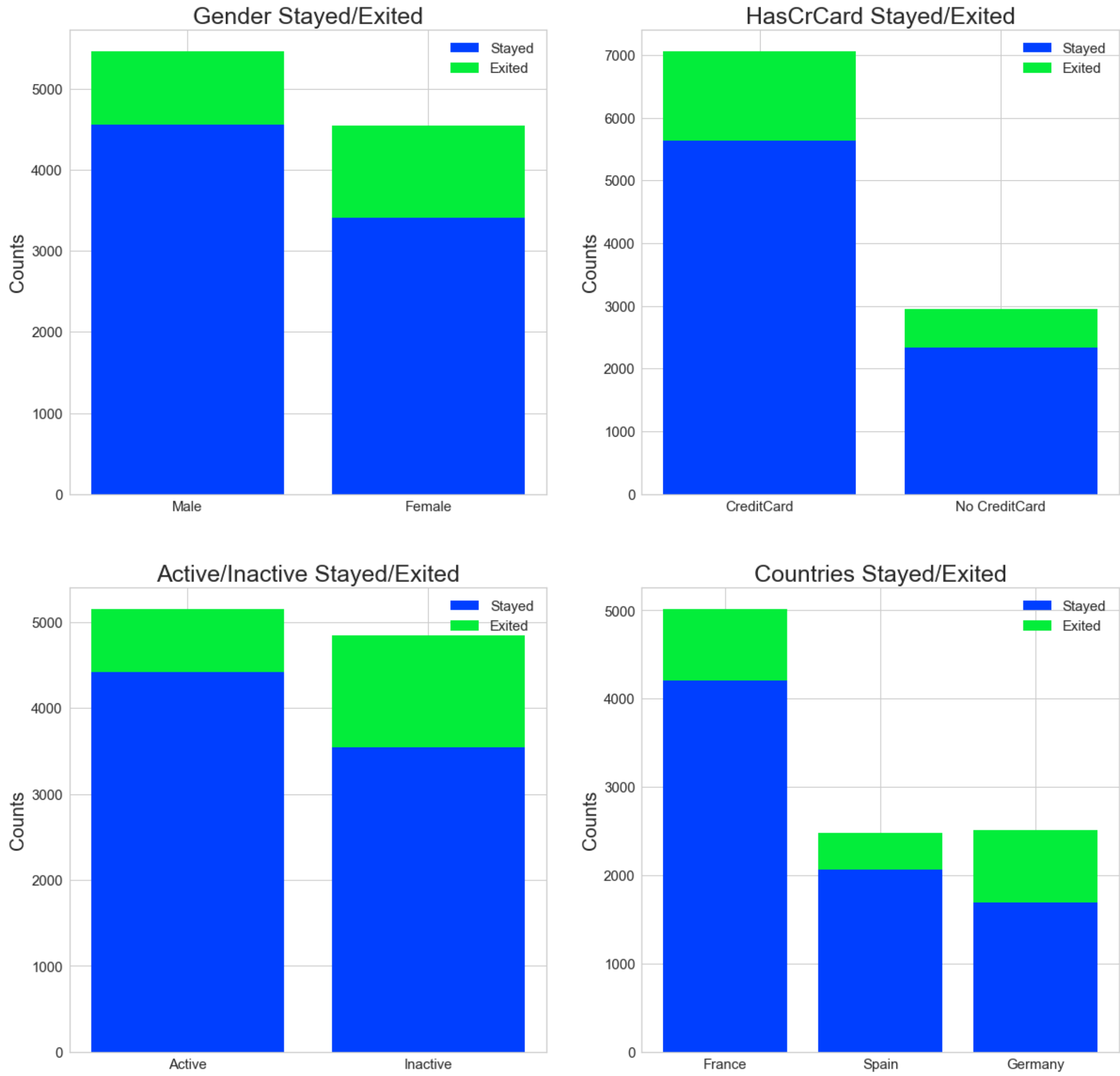
Stacked bar charts showing stays and exits based on:

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

In [14]:

```
# 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 [15]:

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

data = pd.read_excel(DataFile)
```

In [16]:

```
data.columns
```

Out[16]:

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

In [17]:

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

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

In [19]:
```
# 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[19]:
```
array([[1, 0, 0],
       [0, 0, 1],
       [1, 0, 0],
       ...,
       [1, 0, 0],
       [0, 1, 0],
       [1, 0, 0]], dtype=int32)
```

In [20]:
```
one_hot.classes_
```
Out[20]:
```
array(['France', 'Germany', 'Spain'], dtype='<U7')
```

In [21]:
```
dummies = pd.get_dummies(feature)
dummies.head()
```
Out[21]:

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

In [22]:
```
# Drop Geography column
data = data.drop(['Geography'],axis=1)
```

In [23]:
```
# Add dummies
data[dummies.columns] = dummies
data.head()
```

Out[23]:

	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 [24]:
```
# one-hot code Gender
feature = np.array(data['Gender'])
one_hot = LabelBinarizer()

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

Out[24]:

	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 [25]:
```
 #drop Gender and add dummies
data = data.drop(['Gender'],axis=1)
data[dummies.columns] = dummies
data.head()
```

Out[25]:

	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 [26]:
```
# Drop spain and male to avoid dummy trap
data = data.drop(['Male','Spain'],axis=1)
data.head()
```

Out[26]:

	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 [64]:
```
# Move the dependent variable column to the last position.

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

In [97]:
```
data = data.drop(['Exited'],axis=1)
data['Exited'] = Exited
data.head()
```

Out[97]:

	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 [99]:
```
# 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 [100]:
```
data.shape
```

Out[100]:
```
(10000, 12)
```

In [101]:
```
X.shape
```

Out[101]:
```
(10000, 11)
```

In [102]:
```
y.shape
```

Out[102]:
```
(10000,)
```

In [103]:
```
# 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 [104]:

```
# 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 [105]:

```
# 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 : Filling in Missing Values

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.

- Split_Train_Test

- Model Selection and Evaluation

In [107]:
```
import pandas as pd
import yellowbrick

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

In [111]:
```
print("Indpenden variables matrix:\n")
print(X)
```

```
Indpenden 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 [112]:
```
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 [114]:
```
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 [115]:
```
print(y_train.shape)
```

```
(7000,)
```

In [116]:
```
print(y_val.shape)
```

```
(3000,)
```

In [117]:
```
# 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 [118]:
```
print('\n')
print('No. of customer who stayed and exited  in the validation set:')
print(pd.Series(y_val).value_counts())
```

```

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

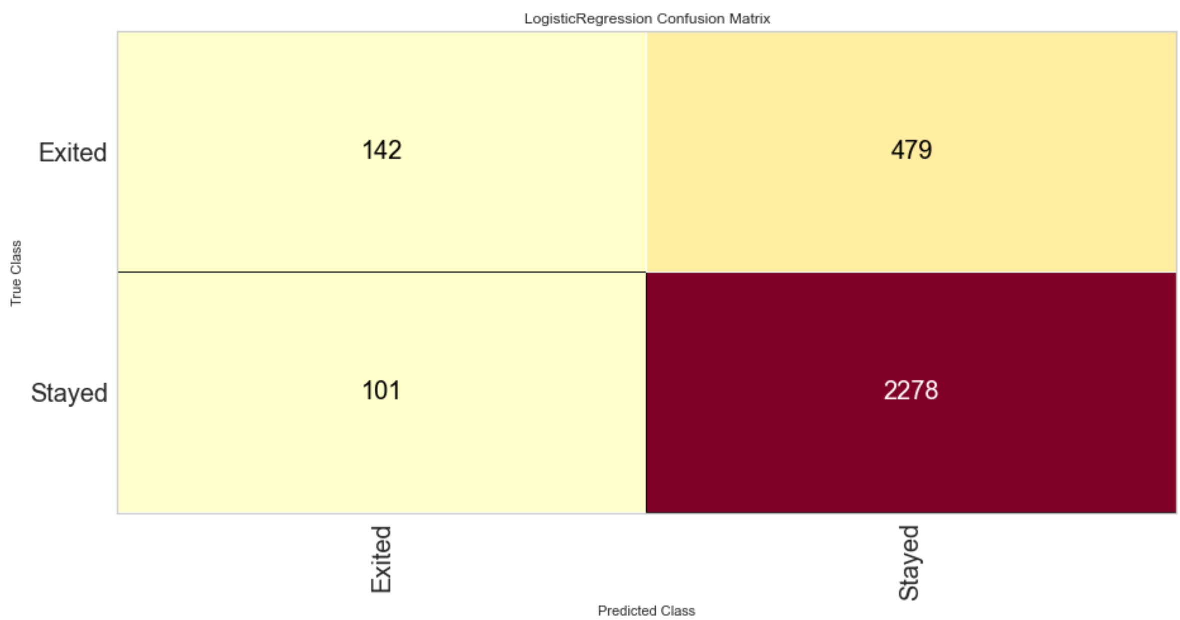
Step 15 - Model evaluation and metrics

Create a logistics regression model

In [119]:
```
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

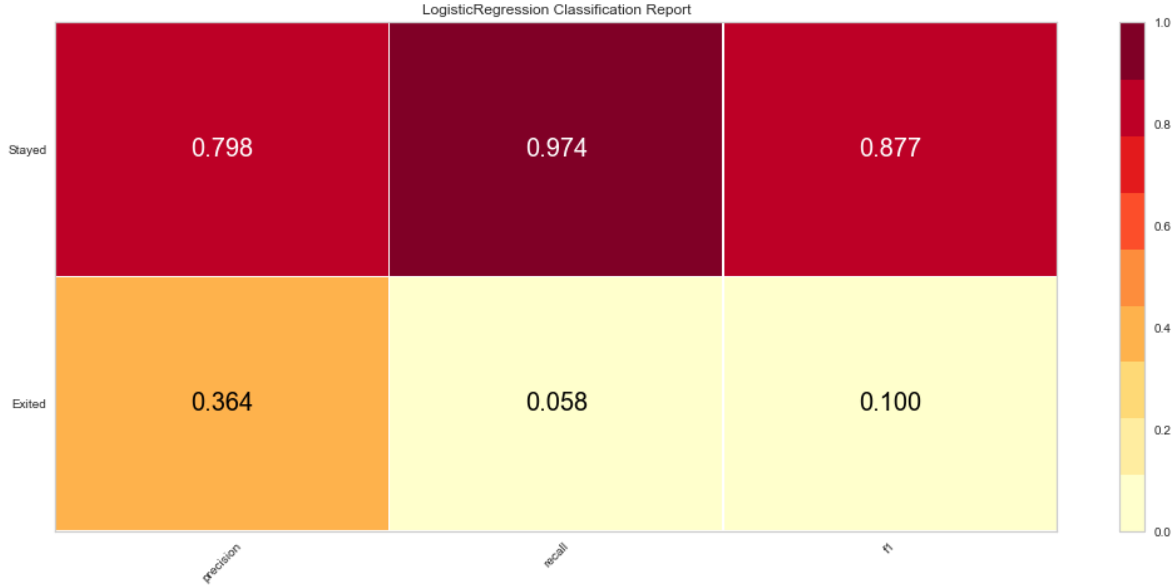
In [120]:
```
#The ConfusionMatrix visualizer taxes a model
classes = ['Exited','Stayed']
cm = ConfusionMatrix(model, classes=classes, percent=False)

#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_val, y_val)

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

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

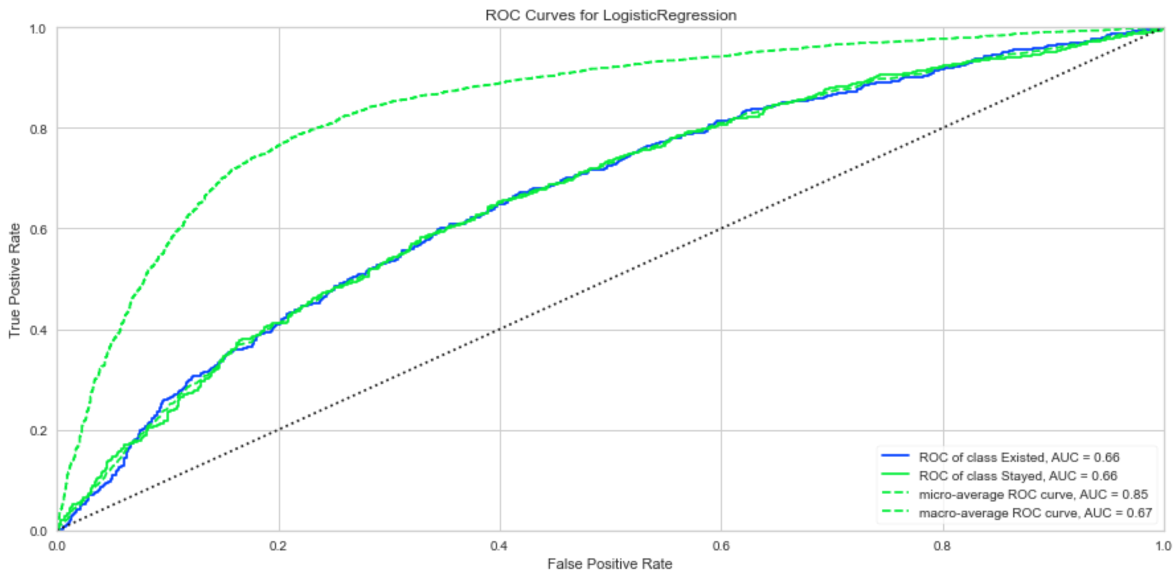


Out[120]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x10be7bb0>
```

Precision, Recall, and F1 Score metrics:

In [95]:
```
#%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_val, y_val)  # 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_val, y_val)  # Evaluate the model on the test data
g = visualizer.poof()
```


LogisticRegression Classification Report
Stayed
Exited
0.798
0.974
0.877
0.364
0.058
0.100
precision
recall
f1
1.0
0.8
0.6
0.4
0.2
0.0


ROC Curves for LogisticRegression
True Postive Rate
False Positive Rate
ROC of class Existed, AUC = 0.66
ROC of class Stayed, AUC = 0.66
micro-average ROC curve, AUC = 0.85
macro-average ROC curve, AUC = 0.67
0.0
0.2
0.4
0.6
0.8
1.0