```
## import required libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib import rcParams


##2.Load the dataset.

df=pd.read_csv('Churn_Modelling.csv')
df.head()
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	B
0	1	15634602	Hargrave	619	France	Female	42	2	
1	2	15647311	Hill	608	Spain	Female	41	1	8
2	3	15619304	Onio	502	France	Female	42	8	15
3	4	15701354	Boni	699	France	Female	39	1	
4	5	15737888	Mitchell	850	Spain	Female	43	2	12

```
df.shape
```

```
(10000, 14)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   RowNumber        10000 non-null  int64  
 1   CustomerId       10000 non-null  int64  
 2   Surname          10000 non-null  object 
 3   CreditScore      10000 non-null  int64  
 4   Geography        10000 non-null  object 
 5   Gender           10000 non-null  object 
 6   Age              10000 non-null  int64  
 7   Tenure           10000 non-null  int64  
 8   Balance          10000 non-null  float64
 9   NumOfProducts    10000 non-null  int64  
 10  HasCrCard        10000 non-null  int64  
 11  IsActiveMember   10000 non-null  int64  
 12  EstimatedSalary  10000 non-null  float64
```

```
 13  Exited           10000 non-null  int64
dtypes: float64(2), int64(9), object(3)
memory usage: 1.1+ MB
```

```
df.isnull().any()
```

```
RowNumber          False
CustomerId         False
Surname            False
CreditScore        False
Geography          False
Gender             False
Age                False
Tenure             False
Balance            False
NumOfProducts      False
HasCrCard          False
IsActiveMember     False
EstimatedSalary    False
Exited             False
dtype: bool
```

```
df.Gender.value_counts()
```

```
Male      5457
Female    4543
Name: Gender, dtype: int64
```

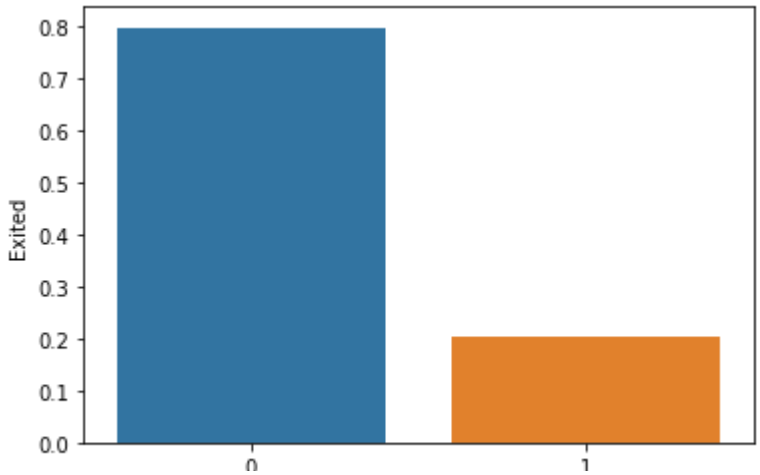
```
df.describe()
```

	RowNumber	CustomerId	CreditScore	Age	Tenure	Balance
count	10000.00000	1.000000e+04	10000.000000	10000.000000	10000.000000	10000.000000
mean	5000.50000	1.569094e+07	650.528800	38.921800	5.012800	76485.889288
std	2886.89568	7.193619e+04	96.653299	10.487806	2.892174	62397.405202
min	1.00000	1.556570e+07	350.000000	18.000000	0.000000	0.000000
25%	2500.75000	1.562853e+07	584.000000	32.000000	3.000000	0.000000
50%	5000.50000	1.569074e+07	652.000000	37.000000	5.000000	97198.540000
75%	7500.25000	1.575323e+07	718.000000	44.000000	7.000000	127644.240000
max	10000.00000	1.581569e+07	850.000000	92.000000	10.000000	250898.090000

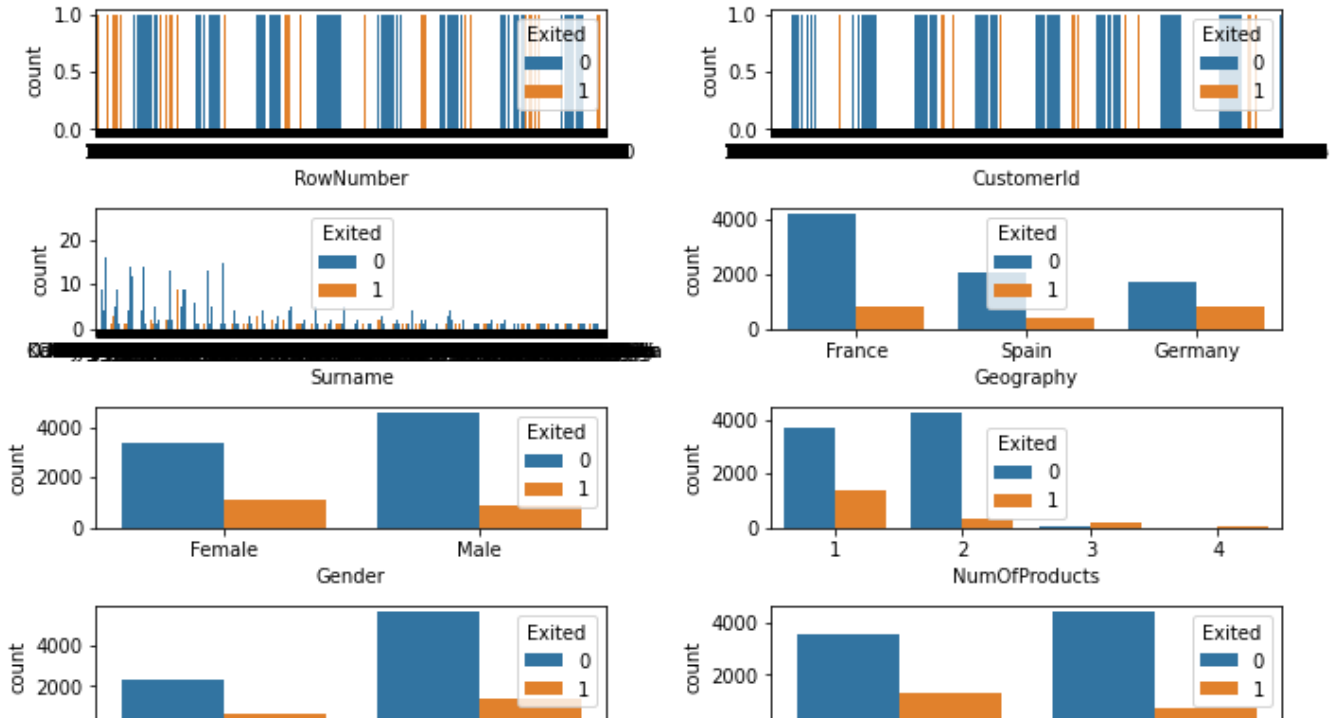
```
#3. Perform Below Visualizations.
#● Univariate Analysis
#● Bi - Variate Analysis
#● Multi - Variate Analysis
```

```
import seaborn as sns
density = df['Exited'].value_counts(normalize=True).reset_index()
sns.barplot(data=density, x='index', y='Exited', );
density
```

	index	Exited
0	0	0.7963
1	1	0.2037




```
#the data is significantly imbalanced
import matplotlib.pyplot as plt
categorical = df.drop(columns=['CreditScore', 'Age', 'Tenure', 'Balance', 'EstimatedS
rows = int(np.ceil(categorical.shape[1] / 2)) - 1



# create sub-plots anf title them
fig, axes = plt.subplots(nrows=rows, ncols=2, figsize=(10,6))
axes = axes.flatten()

for row in range(rows):
    cols = min(2, categorical.shape[1] - row*2)
    for col in range(cols):
        col_name = categorical.columns[2 * row + col]
        ax = axes[row*2 + col]

        sns.countplot(data=categorical, x=col_name, hue="Exited", ax=ax);

plt.tight_layout()
```



#4. Perform descriptive statistics on the dataset.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   RowNumber        10000 non-null  int64  
 1   CustomerId       10000 non-null  int64  
 2   Surname          10000 non-null  object 
 3   CreditScore      10000 non-null  int64  
 4   Geography        10000 non-null  object 
 5   Gender           10000 non-null  object 
 6   Age              10000 non-null  int64  
 7   Tenure           10000 non-null  int64  
 8   Balance          10000 non-null  float64
 9   NumOfProducts    10000 non-null  int64  
 10  HasCrCard        10000 non-null  int64  
 11  IsActiveMember   10000 non-null  int64  
 12  EstimatedSalary  10000 non-null  float64
 13  Exited           10000 non-null  int64  
dtypes: float64(2), int64(9), object(3)
memory usage: 1.1+ MB
```

```
df.describe()
```

	RowNumber	CustomerId	CreditScore	Age	Tenure	Balance
count	10000.00000	1.000000e+04	10000.000000	10000.000000	10000.000000	10000.000000
mean	5000.50000	1.569094e+07	650.528800	38.921800	5.012800	76485.889288
std	2886.89568	7.193619e+04	96.653299	10.487806	2.892174	62397.405202
min	1.00000	1.556570e+07	350.000000	18.000000	0.000000	0.000000
25%	2500.75000	1.562853e+07	584.000000	32.000000	3.000000	0.000000
75%	7500.25000	1.575323e+07	718.000000	44.000000	7.000000	127644.240000

```
#5. Handle the Missing values.
```

```
df.isna().sum()
```

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

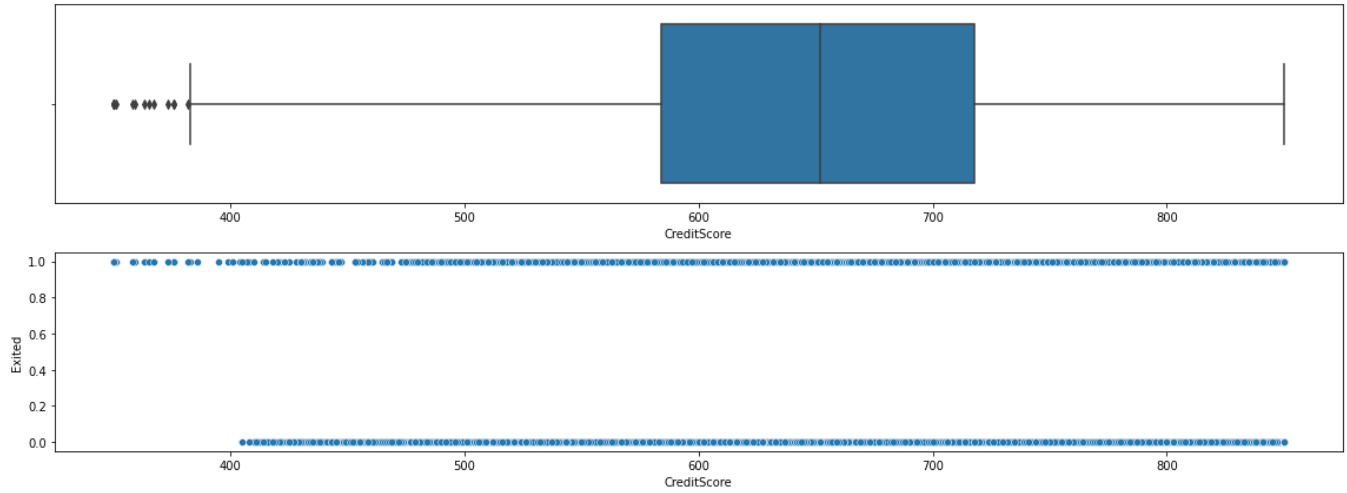
```
#there is no missing values in dataset


for i in df:
    if df[i].dtype=='object' or df[i].dtype=='category':
        print("unique of "+i+" is "+str(len(set(df[i])))+" they are "+str(set(df[i])))
```

```
unique of Surname is 2932 they are {'Bezrukov', 'Estes', 'Fontaine', 'Estrada',
unique of Geography is 3 they are {'France', 'Germany', 'Spain'}
unique of Gender is 2 they are {'Female', 'Male'}
```

```
#6. Find the outliers and replace the outliers


#Checking for outliers


def box_scatter(data, x, y):
    fig, (ax1, ax2) = plt.subplots(nrows=2, ncols=1, figsize=(16,6))
    sns.boxplot(data=data, x=x, ax=ax1)
    sns.scatterplot(data=data, x=x,y=y,ax=ax2)
```

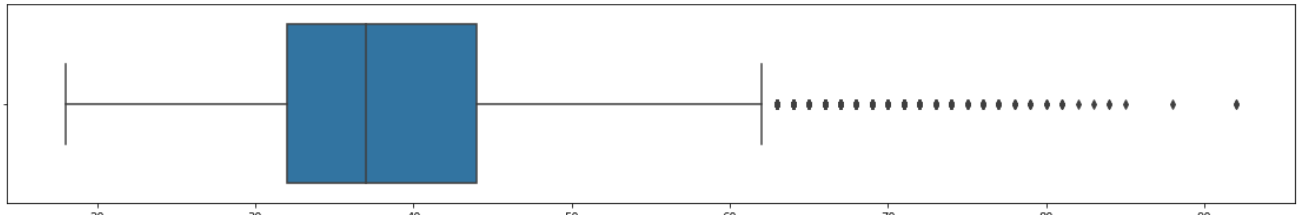
```
box_scatter(df,'CreditScore','Exited');
plt.tight_layout()
print(f"# of Bivariate Outliers: {len(df.loc[df['CreditScore'] < 400])}")
```

```
# of Bivariate Outliers: 19
```



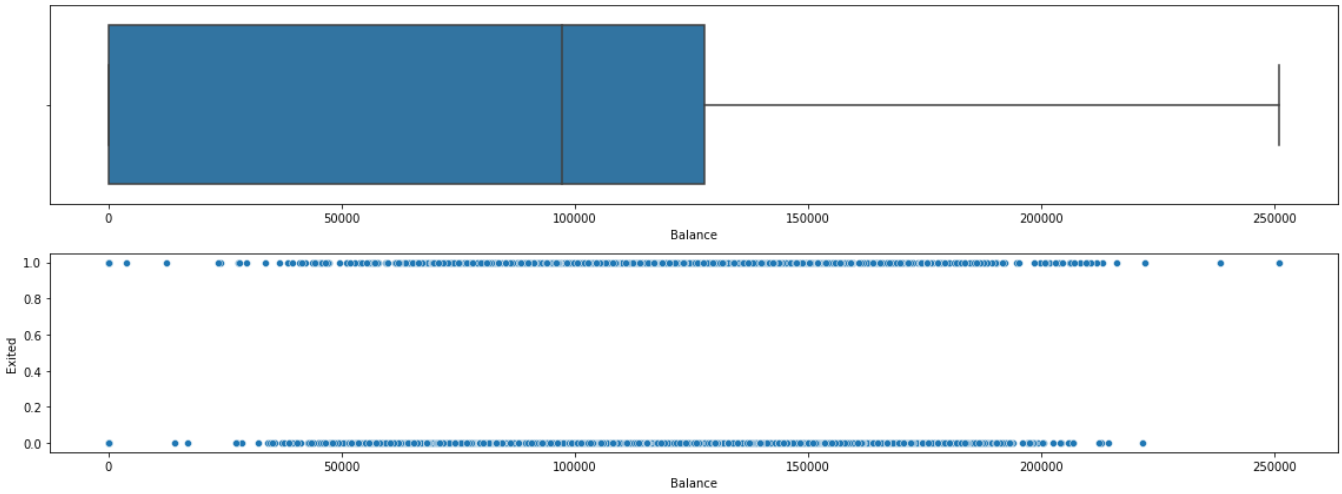

```
box_scatter(df,'Age','Exited');
plt.tight_layout()
print(f"# of Bivariate Outliers: {len(df.loc[df['Age'] > 87])}")
```

```
# of Bivariate Outliers: 3
```

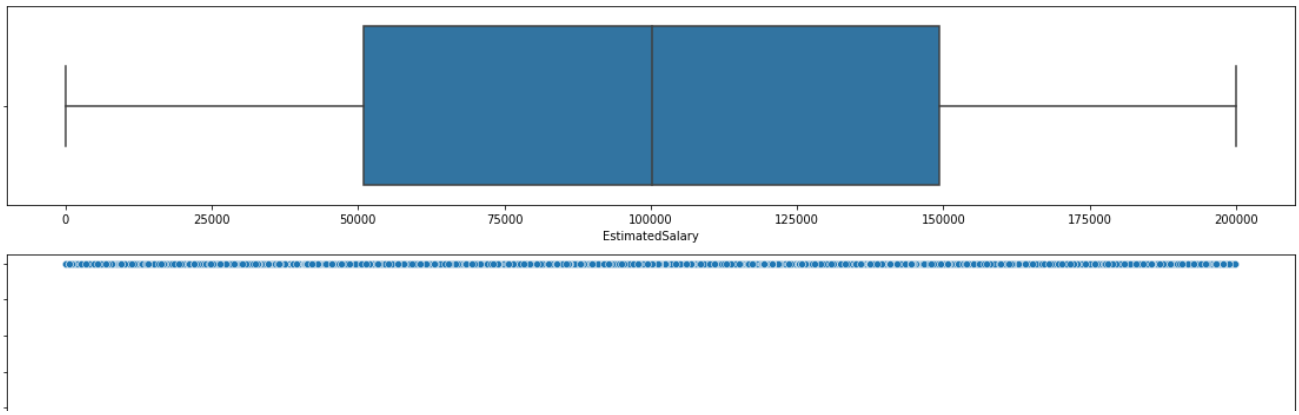


```
box_scatter(df,'Balance','Exited');
plt.tight_layout()
print(f"# of Bivariate Outliers: {len(df.loc[df['Balance'] > 220000])}")
```

```
# of Bivariate Outliers: 4
```

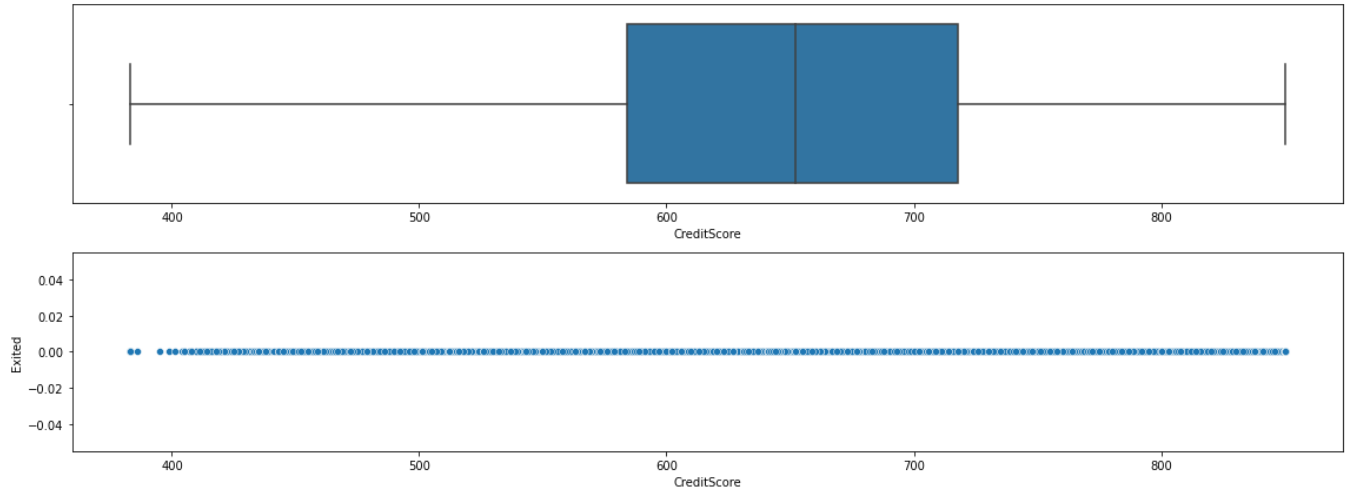



```
box_scatter(df,'EstimatedSalary','Exited');
plt.tight_layout()
```



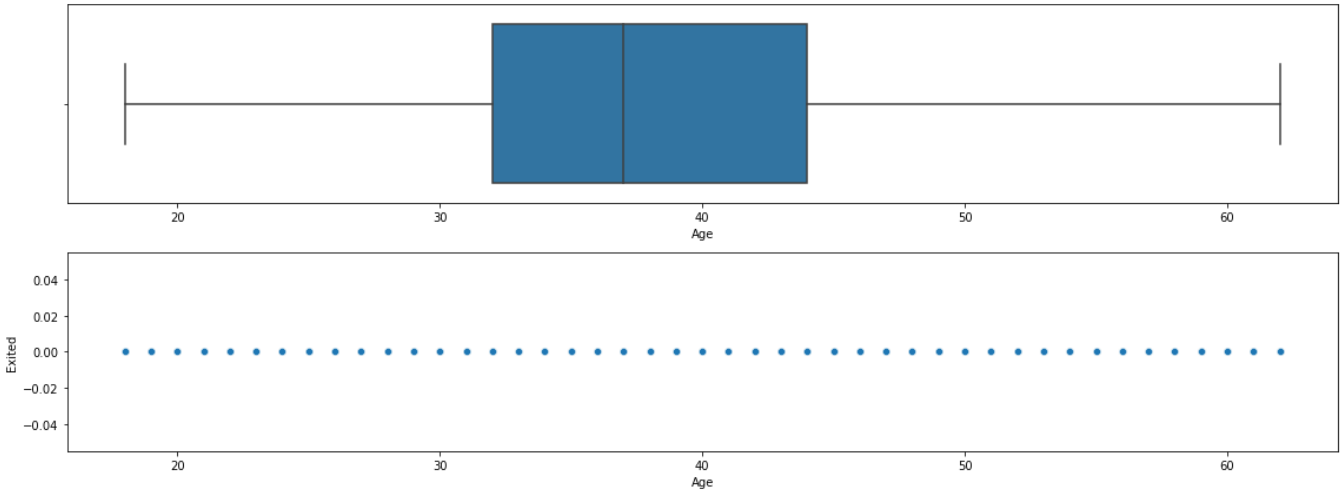

```
#Removing outliers


for i in df:
    if df[i].dtype=='int64' or df[i].dtypes=='float64':
        q1=df[i].quantile(0.25)
        q3=df[i].quantile(0.75)
        iqr=q3-q1
        upper=q3+1.5*iqr
        lower=q1-1.5*iqr
        df[i]=np.where(df[i] >upper, upper, df[i])
        df[i]=np.where(df[i] <lower, lower, df[i])



#After removing outliers, boxplot will be like


box_scatter(df,'CreditScore','Exited');
plt.tight_layout()
print(f"# of Bivariate Outliers: {len(df.loc[df['CreditScore'] < 400])}")
```

```
 # of Bivariate Outliers: 19
```



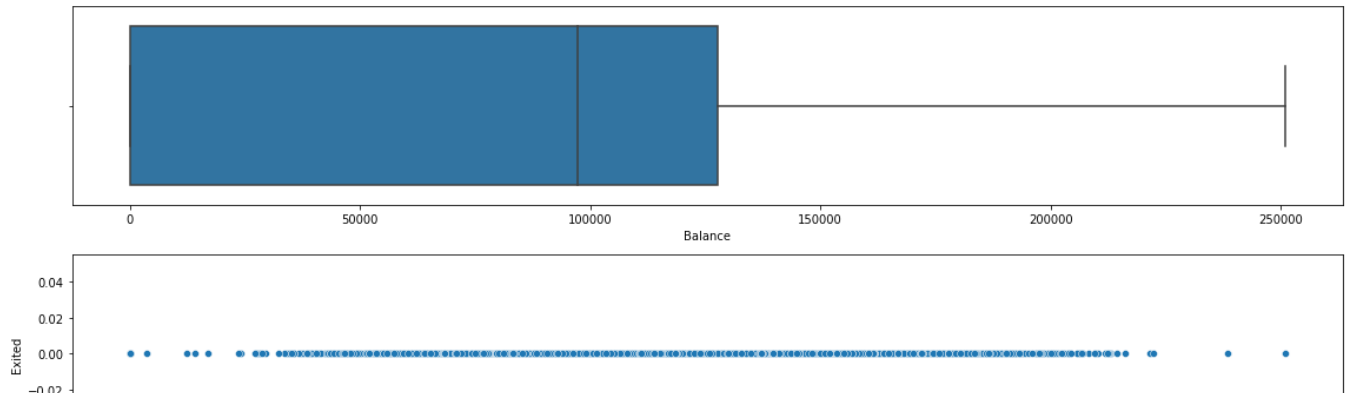
```
box_scatter(df,'Age','Exited');
plt.tight_layout()
print(f"# of Bivariate Outliers: {len(df.loc[df['Age'] > 87])}")
```

```
# of Bivariate Outliers: 0
```




```
box_scatter(df,'Balance','Exited');
plt.tight_layout()
print(f"# of Bivariate Outliers: {len(df.loc[df['Balance'] > 220000])}")
```

```
# of Bivariate Outliers: 4
```



#7. Check for Categorical columns and perform encoding.


```
from sklearn.preprocessing import LabelEncoder
encoder=LabelEncoder()
for i in df:
    if df[i].dtype=='object' or df[i].dtype=='category':
        df[i]=encoder.fit_transform(df[i])
```

#8. Split the data into dependent and independent variables.

```
x=df.iloc[:,:-1]
x.head()
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	
0	1.0	15634602.0	1115	619.0	0	0	42.0	2.0	
1	2.0	15647311.0	1177	608.0	2	0	41.0	1.0	8
2	3.0	15619304.0	2040	502.0	0	0	42.0	8.0	15
3	4.0	15701354.0	289	699.0	0	0	39.0	1.0	
4	5.0	15737888.0	1822	850.0	2	0	43.0	2.0	12

```
y=df.iloc[:,-1]
y.head()
```

```
0    0.0
1    0.0
2    0.0
3    0.0
4    0.0
Name: Exited, dtype: float64
```

#9. Scale the independent variables

```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
x=scaler.fit_transform(x)
```

```
x
```

```
array([[-1.73187761, -0.78321342, -0.46418322, ...,  0.64609167,
         0.97024255,  0.02188649],
       [-1.7315312 , -0.60653412, -0.3909112 , ..., -1.54776799,
         0.97024255,  0.21653375],
       [-1.73118479, -0.99588476,  0.62898807, ...,  0.64609167,
        -1.03067011,  0.2406869 ],
       ...,
       [ 1.73118479, -1.47928179,  0.07353887, ..., -1.54776799,
         0.97024255, -1.00864308],
       [ 1.7315312 , -0.11935577,  0.98943914, ...,  0.64609167,
        -1.03067011, -0.12523071],
       [ 1.73187761, -0.87055909,  1.4692527 , ...,  0.64609167,
        -1.03067011, -1.07636976]])
```

```
#10. Split the data into training and testing
```

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.33)
```

```
x_train
```

```
array([[ 1.42565103,  0.02831026, -0.26327606, ..., -1.54776799,
        -1.03067011, -0.391208  ],
       [ 0.60777663, -0.03969785,  1.49170703, ..., -1.54776799,
         0.97024255, -1.14755267],
       [-0.04382089,  0.15843209,  1.12298331, ...,  0.64609167,
         0.97024255, -0.44810568],
       ...,
       [-0.70477148, -0.50556458,  0.64671517, ...,  0.64609167,
        -1.03067011,  1.30118798],
       [ 0.90984629,  1.23234033,  0.61835181, ...,  0.64609167,
        -1.03067011, -1.28401042],
       [ 0.61331919,  1.72876344, -0.20654934, ...,  0.64609167,
        -1.03067011,  0.35072171]])
```

```
x_train.shape
```

```
(6700, 13)
```

```
y_train.shape
```

```
(6700,)
```

```
x_test
```

```
array([[ 0.34450491,  0.76633458,  0.07708429, ...,  0.64609167,
         0.97024255, -1.30923126],
       [-1.5680256 , -0.12839201, -0.7749984 , ...,  0.64609167,
        -1.03067011, -0.39749464],
       [ 0.55754716,  1.04412244,  1.22461869, ...,  0.64609167,
         0.97024255, -0.50342774],
       ...,
       [-0.47510154, -1.63516385, -1.62353568, ..., -1.54776799,
        -1.03067011,  1.69261751],
       [ 1.30856439,  0.44657687, -0.10491395, ...,  0.64609167,
        -1.03067011, -1.36037561],
       [-1.64388943, -0.34902914, -1.24063028, ..., -1.54776799,
        -1.03067011,  1.33240141]])
```

```
x_test.shape
```

```
(3300, 13)
```