

## Assignment -2

### Data Visualization and Pre-processing

#### A - Load the dataset

```
import pandas as pd
df=pd.read_csv("Churn_Modelling.csv") # import dataset
print(df)
```

	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						
...	...	...	...	...	...	...
...						
9995	9996	15606229	Obijiaku	771	France	Male
39						
9996	9997	15569892	Johnstone	516	France	Male
35						
9997	9998	15584532	Liu	709	France	Female
36						
9998	9999	15682355	Sabbatini	772	Germany	Male
42						
9999	10000	15628319	Walker	792	France	Female
28						

	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
...	...	...	...	...		...
9995	5	0.00	2	1		0
9996	10	57369.61	1	1		1
9997	7	0.00	1	0		1
9998	3	75075.31	2	1		0

```
9999          4  130142.79          1          1          0
```

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

```
[10000 rows x 14 columns]
```

## B - Perform Below Visualizations.

### 1. Univariate Analysis

There are three ways to perform univariate analysis

#### i) Summary statistics

```
# Summary statistics
```

```
import pandas as pd
df=pd.read_csv("Churn_Modelling.csv")
```

```
#mean of CreditScore
```

```
M=df['CreditScore'].mean()
```

```
#median of CreditScore
```

```
Me=df['CreditScore'].median()
```

```
# standard deviation of CreditScore
```

```
std = df['CreditScore'].std()
```

```
print("mean value of CreditScore is {}".format(M))
print("median value of CreditScore is {}".format(Me))
print("Standard deviation of CreditScore is {}".format(std))
```

```
mean value of CreditScore is 650.5288
```

```
median value of CreditScore is 652.0
```

```
Standard deviation of CreditScore is 96.65329873613061
```

## ii) Frequency table

```
#Frequency table
import pandas as pd
df=pd.read_csv("Churn_Modelling.csv")

#frequency table for age
ft=df['Age'].value_counts()

print("Frequency table for Age is given below")
print("{}".format(ft))
```

Frequency table for Age is given below

```
37    478
38    477
35    474
36    456
34    447
```

```
...
92      2
82      1
88      1
85      1
83      1
```

Name: Age, Length: 70, dtype: int64

## iii) Charts

```
#Chart

import matplotlib.pyplot as plt
dfs = df.head() # print first five table from top
print(dfs)

#box plot for Balance column

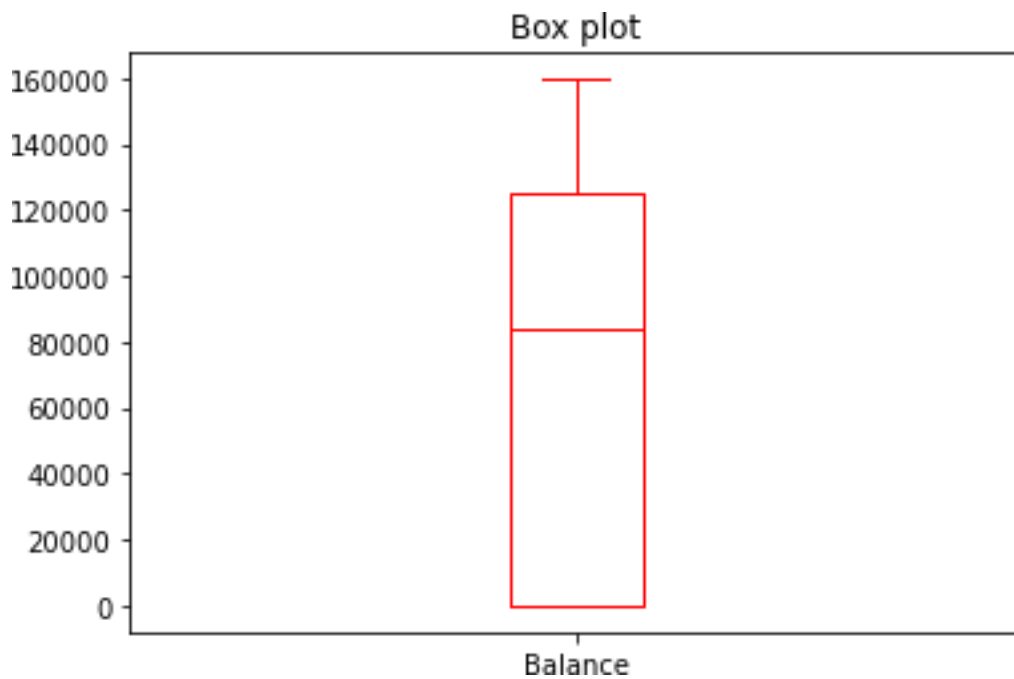
dfs.boxplot(column="Balance",grid=False,color="red")
plt.title('Box plot')
```

	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

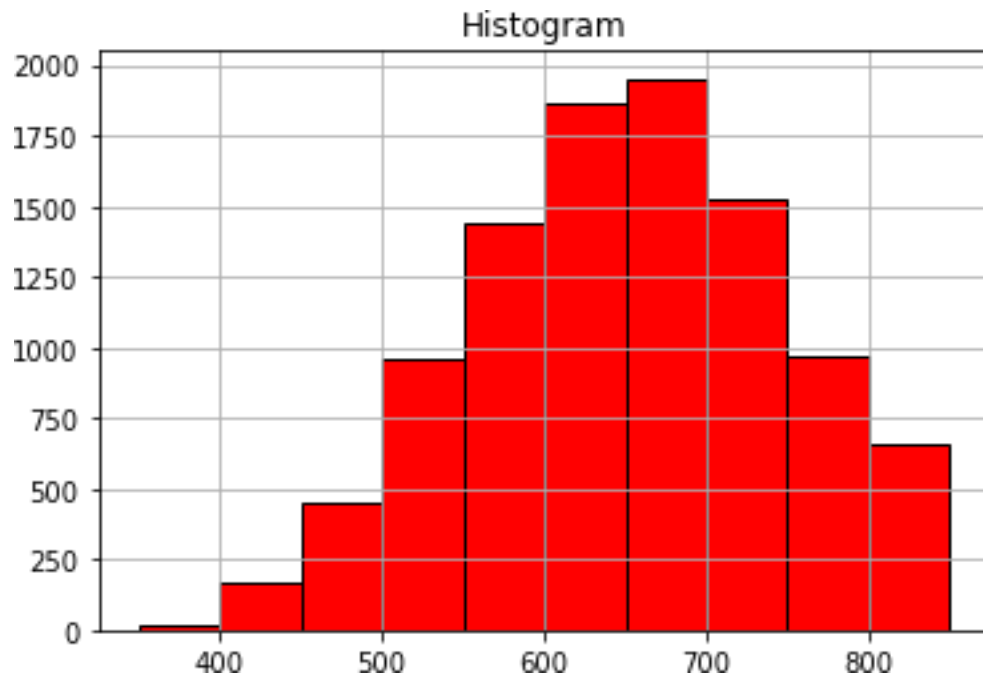
Text(0.5, 1.0, 'Box plot')



*# Histogram for Credit Score*

```
df.hist(column="CreditScore", grid=True, edgecolor='black', color='red')
plt.title('Histogram')
```

Text(0.5, 1.0, 'Histogram')



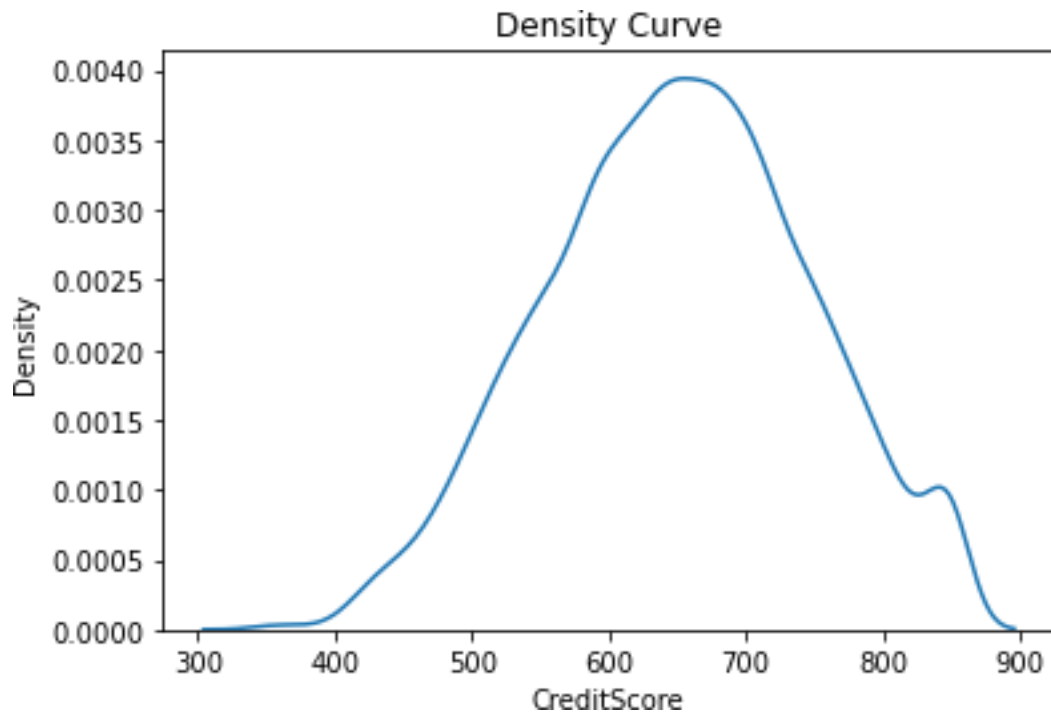
```
# Density curve
```

```
import seaborn as sns #statistical data visualization
```

```
sns.kdeplot(df['CreditScore'])
```

```
plt.title('Density Curve')
```

```
Text(0.5, 1.0, 'Density Curve')
```



## 2. Bi - Variate Analysis

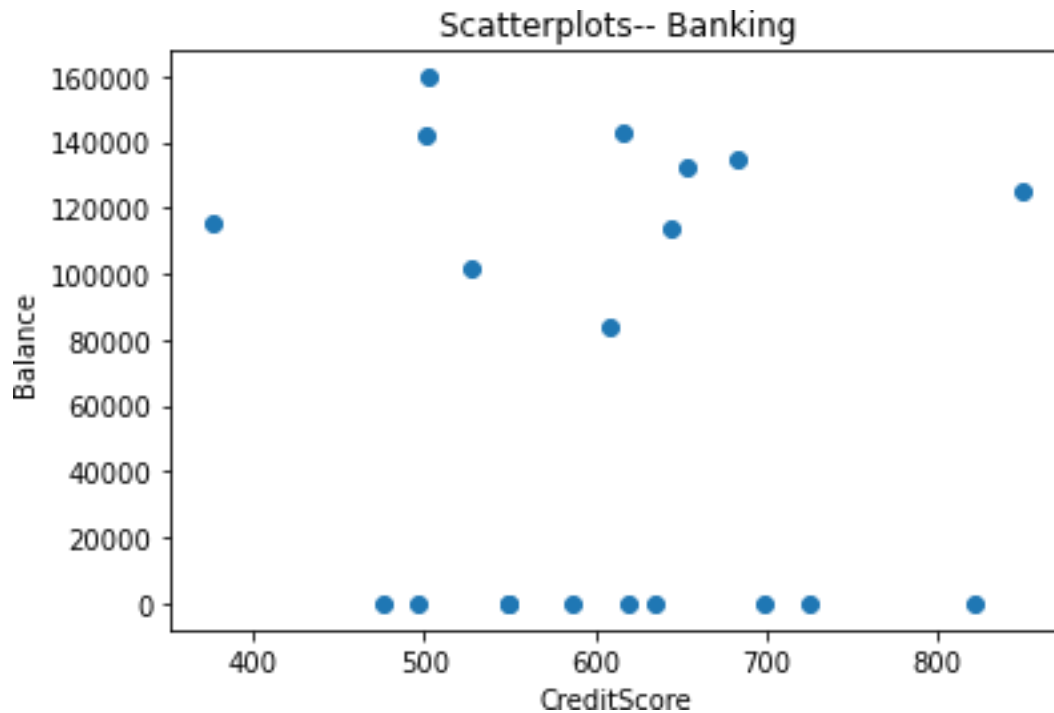
There are three common ways to perform bivariate analysis:

### i. Scatterplots

```
import matplotlib.pyplot as plt # library for charts
```

```
dfs1 = df.head(20)
plt.scatter(dfs1.CreditScore,dfs1.Balance)
plt.title('Scatterplots-- Banking')
plt.xlabel("CreditScore")
plt.ylabel("Balance")
```

```
Text(0, 0.5, 'Balance')
```



## ii. Correlation Coefficient

```
df.corr()
```

	RowNumber	CustomerId	CreditScore	Age	
Tenure \					
RowNumber	1.000000	0.004202	0.005840	0.000783	-
0.006495					
CustomerId	0.004202	1.000000	0.005308	0.009497	-
0.014883					
CreditScore	0.005840	0.005308	1.000000	-0.003965	
0.000842					
Age	0.000783	0.009497	-0.003965	1.000000	-
0.009997					
Tenure	-0.006495	-0.014883	0.000842	-0.009997	
1.000000					
Balance	-0.009067	-0.012419	0.006268	0.028308	-
0.012254					
NumOfProducts	0.007246	0.016972	0.012238	-0.030680	
0.013444					
HasCrCard	0.000599	-0.014025	-0.005458	-0.011721	
0.022583					
IsActiveMember	0.012044	0.001665	0.025651	0.085472	-
0.028362					
EstimatedSalary	-0.005988	0.015271	-0.001384	-0.007201	
0.007784					
Exited	-0.016571	-0.006248	-0.027094	0.285323	-
0.014001					

	Balance	NumOfProducts	HasCrCard	IsActiveMember	\
RowNumber	-0.009067	0.007246	0.000599	0.012044	
CustomerId	-0.012419	0.016972	-0.014025	0.001665	
CreditScore	0.006268	0.012238	-0.005458	0.025651	
Age	0.028308	-0.030680	-0.011721	0.085472	
Tenure	-0.012254	0.013444	0.022583	-0.028362	
Balance	1.000000	-0.304180	-0.014858	-0.010084	
NumOfProducts	-0.304180	1.000000	0.003183	0.009612	
HasCrCard	-0.014858	0.003183	1.000000	-0.011866	
IsActiveMember	-0.010084	0.009612	-0.011866	1.000000	
EstimatedSalary	0.012797	0.014204	-0.009933	-0.011421	
Exited	0.118533	-0.047820	-0.007138	-0.156128	

	EstimatedSalary	Exited
RowNumber	-0.005988	-0.016571
CustomerId	0.015271	-0.006248
CreditScore	-0.001384	-0.027094
Age	-0.007201	0.285323
Tenure	0.007784	-0.014001
Balance	0.012797	0.118533
NumOfProducts	0.014204	-0.047820
HasCrCard	-0.009933	-0.007138
IsActiveMember	-0.011421	-0.156128
EstimatedSalary	1.000000	0.012097
Exited	0.012097	1.000000

### iii. Simple Linear Regression

```
import statsmodels.api as sm
# response variable
y = df['CreditScore']

# explanatory variable
x = df[['Balance']]

#add constant to predictor variables
x = sm.add_constant(x)

#fit linear regression model
model = sm.OLS(y, x).fit()

#view model summary
print(model.summary())
```

#### OLS Regression Results

```
=====
=====
Dep. Variable:          CreditScore    R-squared:
0.000
```



```

Model:                                OLS    Adj. R-squared:
-0.000
Method:                               Least Squares    F-statistic:
0.3929
Date:                                 Sun, 25 Sep 2022    Prob (F-statistic):
0.531
Time:                                 13:06:05    Log-Likelihood:
-59900.
No. Observations:                     10000    AIC:
1.198e+05
Df Residuals:                         9998    BIC:
1.198e+05
Df Model:                             1

```

Covariance Type: nonrobust

```

=====
=====
              coef      std err          t      P>|t|      [0.025
0.975]
-----
const      649.7861      1.529      424.948      0.000      646.789
652.783
Balance     9.71e-06     1.55e-05      0.627      0.531     -2.07e-05
4.01e-05
=====
=====
Omnibus:      132.594    Durbin-Watson:
2.014
Prob(Omnibus):      0.000    Jarque-Bera (JB):
84.114
Skew:          -0.072    Prob(JB):
5.43e-19
Kurtosis:      2.574    Cond. No.
1.56e+05
=====
=====

```

#### Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

[2] The condition number is large, 1.56e+05. This might indicate that there are strong multicollinearity or other numerical problems.

### 3. Multi - Variate Analysis

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

#### i. A Matrix Scatterplot

#### ii. A Scatterplot with the Data Points Labelled by their Group

#### iii. A Profile Plot

#### iv. Calculating Summary Statistics for Multivariate Data

#### v. Means and Variances Per Group

#### vi. Between-groups Variance and Within-groups Variance for a Variable

#### vii. Between-groups Covariance and Within-groups Covariance for Two Variables

#### viii. Calculating Correlations for Multivariate Data

#### ix. Standardising Variables

```
df=sns.catplot(x="Geography",y="EstimatedSalary",hue="Gender",kind="swarm",data=df)
print(df)
```

```
/home/lokes/anaconda3/lib/python3.9/site-packages/seaborn/categorical.py:1296: UserWarning: 80.8% of the points cannot be placed; you may want to decrease the size of the markers or use stripplot.
```

```
warnings.warn(msg, UserWarning)
```

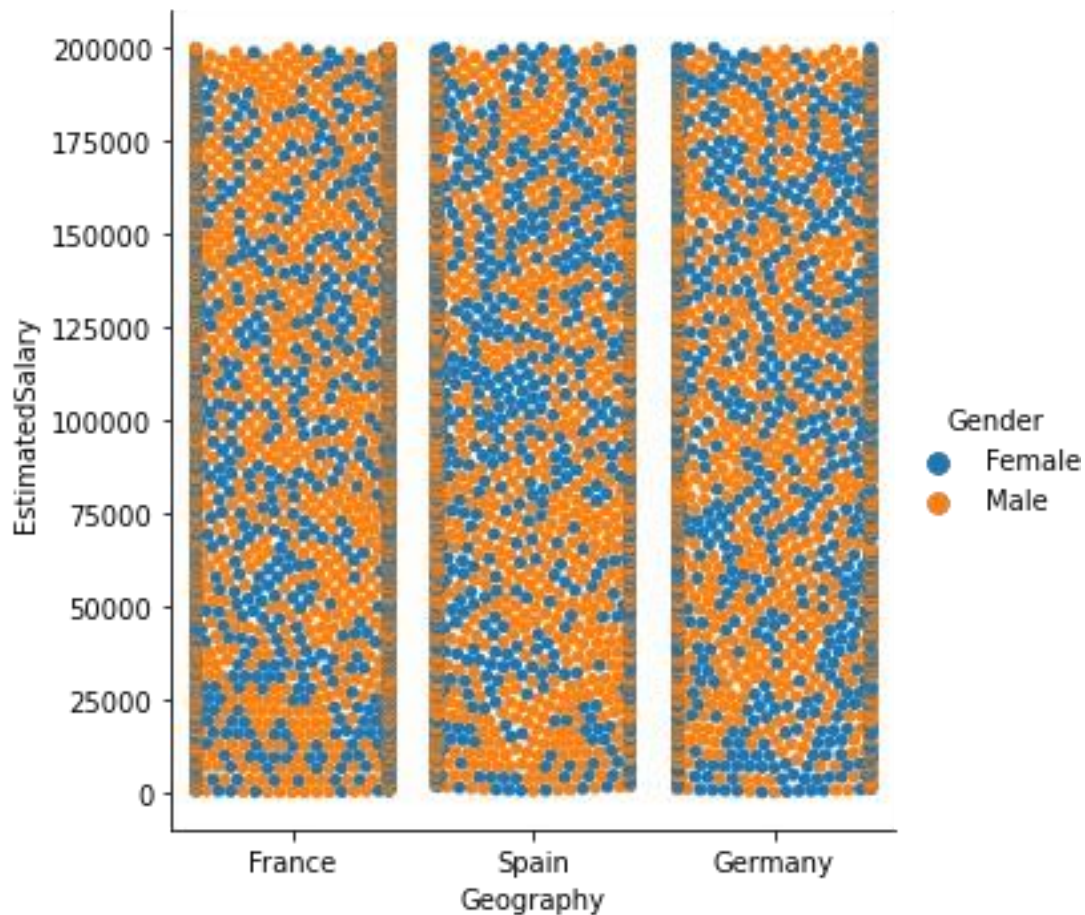
```
/home/lokes/anaconda3/lib/python3.9/site-packages/seaborn/categorical.py:1296: UserWarning: 62.1% of the points cannot be placed; you may want to decrease the size of the markers or use stripplot.
```

```
warnings.warn(msg, UserWarning)
```

```
/home/lokes/anaconda3/lib/python3.9/site-packages/seaborn/categorical.py:1296: UserWarning: 62.6% of the points cannot be placed; you may want to decrease the size of the markers or use stripplot.
```

```
warnings.warn(msg, UserWarning)
```

```
<seaborn.axisgrid.FacetGrid object at 0x7ffb0fd0b1c0>
```



#### 4. Perform descriptive statistics on the dataset.

```
#load data set into ld
ld= pd.read_csv("Churn_Modelling.csv")
five = ld.head() #for print first five rows

# information about used data set
ld.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
```

```
ld.describe() #description of the data in the Dataset
```

	RowNumber	CustomerId	CreditScore	Age
Tenure \				
count	10000.000000	1.000000e+04	10000.000000	10000.000000
mean	5000.500000	1.569094e+07	650.528800	38.921800
std	2886.89568	7.193619e+04	96.653299	10.487806
min	1.000000	1.556570e+07	350.000000	18.000000
25%	2500.750000	1.562853e+07	584.000000	32.000000
50%	5000.500000	1.569074e+07	652.000000	37.000000
75%	7500.250000	1.575323e+07	718.000000	44.000000
max	10000.000000	1.581569e+07	850.000000	92.000000

	Balance	NumOfProducts	HasCrCard	IsActiveMember \
count	10000.000000	10000.000000	10000.000000	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

## 5. Handle the Missing values.

```
ld.isnull().any()
```

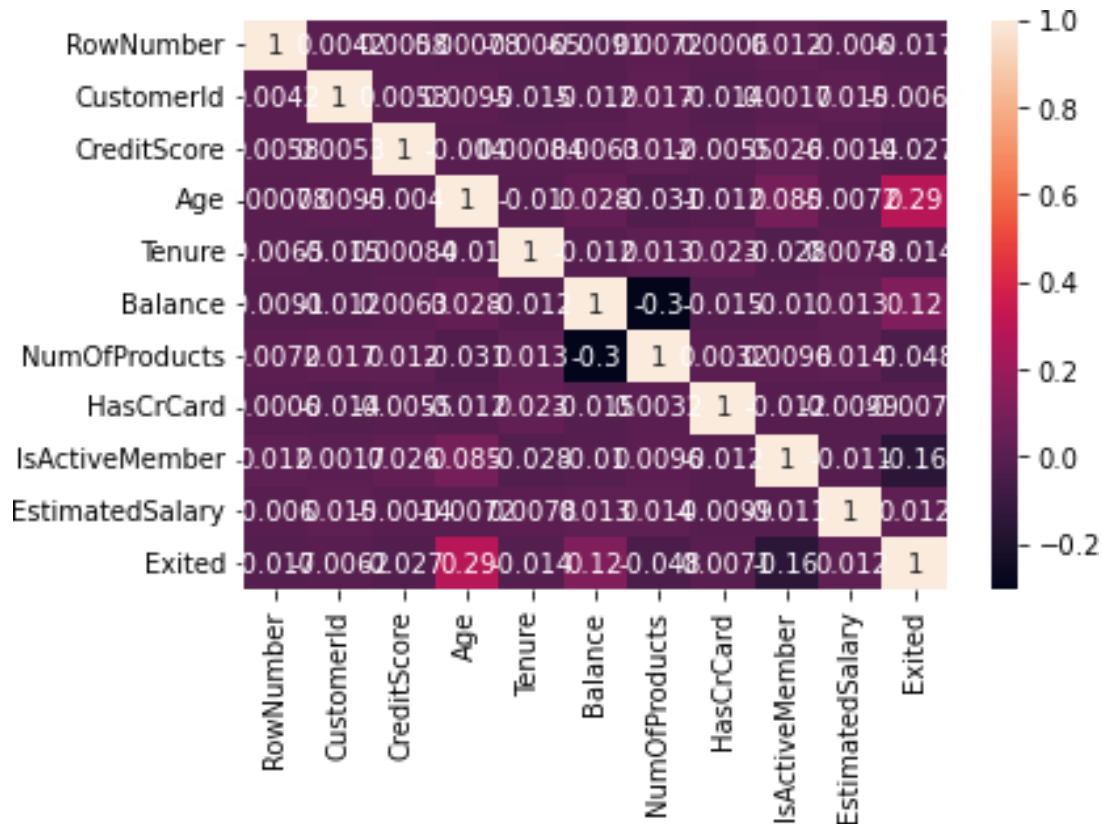
```
RowNumber      False
CustomerId      False
Surname         False
CreditScore     False
Geography       False
Gender          False
Age            False
Tenure          False
Balance         False
NumOfProducts  False
HasCrCard       False
IsActiveMember False
EstimatedSalary False
Exited        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

```
ld.isnull().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
```

```
sns.heatmap(ld.corr(),annot=True) # heatmap -a plot of rectangular
data as a color-encoded matrix
```

```
<AxesSubplot:>
```



## 6. Find the outliers and replace the outliers

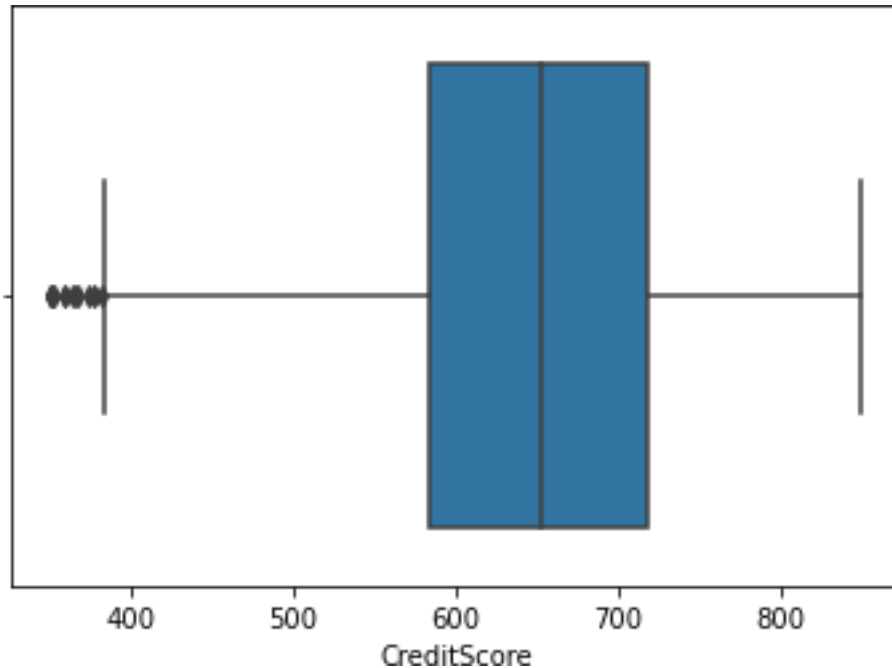
*#occurence of outliers*

```
ld1= pd.read_csv("Churn_Modelling.csv")
sns.boxplot(ld1.CreditScore)
```

/home/lokes/anaconda3/lib/python3.9/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(
```

```
<AxesSubplot:xlabel='CreditScore'>
```

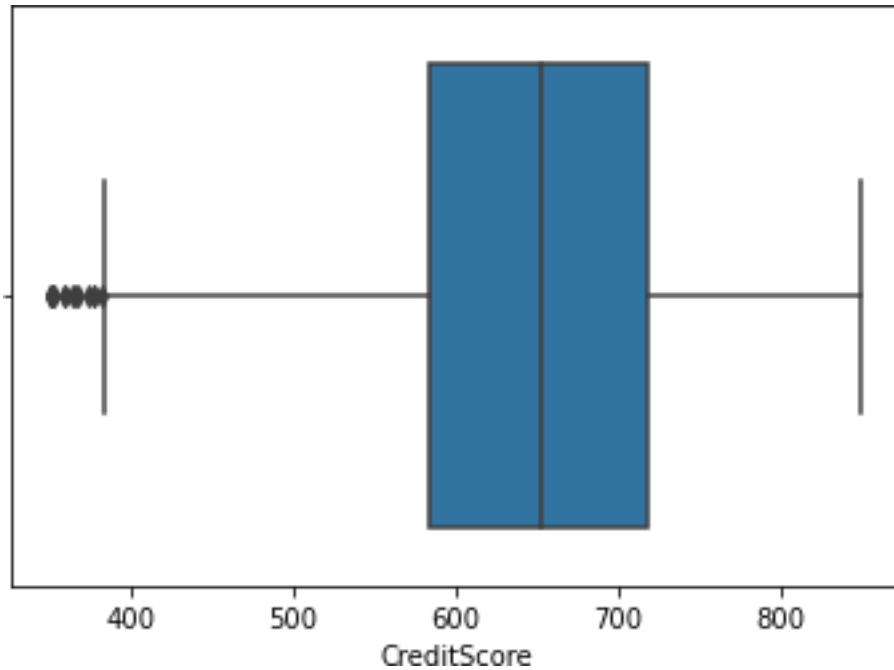


*#Use Mean Detection and Nearest Fill Methods - Outliers*

```
Q1= ld1.CreditScore.quantile(0.25)
Q3=ld1.CreditScore.quantile(0.75)
IQR=Q3-Q1
upper_limit =Q3 + 1.5*IQR
lower_limit =Q1 - 1.5*IQR
ld1['CreditScore'] =
np.where(ld1['CreditScore']>upper_limit,30,ld1['CreditScore'])
sns.boxplot(ld1.CreditScore)

/home/lokes/anaconda3/lib/python3.9/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(

<AxesSubplot:xlabel='CreditScore'>
```



## 7. Check for Categorical columns and perform encoding.

```
ld1.head(5)
```

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

```
#label encoder
```

```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
ld1.Gender= le.fit_transform(ld1.Gender)
ld1.head(5)
```

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

```
#one hot encoding
```

```
ld1_main=pd.get_dummies(ld1,columns=['Geography'])
ld1_main.head()
```

	RowNumber	CustomerId	Surname	CreditScore	Gender	Age	Tenure
\							
0	1	15634602	Hargrave	619	0	42	2
1	2	15647311	Hill	608	0	41	1
2	3	15619304	Onio	502	0	42	8
3	4	15701354	Boni	699	0	39	1

4	5	15737888	Mitchell	850	0	43	2
---	---	----------	----------	-----	---	----	---

	Balance	NumOfProducts	HasCrCard	IsActiveMember
EstimatedSalary \				
0	0.00	1	1	1
101348.88				
1	83807.86	1	0	1
112542.58				
2	159660.80	3	1	0
113931.57				
3	0.00	2	0	0
93826.63				
4	125510.82	1	1	1
79084.10				

	Exited	Geography_France	Geography_Germany	Geography_Spain
0	1	1	0	0
1	0	0	0	1
2	1	1	0	0
3	0	1	0	0
4	0	0	0	1

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

*#Splitting the Dataset into the Independent Feature Matrix*

```
df=pd.read_csv("Churn_Modelling.csv")
```

```
X = df.iloc[:, :-1].values
```

```
print(X)
```

```
[[1 15634602 'Hargrave' ... 1 1 101348.88]
 [2 15647311 'Hill' ... 0 1 112542.58]
 [3 15619304 'Onio' ... 1 0 113931.57]
 ...
 [9998 15584532 'Liu' ... 0 1 42085.58]
 [9999 15682355 'Sabbatini' ... 1 0 92888.52]
 [10000 15628319 'Walker' ... 1 0 38190.78]]
```

*#Extracting the Dataset to Get the Dependent Vector*

```
Y = df.iloc[:, -1].values
```

```
print(Y)
```

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

## 9. Scale the independent variables

```
w = df.head()
```

```
q = w[['Age','Balance','EstimatedSalary']] #splitting the dataset into  
measureable values
```

```
q
```

	Age	Balance	EstimatedSalary
0	42	0.00	101348.88
1	41	83807.86	112542.58
2	42	159660.80	113931.57
3	39	0.00	93826.63
4	43	125510.82	79084.10

```
from sklearn.preprocessing import scale # library for scalling
from sklearn.preprocessing import MinMaxScaler
mm = MinMaxScaler()
```

```
x_scaled = mm.fit_transform(q)
x_scaled
```

```
array([[0.75      , 0.        , 0.63892099],
       [0.5       , 0.52491194, 0.96014087],
       [0.75      , 1.        , 1.        ],
       [0.        , 0.        , 0.42305883],
       [1.        , 0.78610918, 0.        ]])
```

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_ss = sc.fit_transform(q)
x_ss
```

```
array([[ 0.44232587, -1.13763618,  0.09337626],
       [-0.29488391,  0.15434425,  0.96285595],
       [ 0.44232587,  1.32369179,  1.07074687],
       [-1.76930347, -1.13763618, -0.49092058],
       [ 1.17953565,  0.79723632, -1.6360585 ]])
```

```
from sklearn.preprocessing import scale
X_scaled=pd.DataFrame(scale(q),columns=q.columns)
X_scale=X_scaled.head()
X_scale
```

	Age	Balance	EstimatedSalary
0	0.442326	-1.137636	0.093376
1	-0.294884	0.154344	0.962856
2	0.442326	1.323692	1.070747
3	-1.769303	-1.137636	-0.490921
4	1.179536	0.797236	-1.636059

## 10. Split the data into training and testing

```
x= df[['Age','Balance','EstimatedSalary']]
x
```

	Age	Balance	EstimatedSalary
0	42	0.00	101348.88
1	41	83807.86	112542.58

```

2      42  159660.80      113931.57
3      39      0.00      93826.63
4      43  125510.82      79084.10
...    ...    ...    ...
9995   39      0.00      96270.64
9996   35   57369.61     101699.77
9997   36      0.00      42085.58
9998   42   75075.31      92888.52
9999   28  130142.79      38190.78

```

```
[10000 rows x 3 columns]
```

```

y = df['Balance']
y

```

```

0      0.00
1    83807.86
2    159660.80
3      0.00
4    125510.82
...
9995      0.00
9996    57369.61
9997      0.00
9998    75075.31
9999   130142.79

```

```
Name: Balance, Length: 10000, dtype: float64
```

```
#scaling
```

```

from sklearn.preprocessing import StandardScaler, MinMaxScaler
sc = StandardScaler()
x_scaled1 = sc.fit_transform(x)
x_scaled1

```

```

array([[ 0.29351742, -1.22584767,  0.02188649],
       [ 0.19816383,  0.11735002,  0.21653375],
       [ 0.29351742,  1.33305335,  0.2406869 ],
       ...,
       [-0.27860412, -1.22584767, -1.00864308],
       [ 0.29351742, -0.02260751, -0.12523071],
       [-1.04143285,  0.85996499, -1.07636976]])

```

```
#train and test data
```

```

from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x_scaled1, y,
test_size = 0.3, random_state = 0)

```

```
x_train
```

```

array([[ -0.56466489,  1.11721307, -0.77021814],
       [ 0.00745665, -1.22584767, -1.39576675],
       [ 3.53553951,  1.35419118, -1.49965629],

```

```
.../  
[-0.37395771,  1.35890908,  1.41441489],  
[-0.08789694, -1.22584767,  0.84614739],  
[ 0.86563897,  0.50630343,  0.32630495]])
```

```
x_train.shape
```

```
(7000, 3)
```

```
x_test
```

```
array([[ -0.37395771,  0.87532296,  1.61304597],  
       [ 0.10281024,  0.42442221,  0.49753166],  
       [ 0.29351742,  0.30292727, -0.4235611 ]],  
      .../  
       [ 0.10281024,  1.46672809,  1.17045451],  
       [ 2.86806437,  1.25761599, -0.50846777],  
       [ 0.96099256,  0.19777742, -1.15342685]])
```

```
x_test.shape
```

```
(3000, 3)
```

```
y_train
```

```
7681    146193.60  
9031         0.00  
3691    160979.68  
202         0.00  
5625    143262.04
```

```
...  
9225    120074.97  
4859    114440.24  
3264    161274.05  
9845         0.00  
2732    108076.33
```

```
Name: Balance, Length: 7000, dtype: float64
```

```
y_test
```

```
9394    131101.04  
898     102967.41  
2398     95386.82  
5906    112079.58  
2343    163034.82
```

```
...  
4004         0.00  
7375     80926.02  
9307    168001.34  
8394    154953.94  
5233     88826.07
```

```
Name: Balance, Length: 3000, dtype: float64
```

