

# Assignment -2

## Python Programming

Assignment Date	21 September 2022
Student Name	Mr. Jeya prathap P
Student Register Number	910619104035
Maximum Marks	

### 1.Importing necessary Libraries and Dataset

```
In [143]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

import scipy.stats
import statsmodels.api as sm
#import statsmodels.stats.api as sms
import statsmodels.formula.api as smf
from statsmodels.stats.stattools import jarque_bera

sns.set_style('darkgrid')
sns.set(font_scale=1.3)

data = pd.read_csv('Churn_Modelling.csv')
data
```

```
Out[143]:
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveM
0	1	15634602	Hargrave	619	France	Female	42	2	0.00	1	1	
1	2	15647311	Hill	608	Spain	Female	41	1	83807.86	1	0	
2	3	15619304	Onio	502	France	Female	42	8	159660.80	3	1	
3	4	15701354	Boni	699	France	Female	39	1	0.00	2	0	
4	5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	1	
...	...	...	...	...	...	...	...	...	...	...	...	...
9995	9996	15606229	Obijiaku	771	France	Male	39	5	0.00	2	1	
9996	9997	15569892	Johnstone	516	France	Male	35	10	57369.61	1	1	
9997	9998	15584532	Liu	709	France	Female	36	7	0.00	1	0	
9998	9999	15682355	Sabbatini	772	Germany	Male	42	3	75075.31	2	1	
9999	10000	15628319	Walker	792	France	Female	28	4	130142.79	1	1	

10000 rows x 14 columns

### SHAPE AND SIZE OF THE DATASET

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

```
Out[101]: (10000, 14)
```

### DataTypes

```
In [103]: data.describe(include='object')
```

```
Out[103]:
```

```
In [102]: data.dtypes
```

```
Out[102]: RowNumber          int64
CustomerId        int64
Surname           object
CreditScore       int64
Geography         object
Gender            object
Age              int64
Tenure            int64
Balance           float64
NumOfProducts    int64
HasCrCard         int64
IsActiveMember    int64
EstimatedSalary  float64
Exited            int64
dtype: object
```

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

```
In [104]: data.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
```

Describe function to watch the Mean,Medium,etc

In [105]: data.describe()

Out[105]:

	RowNumber	CustomerId	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember
count	10000.00000	1.000000e+04	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.00000	10000.000000
mean	5000.50000	1.569094e+07	650.528800	38.921800	5.012800	76485.889288	1.530200	0.70550	0.515100
std	2886.89568	7.193619e+04	96.653299	10.487806	2.892174	62397.405202	0.581654	0.45584	0.499797
min	1.00000	1.556570e+07	350.000000	18.000000	0.000000	0.000000	1.000000	0.00000	0.000000
25%	2500.75000	1.562853e+07	584.000000	32.000000	3.000000	0.000000	1.000000	0.00000	0.000000
50%	5000.50000	1.569074e+07	652.000000	37.000000	5.000000	97198.540000	1.000000	1.00000	1.000000
75%	7500.25000	1.575323e+07	718.000000	44.000000	7.000000	127644.240000	2.000000	1.00000	1.000000
max	10000.00000	1.581569e+07	850.000000	92.000000	10.000000	250898.090000	4.000000	1.00000	1.000000

## 2. Perform Visualizations

### UNIVARIATE ANALYSIS

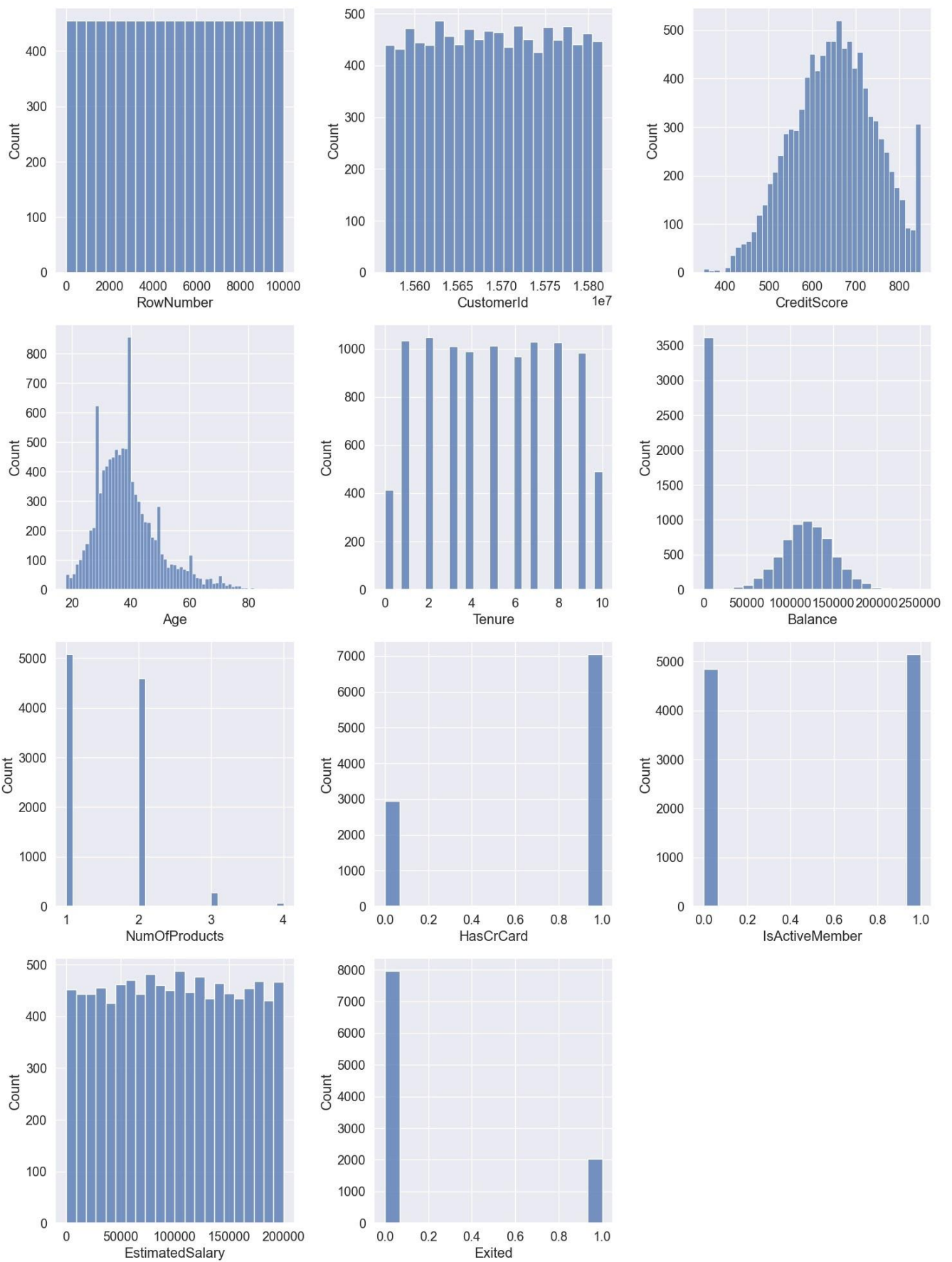
#### Histogram

```
In [106]: cols = 3
rows = 4
num_cols = data.select_dtypes(exclude='object').columns
fig = plt.figure( figsize=(cols*5, rows*5))
for i, col in enumerate(num_cols):

    ax=fig.add_subplot(rows,cols,i+1)

    sns.histplot(x = data[col], ax = ax)

fig.tight_layout()
plt.show()
```



## Distplot

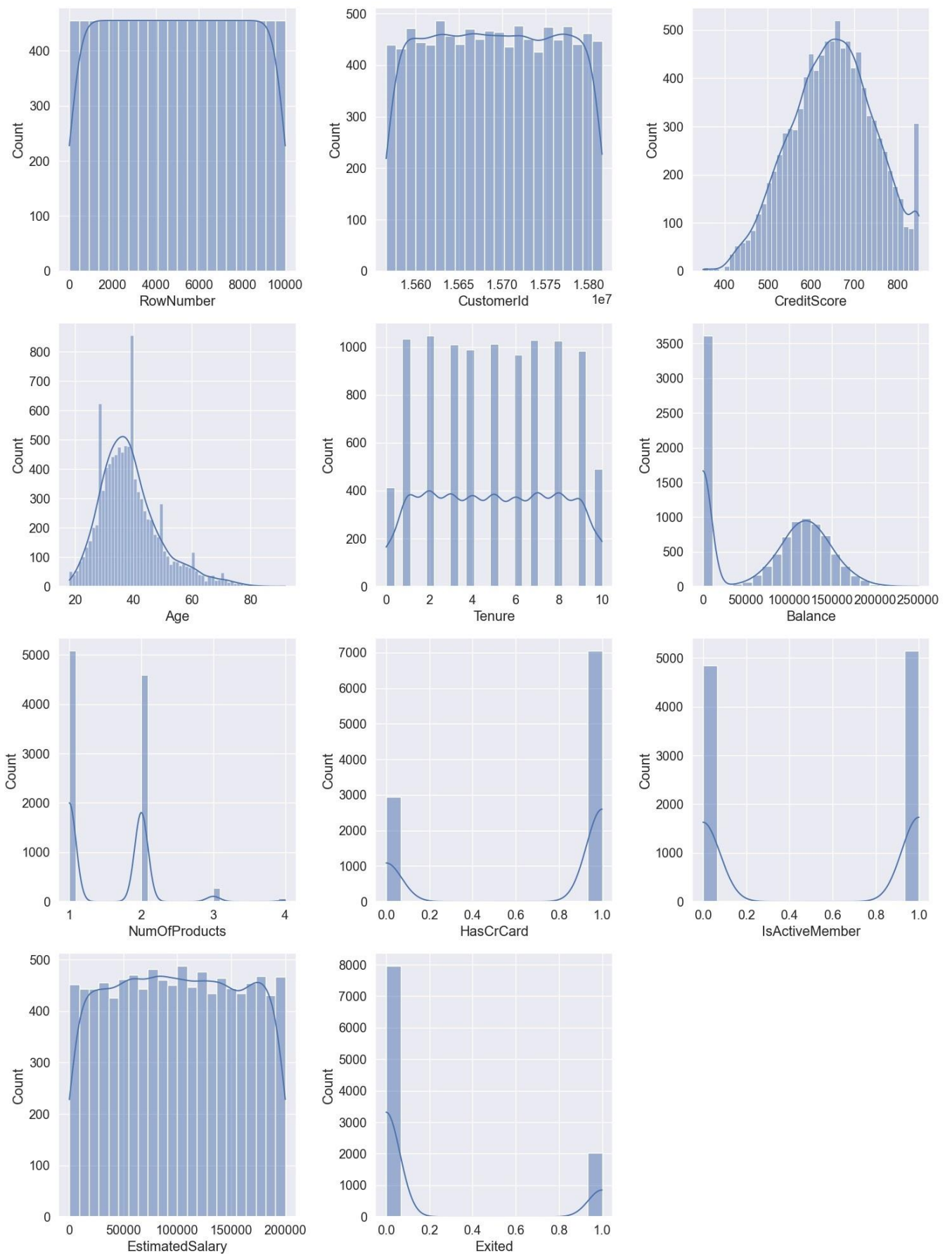
```
In [107]: cols = 3
rows = 4
num_cols = data.select_dtypes(exclude='object').columns
fig = plt.figure(figsize=(cols*5, rows*5))
for i, col in enumerate(num_cols):

    ax=fig.add_subplot(rows,cols,i+1)

    sns.histplot(x = data[col], ax = ax, data=data, kde='True')

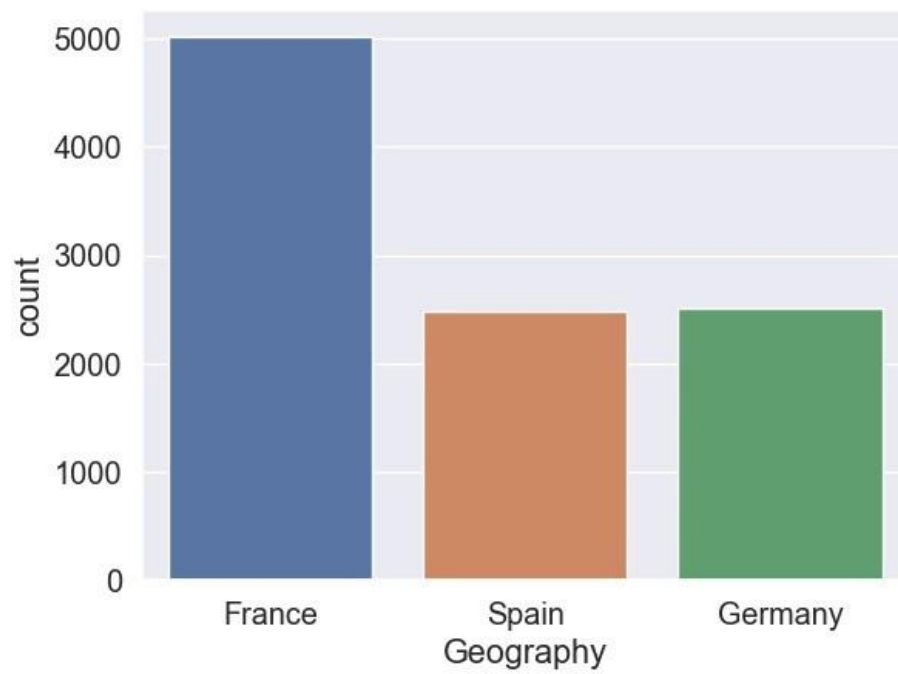
fig.tight_layout()
```

```
plt.show()
```



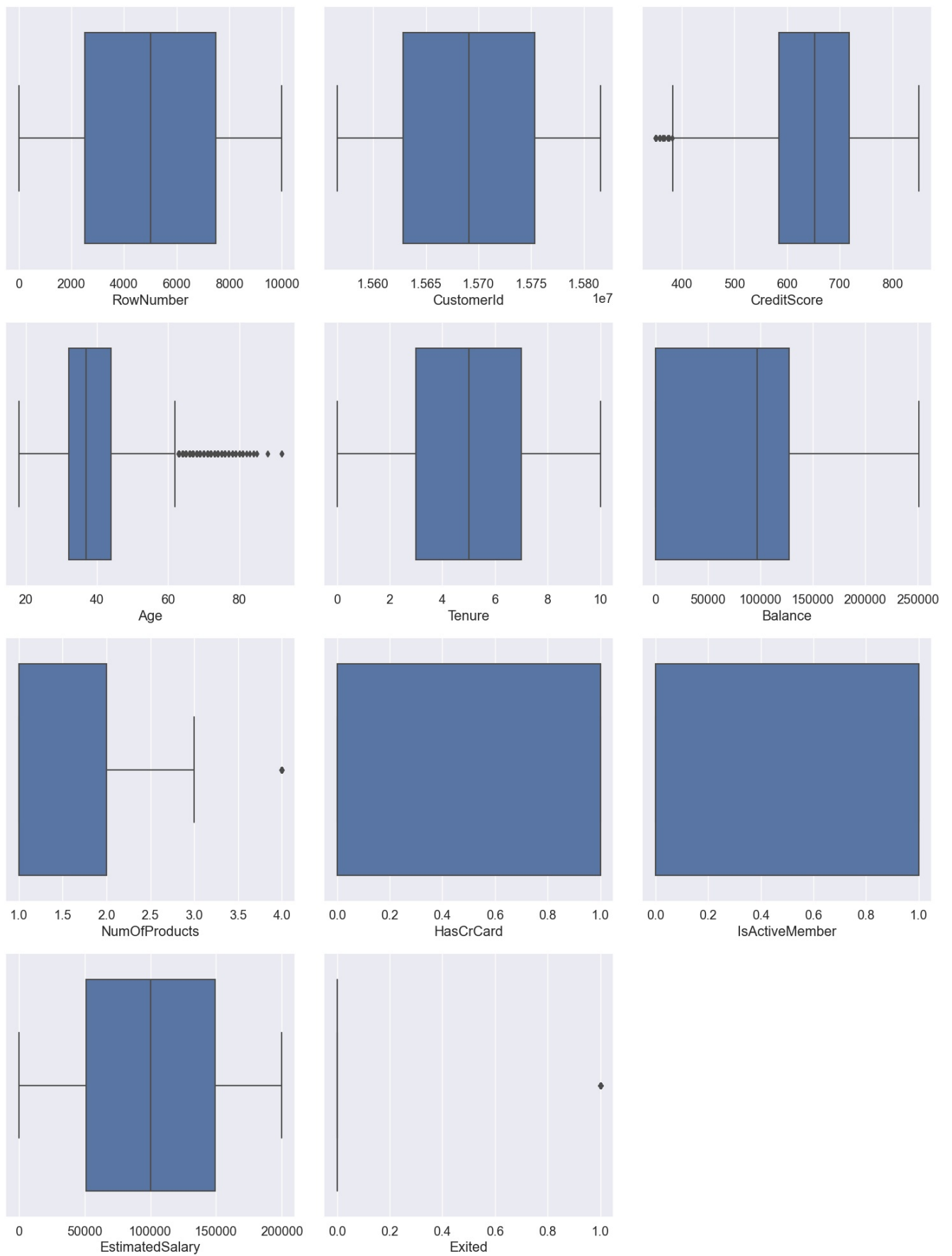
## Countplot

```
In [168]: sns.countplot(x = 'Geography', data=data)
plt.show()
```



## Boxplot

```
In [109]: cols = 3
rows = 4
num_cols = data.select_dtypes(exclude='object').columns
fig = plt.figure(figsize=(cols*5, rows*5))
for i, col in enumerate(num_cols):
    ax=fig.add_subplot(rows,cols,i+1)
    sns.boxplot(x = data[col], ax = ax)
fig.tight_layout()
plt.show()
```



## Bivariate Analysis

```
In [110]: data.head()
```

Out[110]:	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMem
0	1	15634602	Hargrave	619	France	Female	42	2	0.00	1	1	
1	2	15647311	Hill	608	Spain	Female	41	1	83807.86	1	0	
2	3	15619304	Onio	502	France	Female	42	8	159660.80	3	1	
3	4	15701354	Boni	699	France	Female	39	1	0.00	2	0	
4	5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	1	

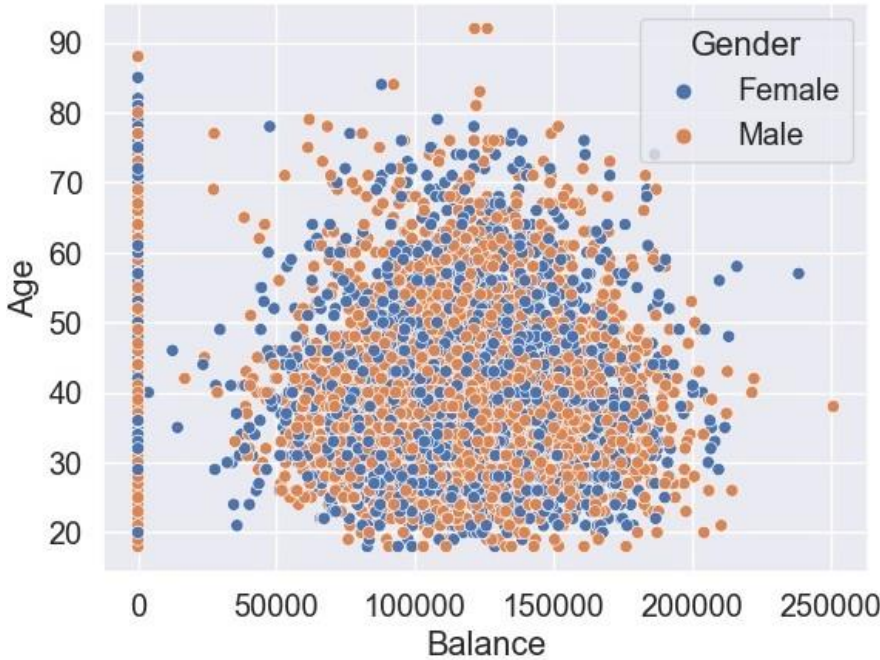
```
In [111]: data.info()

<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
```

## Numerical variable vs Numerical variable

### Scatterplot

```
In [112]: sns.scatterplot(x='Balance', y='Age', data = data, hue='Gender')
plt.show()
```

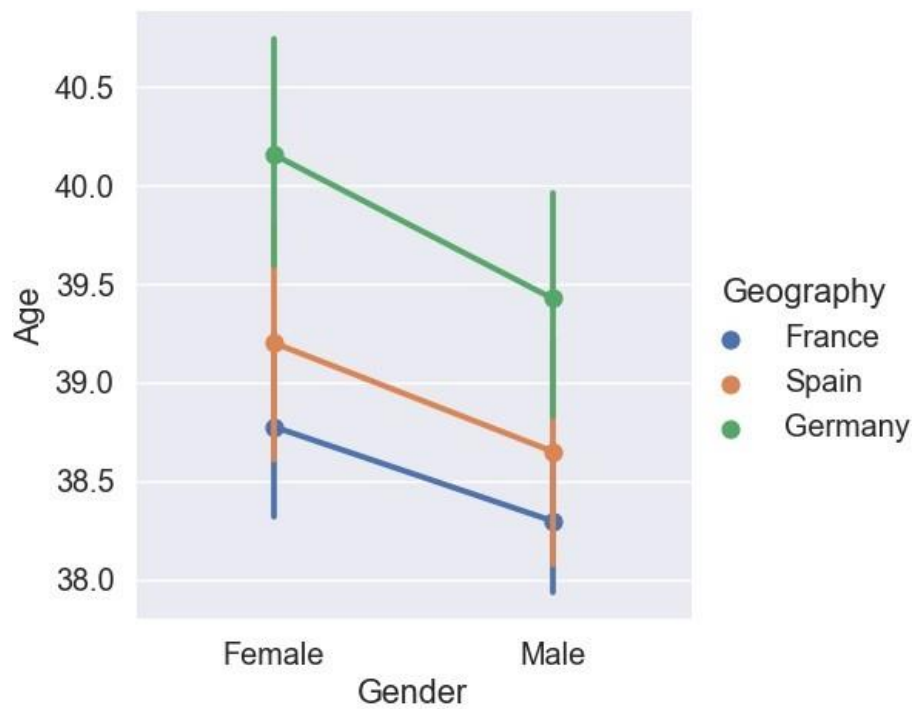


## Categorical vs Categorical

### Catplot

```
In [113]: sns.catplot(x='Gender', y='Age', data=data, kind='point', hue='Geography')
plt.show()
```





## Multivariate Analysis

### PairPlot

```
In [114]: sns.pairplot(data)
```

```
Out[114]: <seaborn.axisgrid.PairGrid at 0x24161886980>
```



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

```
In [144]: data.describe(include='all')
```

Out[144]:	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProduct
<b>count</b>	10000.00000	1.000000e+04	10000	10000.000000	10000	10000	10000.000000	10000.000000	10000.000000	10000.00000
<b>unique</b>	NaN	NaN	2932	NaN	3	2	NaN	NaN	NaN	Na
<b>top</b>	NaN	NaN	Smith	NaN	France	Male	NaN	NaN	NaN	Na
<b>freq</b>	NaN	NaN	32	NaN	5014	5457	NaN	NaN	NaN	Na
<b>mean</b>	5000.50000	1.569094e+07	NaN	650.528800	NaN	NaN	38.921800	5.012800	76485.889288	1.53020
<b>std</b>	2886.89568	7.193619e+04	NaN	96.653299	NaN	NaN	10.487806	2.892174	62397.405202	0.58165
<b>min</b>	1.00000	1.556570e+07	NaN	350.000000	NaN	NaN	18.000000	0.000000	0.000000	1.00000
<b>25%</b>	2500.75000	1.562853e+07	NaN	584.000000	NaN	NaN	32.000000	3.000000	0.000000	1.00000
<b>50%</b>	5000.50000	1.569074e+07	NaN	652.000000	NaN	NaN	37.000000	5.000000	97198.540000	1.00000
<b>75%</b>	7500.25000	1.575323e+07	NaN	718.000000	NaN	NaN	44.000000	7.000000	127644.240000	2.00000
<b>max</b>	10000.00000	1.581569e+07	NaN	850.000000	NaN	NaN	92.000000	10.000000	250898.090000	4.00000

```
In [145]: data.count()
```

```
Out[145]: RowNumber      10000
CustomerId    10000
Surname       10000
CreditScore   10000
Geography     10000
Gender        10000
Age           10000
Tenure        10000
Balance       10000
NumOfProducts 10000
HasCrCard     10000
IsActiveMember 10000
EstimatedSalary 10000
Exited        10000
dtype: int64
```

4. Handle the Missing values.

Fill with Zeros for NAN values

```
In [146]: a = data.fillna(0)
a
```

Out[146]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveM
	0	1	15634602	Hargrave	619	France	Female	42	2	0.00	1	1
	1	2	15647311	Hill	608	Spain	Female	41	1	83807.86	1	0
	2	3	15619304	Onio	502	France	Female	42	8	159660.80	3	1
	3	4	15701354	Boni	699	France	Female	39	1	0.00	2	0
	4	5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	1
	...	...	...	...	...	...	...	...	...	...	...	...
	9995	9996	15606229	Obijiaku	771	France	Male	39	5	0.00	2	1
	9996	9997	15569892	Johnstone	516	France	Male	35	10	57369.61	1	1
	9997	9998	15584532	Liu	709	France	Female	36	7	0.00	1	0
	9998	9999	15682355	Sabbatini	772	Germany	Male	42	3	75075.31	2	1
	9999	10000	15628319	Walker	792	France	Female	28	4	130142.79	1	1

10000 rows x 14 columns

5. Find the outliers and replace the outliers

```
In [147]: a
```

Out[147]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveM
	0	1	15634602	Hargrave	619	France	Female	42	2	0.00	1	1
	1	2	15647311	Hill	608	Spain	Female	41	1	83807.86	1	0
	2	3	15619304	Onio	502	France	Female	42	8	159660.80	3	1
	3	4	15701354	Boni	699	France	Female	39	1	0.00	2	0
	4	5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	1
	...	...	...	...	...	...	...	...	...	...	...	...
	9995	9996	15606229	Obijiaku	771	France	Male	39	5	0.00	2	1
	9996	9997	15569892	Johnstone	516	France	Male	35	10	57369.61	1	1
	9997	9998	15584532	Liu	709	France	Female	36	7	0.00	1	0
	9998	9999	15682355	Sabbatini	772	Germany	Male	42	3	75075.31	2	1
	9999	10000	15628319	Walker	792	France	Female	28	4	130142.79	1	1

10000 rows x 14 columns

```
In [148]: missing_values=data.isnull().sum()
missing_values[missing_values>0]/len(data)*100
```

```
Out[148]: Series([], dtype: float64)
```

```
In [149]: data.drop(['RowNumber','Exited','CustomerId','Surname','Geography','Gender'],axis=1,inplace=True)
data.head()
```

Out[149]:

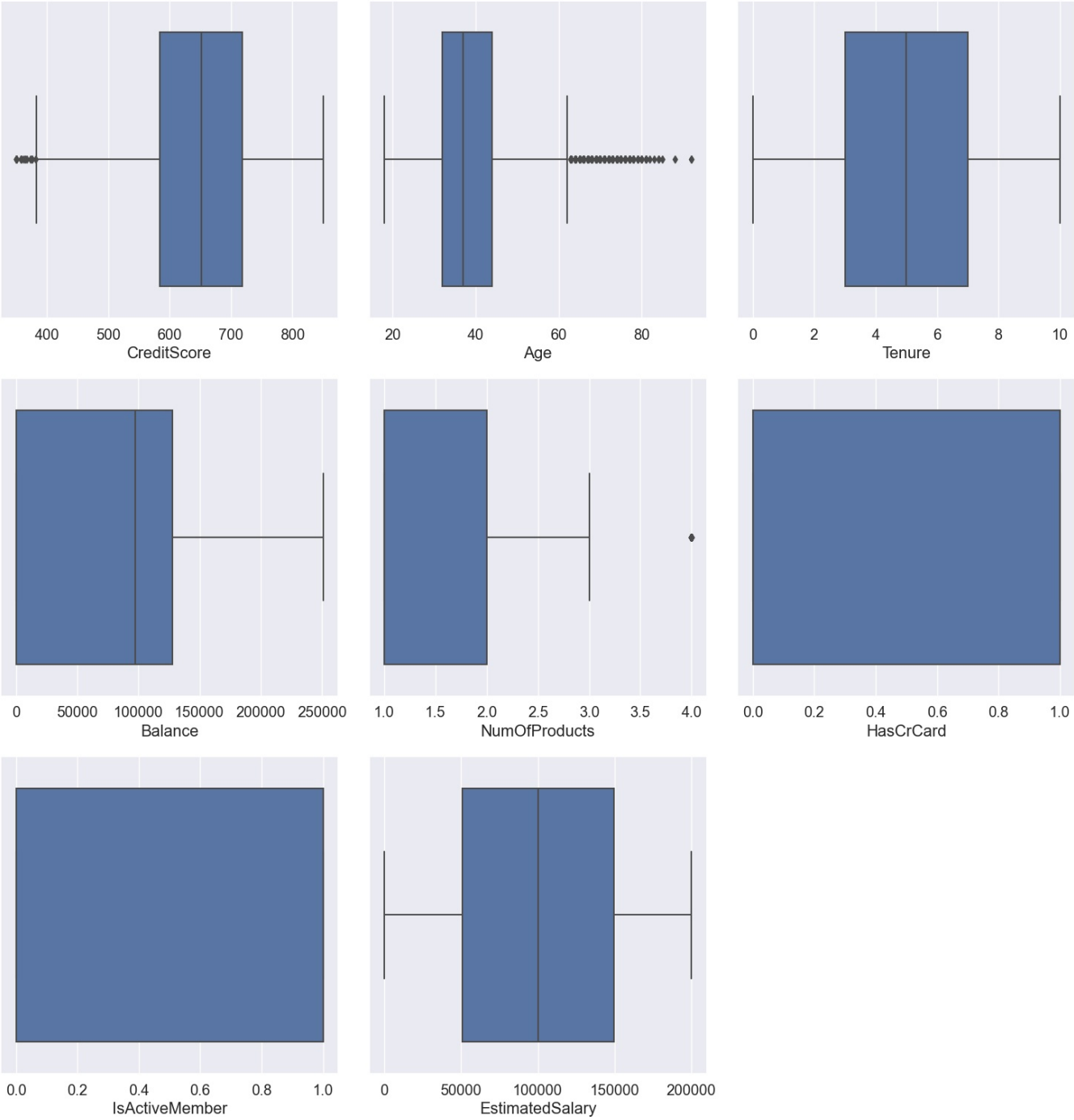
	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary
0	619	42	2	0.00	1	1	1	101348.88
1	608	41	1	83807.86	1	0	1	112542.58
2	502	42	8	159660.80	3	1	0	113931.57
3	699	39	1	0.00	2	0	0	93826.63
4	850	43	2	125510.82	1	1	1	79084.10

```
In [150]: cols = 3
rows = 4
num_cols = data.select_dtypes(exclude='object').columns
fig = plt.figure(figsize=(cols*5, rows*5))
for i, col in enumerate(num_cols):

    ax=fig.add_subplot(rows,cols,i+1)

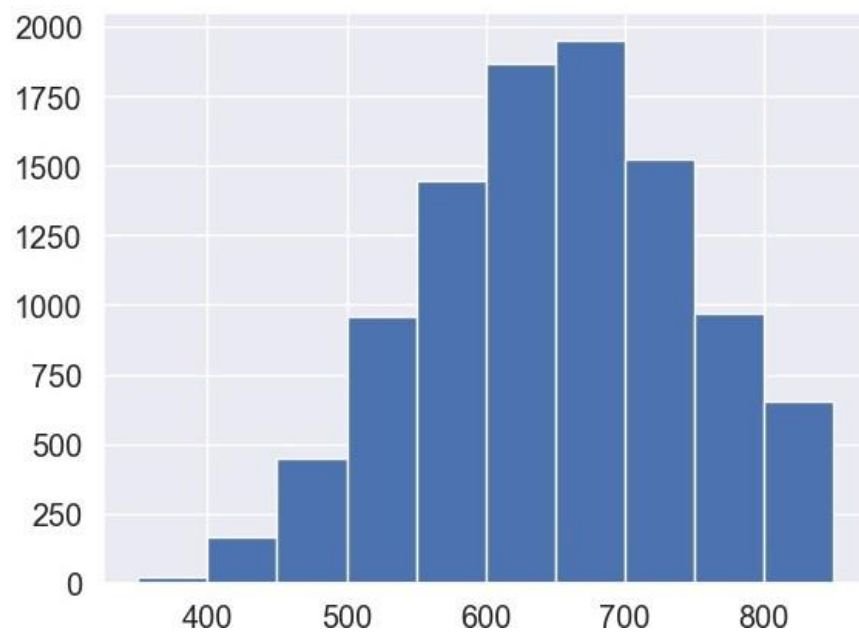
    sns.boxplot(x = data[col], ax = ax)

fig.tight_layout()
plt.show()
```



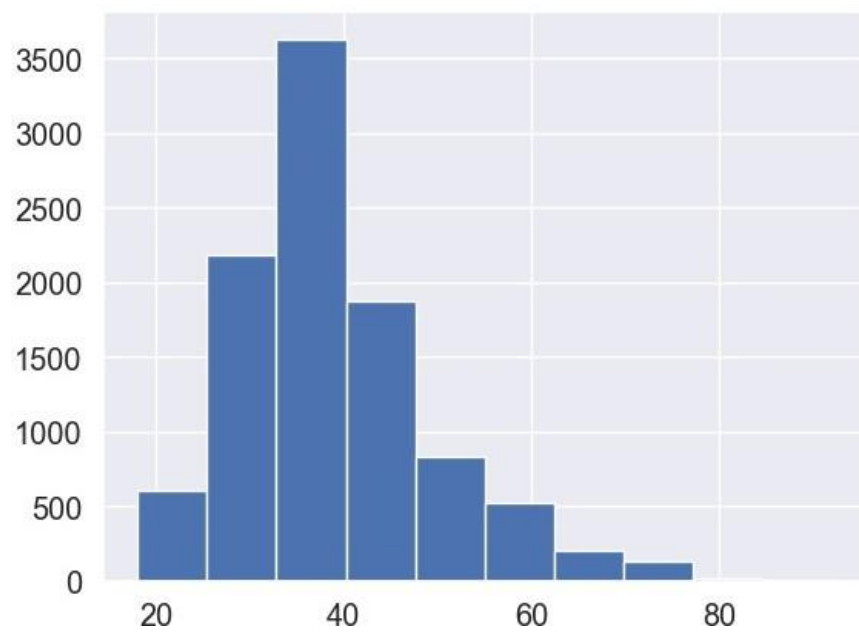
```
In [123]: data['CreditScore'].hist()
```

Out[123]: <AxesSubplot: >



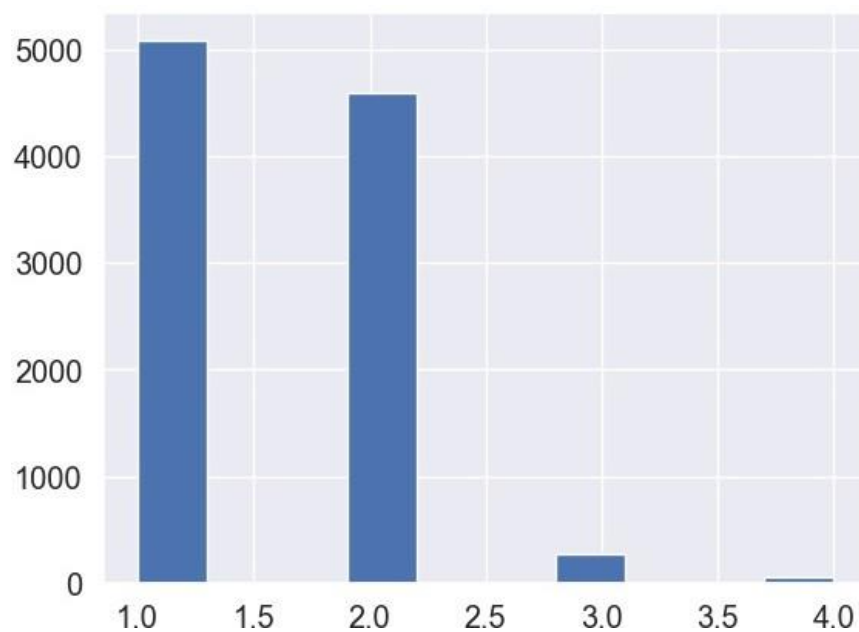
```
In [124]: data['Age'].hist()
```

```
Out[124]: <AxesSubplot: >
```



```
In [125]: data['NumOfProducts'].hist()
```

```
Out[125]: <AxesSubplot: >
```



```
In [151]: print('Skewness value of Age: ',data['Age'].skew())
Age_mean = data['Age'].mean()
print('Mean of Age is: ',Age_mean)
Age_std = data['Age'].std()
print('Standard Deviation of Age is: ',Age_std)
low= Age_mean - (3 * Age_std)
high= Age_mean + (3 * Age_std)
Age_outliers = data[(data['Age'] < low) | (data['Age'] > high)]
#print('Outliers of Age is:\n',Age_outliers)
print('Outliers of Age is:')
Age_outliers.head()
```

```
Skewness value of Age: 1.0113202630234552
Mean of Age is: 38.9218
Standard Deviation of Age is: 10.487806451704609
Outliers of Age is:
```

```
Out[151]:
```

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary
85	652	75	10	0.00	2	1	1	114675.75
158	646	73	6	97259.25	1	0	1	104719.66
230	673	72	1	0.00	2	0	1	111981.19
252	681	79	0	0.00	2	0	1	170968.99
310	652	80	4	0.00	2	1	1	188603.07

```
In [152]: print('Skewness value of CreditScore: ',data['CreditScore'].skew())
CreditScore_mean = data['CreditScore'].mean()
print('Mean of CreditScore is: ',CreditScore_mean)
CreditScore_std = data['CreditScore'].std()
print('Standard Deviation of CreditScore is: ',CreditScore_std)
low= CreditScore_mean - (3 * CreditScore_std)
high= CreditScore_mean + (3 * CreditScore_std)
CreditScore_outliers = data[(data['CreditScore'] < low) | (data['CreditScore'] > high)]
#print('Outliers of Age is:\n',Age_outliers)
print('Outliers of CreditScore is:')
```

```
CreditScore_outliers.head()
```

```
Skewness value of CreditScore: -0.07160660820092675
Mean of CreditScore is: 650.5288
Standard Deviation of CreditScore is: 96.65329873613035
Outliers of CreditScore is:
```

```
Out[152]:
```

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary
1405	359	44	6	128747.69	1	1	0	146955.71
1631	350	54	1	152677.48	1	1	1	191973.49
1838	350	39	0	109733.20	2	0	0	123602.11
1962	358	52	8	143542.36	3	1	0	141959.11
2473	351	57	4	163146.46	1	1	0	169621.69

```
In [153]: print('Skewness value of NumOfProducts: ',data['NumOfProducts'].skew())
NumOfProducts_mean = data['NumOfProducts'].mean()
print('Mean of NumOfProducts is: ',NumOfProducts_mean)
NumOfProducts_std = data['NumOfProducts'].std()
print('Standard Deviation of NumOfProducts is: ',NumOfProducts_std)
low= NumOfProducts_mean -(3 * NumOfProducts_std)
high= NumOfProducts_mean + (3 * NumOfProducts_std)
NumOfProducts_outliers = data[(data['NumOfProducts'] < low) | (data['NumOfProducts'] > high)]
#print('Outliers of Age is:\n',Age_outliers)
print('Outliers of NumOfProducts is:')
NumOfProducts_outliers.head()
```

```
Skewness value of NumOfProducts: 0.7455678882823168
Mean of NumOfProducts is: 1.5302
Standard Deviation of NumOfProducts is: 0.5816543579989906
Outliers of NumOfProducts is:
```

```
Out[153]:
```

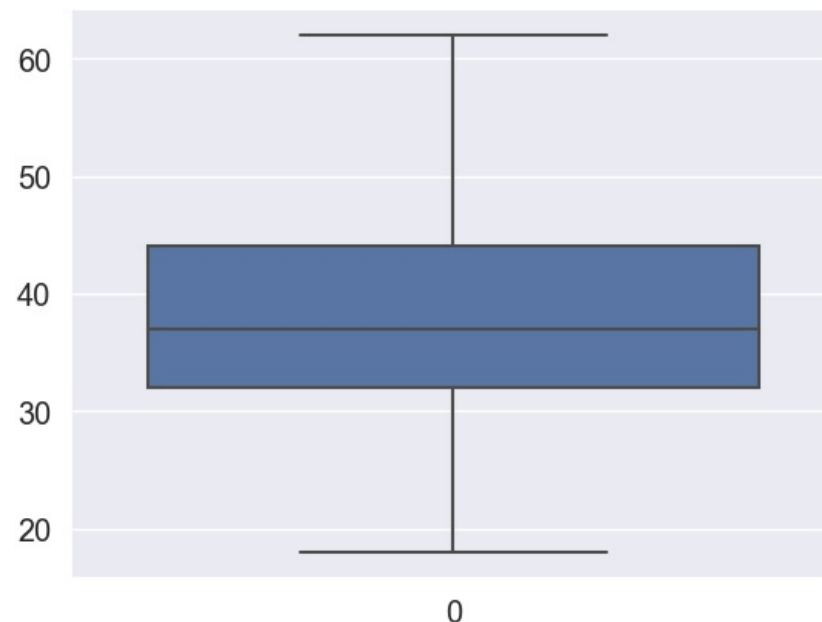
	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary
7	376	29	4	115046.74	4	1	0	119346.88
70	738	58	2	133745.44	4	1	0	28373.86
1254	628	46	1	46870.43	4	1	0	31272.14
1469	819	49	1	120656.86	4	0	0	166164.30
1488	596	30	6	121345.88	4	1	0	41921.75

## Outliers Eliminated

```
In [160]: Q1 = data['Age'].quantile(0.25)
Q3 = data['Age'].quantile(0.75)
IQR = Q3 - Q1
whisker_width = 1.5
lower_whisker = Q1 -(whisker_width*IQR)
upper_whisker = Q3 + (whisker_width*IQR)
data['Age']=np.where(data['Age']>upper_whisker,upper_whisker,np.where(data['Age']<lower_whisker,lower_whisker,d
print('Outliers Removed in Age:')
sns.boxplot(data['Age'])
```

```
Outliers Removed in Age:
```

```
Out[160]: <AxesSubplot: >
```



```
In [161]: Q1 = data['CreditScore'].quantile(0.25)
```

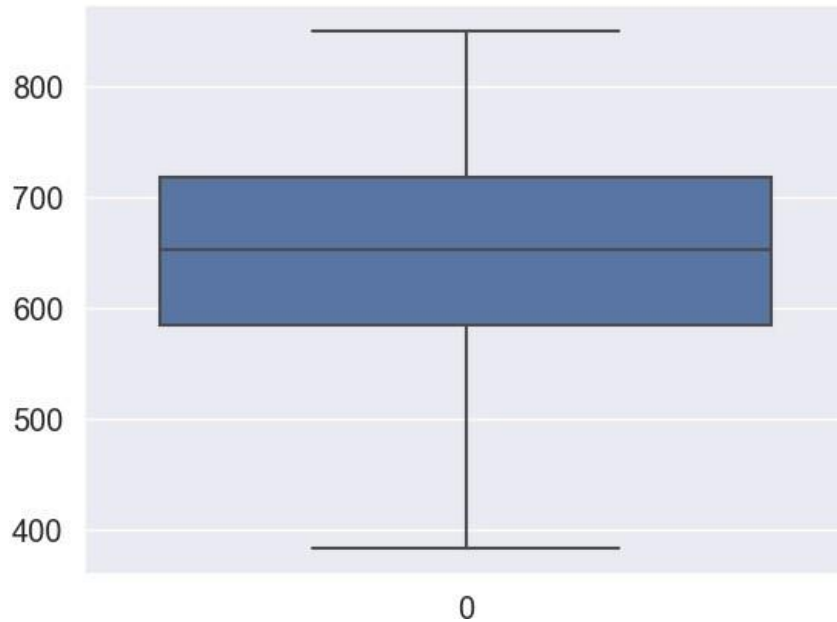
```

Q3 = data['CreditScore'].quantile(0.75)
IQR = Q3 - Q1
whisker_width = 1.5
lower_whisker = Q1 - (whisker_width*IQR)
upper_whisker = Q3 + (whisker_width*IQR)
data['CreditScore'] = np.where(data['CreditScore'] > upper_whisker, upper_whisker, np.where(data['CreditScore'] < lower_whisker, lower_whisker, data['CreditScore']))
print('Outliers Removed in CreditScore:')
sns.boxplot(data['CreditScore'])

```

Outliers Removed in CreditScore:

Out[161]: <AxesSubplot: >



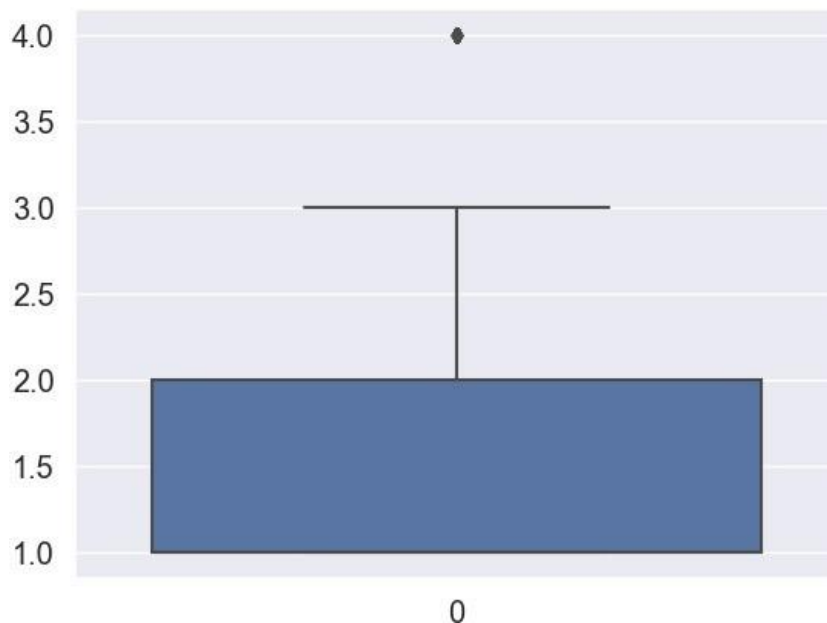
```

In [162]: Q1 = data['NumOfProducts'].quantile(0.25)
Q3 = data['NumOfProducts'].quantile(0.75)
IQR = Q3 - Q1
whisker_width = 1.5
lower_whisker = Q1 - (whisker_width*IQR)
upper_whisker = Q3 + (whisker_width*IQR)
data['Age'] = np.where(data['NumOfProducts'] > upper_whisker, upper_whisker, np.where(data['NumOfProducts'] < lower_whisker, lower_whisker, data['NumOfProducts']))
print('Outliers Removed in NumOfProducts:')
sns.boxplot(data['NumOfProducts'])

```

Outliers Removed in NumOfProducts:

Out[162]: <AxesSubplot: >



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

```

In [1]: #data1 = pd.read_csv('Churn_Modelling.csv')
#data1.head()

```

```

In [2]: import numpy as np #for numpy operations
import pandas as pd #for creating DataFrame using Pandas

```



```
# to split the dataset using sklearn
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
# load titanic dataset
data1 = pd.read_csv('Churn_Modelling.csv')
data1.head(10)
```

Out[2]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMemb
0	1	15634602	Hargrave	619	France	Female	42	2	0.00	1	1	
1	2	15647311	Hill	608	Spain	Female	41	1	83807.86	1	0	
2	3	15619304	Onio	502	France	Female	42	8	159660.80	3	1	
3	4	15701354	Boni	699	France	Female	39	1	0.00	2	0	
4	5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	1	
5	6	15574012	Chu	645	Spain	Male	44	8	113755.78	2	1	
6	7	15592531	Bartlett	822	France	Male	50	7	0.00	2	1	
7	8	15656148	Obinna	376	Germany	Female	29	4	115046.74	4	1	
8	9	15792365	He	501	France	Male	44	4	142051.07	2	0	
9	10	15592389	H?	684	France	Male	27	2	134603.88	1	1	

In [3]:

```
data1.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
```

In [4]:

```
# label_encoder object
label_encoder = LabelEncoder()
# Encode labels in column.
data1['Surname'] = label_encoder.fit_transform(data1['Surname'])
data1.head(10)
```

Out[4]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMemb
0	1	15634602	1115	619	France	Female	42	2	0.00	1	1	
1	2	15647311	1177	608	Spain	Female	41	1	83807.86	1	0	
2	3	15619304	2040	502	France	Female	42	8	159660.80	3	1	
3	4	15701354	289	699	France	Female	39	1	0.00	2	0	
4	5	15737888	1822	850	Spain	Female	43	2	125510.82	1	1	
5	6	15574012	537	645	Spain	Male	44	8	113755.78	2	1	
6	7	15592531	177	822	France	Male	50	7	0.00	2	1	
7	8	15656148	2000	376	Germany	Female	29	4	115046.74	4	1	
8	9	15792365	1146	501	France	Male	44	4	142051.07	2	0	
9	10	15592389	1081	684	France	Male	27	2	134603.88	1	1	

In [7]:

```
data1['Gender'] = label_encoder.fit_transform(data1['Gender'])
data1.head(10)
```

Out[7]:	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMemb
0	1	15634602	1115	619	France	0	42	2	0.00	1	1	
1	2	15647311	1177	608	Spain	0	41	1	83807.86	1	0	
2	3	15619304	2040	502	France	0	42	8	159660.80	3	1	
3	4	15701354	289	699	France	0	39	1	0.00	2	0	
4	5	15737888	1822	850	Spain	0	43	2	125510.82	1	1	
5	6	15574012	537	645	Spain	1	44	8	113755.78	2	1	
6	7	15592531	177	822	France	1	50	7	0.00	2	1	
7	8	15656148	2000	376	Germany	0	29	4	115046.74	4	1	
8	9	15792365	1146	501	France	1	44	4	142051.07	2	0	
9	10	15592389	1081	684	France	1	27	2	134603.88	1	1	

```
In [8]: data1['Geography']= label_encoder.fit_transform(data1['Geography'])
data1.head(10)
```

Out[8]:	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMemb
0	1	15634602	1115	619	0	0	42	2	0.00	1	1	
1	2	15647311	1177	608	2	0	41	1	83807.86	1	0	
2	3	15619304	2040	502	0	0	42	8	159660.80	3	1	
3	4	15701354	289	699	0	0	39	1	0.00	2	0	
4	5	15737888	1822	850	2	0	43	2	125510.82	1	1	
5	6	15574012	537	645	2	1	44	8	113755.78	2	1	
6	7	15592531	177	822	0	1	50	7	0.00	2	1	
7	8	15656148	2000	376	1	0	29	4	115046.74	4	1	
8	9	15792365	1146	501	0	1	44	4	142051.07	2	0	
9	10	15592389	1081	684	0	1	27	2	134603.88	1	1	

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

#

Dependent Variable : A dependent variable is a variable whose value depends on another variable.

#

Independent Variable : An Independent variable is a variable whose value never depends on another variable.

#

```
In [42]: print("The Minimum value of Dataset:\n",data1.min(numeric_only=True))
print("\n")
print("The Maximum value of Dataset:\n",data1.max(numeric_only=True))
print("\n")
print("The Mean value of Dataset:\n",data1.mean(numeric_only=True))
print("\n")

print(data1.count(0))
print(data1.shape)
print(data1.size)
```

```

The Minimum value of Dataset:
  RowNumber      1.00
  CustomerId    15565701.00
  Surname        0.00
  CreditScore    350.00
  Geography      0.00
  Gender         0.00
  Age            18.00
  Tenure         0.00
  Balance        0.00
  NumOfProducts  1.00
  HasCrCard      0.00
  IsActiveMember 0.00
  EstimatedSalary 11.58
  Exited         0.00
dtype: float64

```

```

The Maximum value of Dataset:
  RowNumber      10000.00
  CustomerId    15815690.00
  Surname        2931.00
  CreditScore    850.00
  Geography      2.00
  Gender         1.00
  Age            92.00
  Tenure         10.00
  Balance        250898.09
  NumOfProducts  4.00
  HasCrCard      1.00
  IsActiveMember 1.00
  EstimatedSalary 199992.48
  Exited         1.00
dtype: float64

```

```

The Mean value of Dataset:
  RowNumber      5.000500e+03
  CustomerId    1.569094e+07
  Surname        1.507774e+03
  CreditScore    6.505288e+02
  Geography      7.463000e-01
  Gender         5.457000e-01
  Age            3.892180e+01
  Tenure         5.012800e+00
  Balance        7.648589e+04
  NumOfProducts  1.530200e+00
  HasCrCard      7.055000e-01
  IsActiveMember 5.151000e-01
  EstimatedSalary 1.000902e+05
  Exited         2.037000e-01
dtype: float64

```

```

  RowNumber      10000
  CustomerId      10000
  Surname         10000
  CreditScore     10000
  Geography        10000
  Gender           10000
  Age              10000
  Tenure           10000
  Balance          10000
  NumOfProducts    10000
  HasCrCard        10000
  IsActiveMember    10000
  EstimatedSalary   10000
  Exited           10000
dtype: int64
(10000, 14)
140000

```

```

In [31]: y = data1["Surname"]
x=data1.drop(columns=["Surname"],axis=1)
x.head()

```

```

Out[31]:
   RowNumber  CustomerId  CreditScore  Geography  Gender  Age  Tenure  Balance  NumOfProducts  HasCrCard  IsActiveMember  EstimatedSalary
0          1      15634602          619         0      0   42         2      0.00              1          1              1              1
1          2      15647311          608         2      0   41         1     83807.86              1          0              1              1
2          3      15619304          502         0      0   42         8    159660.80              3          1              0              1
3          4      15701354          699         0      0   39         1         0.00              2          0              0              0
4          5      15737888          850         2      0   43         2    125510.82              1          1              1              1

```

## 9. Scale the independent variables

```
In [33]: names=x.columns
names
```

```
Out[33]: Index(['RowNumber', 'CustomerId', 'CreditScore', 'Geography', 'Gender', 'Age',
              'Tenure', 'Balance', 'NumOfProducts', 'HasCrCard', 'IsActiveMember',
              'EstimatedSalary', 'Exited'],
              dtype='object')
```

```
In [34]: from sklearn.preprocessing import scale
```

```
In [35]: X=scale(x)
```

```
In [36]: X
```

```
Out[36]: array([[ -1.73187761, -0.78321342, -0.32622142, ...,  0.97024255,
                0.02188649,  1.97716468],
               [ -1.7315312 , -0.60653412, -0.44003595, ...,  0.97024255,
                0.21653375, -0.50577476],
               [ -1.73118479, -0.99588476, -1.53679418, ..., -1.03067011,
                0.2406869 ,  1.97716468],
               ...,
               [  1.73118479, -1.47928179,  0.60498839, ...,  0.97024255,
               -1.00864308,  1.97716468],
               [  1.7315312 , -0.11935577,  1.25683526, ..., -1.03067011,
               -0.12523071,  1.97716468],
               [  1.73187761, -0.87055909,  1.46377078, ..., -1.03067011,
               -1.07636976, -0.50577476]])
```

```
In [37]: x = pd.DataFrame(X,columns = names )
x
```

```
Out[37]:
```

	RowNumber	CustomerId	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMem
0	-1.731878	-0.783213	-0.326221	-0.901886	-1.095988	0.293517	-1.041760	-1.225848	-0.911583	0.646092	0.970
1	-1.731531	-0.606534	-0.440036	1.515067	-1.095988	0.198164	-1.387538	0.117350	-0.911583	-1.547768	0.970
2	-1.731185	-0.995885	-1.536794	-0.901886	-1.095988	0.293517	1.032908	1.333053	2.527057	0.646092	-1.030
3	-1.730838	0.144767	0.501521	-0.901886	-1.095988	0.007457	-1.387538	-1.225848	0.807737	-1.547768	-1.030
4	-1.730492	0.652659	2.063884	1.515067	-1.095988	0.388871	-1.041760	0.785728	-0.911583	0.646092	0.970
...	...	...	...	...	...	...	...	...	...	...	...
9995	1.730492	-1.177652	1.246488	-0.901886	0.912419	0.007457	-0.004426	-1.225848	0.807737	0.646092	-1.030
9996	1.730838	-1.682806	-1.391939	-0.901886	0.912419	-0.373958	1.724464	-0.306379	-0.911583	0.646092	0.970
9997	1.731185	-1.479282	0.604988	-0.901886	-1.095988	-0.278604	0.687130	-1.225848	-0.911583	-1.547768	0.970
9998	1.731531	-0.119356	1.256835	0.306591	0.912419	0.293517	-0.695982	-0.022608	0.807737	0.646092	-1.030
9999	1.731878	-0.870559	1.463771	-0.901886	-1.095988	-1.041433	-0.350204	0.859965	-0.911583	0.646092	-1.030

10000 rows x 13 columns

## 10.Split the data into training and testing

#

The train-test split is used to estimate the performance of machine learning algorithms that are applicable for prediction-based Algorithms/Applications. By default, the Test set is split into 30 % of actual data and the training set is split into 70% of the actual data.

#

```
In [38]: from sklearn.model_selection import train_test_split
```

```
In [39]: x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=0)
```

```
In [40]: x_train.head()
```

Out[40]:

	RowNumber	CustomerId	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMem	
	7389	0.827747	-0.195066	0.170424	1.515067	-1.095988	-0.469311	-0.004426	-1.225848	0.807737	0.646092	-1.030
	9275	1.481077	0.810821	-2.312802	0.306591	0.912419	0.293517	-1.387538	-0.012892	-0.911583	0.646092	0.970
	2995	-0.694379	-1.507642	-1.195351	-0.901886	-1.095988	-0.946079	-1.041760	0.575076	-0.911583	0.646092	-1.030
	5316	0.109639	1.243462	0.035916	1.515067	0.912419	0.102810	-0.004426	0.467955	-0.911583	0.646092	-1.030
	356	-1.608556	-1.100775	2.063884	1.515067	-1.095988	1.723821	1.032908	0.806010	0.807737	0.646092	0.970

In [41]:

x\_train.shape,y\_train.shape,x\_test.shape,y\_test.shape

Out[41]:

((8000, 13), (8000,), (2000, 13), (2000,))

Loading [MathJax]/extensions/Safe.js