# Assignment -2

Data Visualization and Pre-Processing

| | |
|---|---|
| Assignment Date | 26 September 2022 |
| Student Name | G.Lakshmi Priya |
| Student Roll Number | 9517201906023 |
| Maximum Marks | 2 Marks |

**Question 1 - Load the dataset.**

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from google.colab import files

file = files.upload()
```

**OUTPUT:**


**Question 2 - Perform  Univariate, Bivariate and Multivariate Analysis**

SOLUTION:

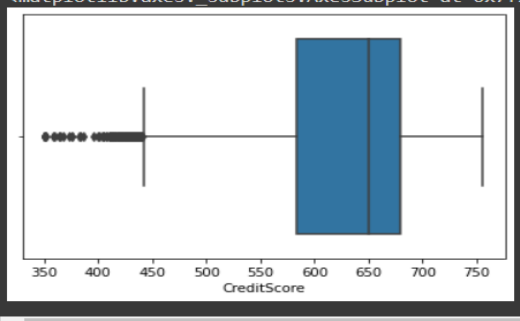
```
sns.boxplot(df['CreditScore'])
sns.boxplot(df['Age'])
sns.boxplot(df['Tenure'])
sns.boxplot(df['Balance'])
sns.boxplot(df['EstimatedSalary'])
```

sns.heatmap(df.corr(), annot=True)

OUTPUT:

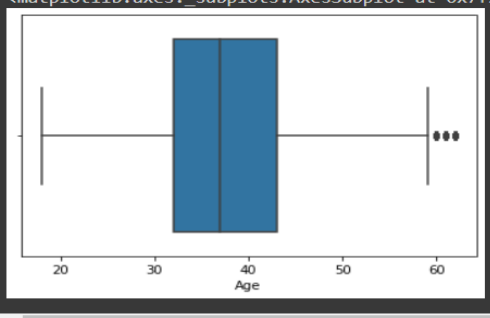
```
sns.boxplot(df['CreditScore'])
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureW
  FutureWarning
<matplotlib.axes._subplots.AxesSubplot at 0x7f2331dcdfd0>
```



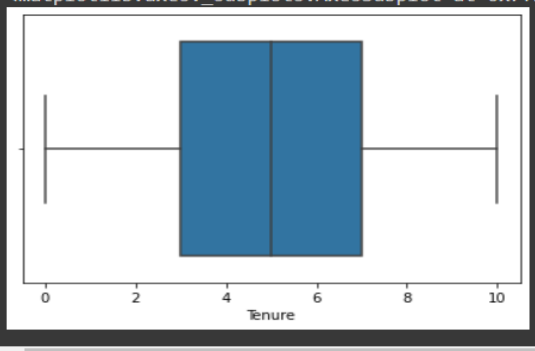

```
sns.boxplot(df['Age'])
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarnin
  FutureWarning
<matplotlib.axes._subplots.AxesSubplot at 0x7f2331c8f810>
```



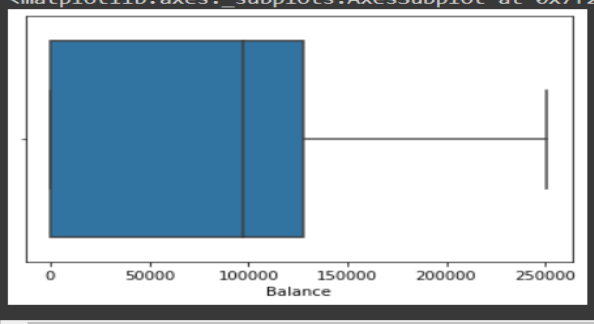

```
sns.boxplot(df['Tenure'])
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: Futu
  FutureWarning
<matplotlib.axes._subplots.AxesSubplot at 0x7f23318216d0>
```



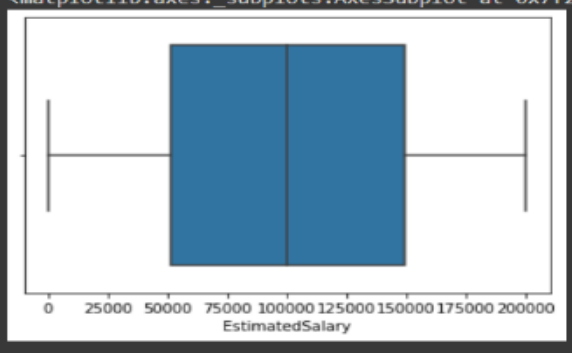
```
sns.boxplot(df['Balance'])
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_dec
  FutureWarning
<matplotlib.axes._subplots.AxesSubplot at 0x7f23318
```



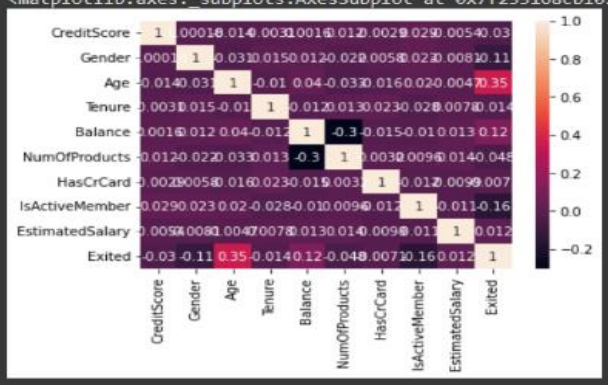

```
sns.boxplot(df['EstimatedSalary'])
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorat
  FutureWarning
<matplotlib.axes._subplots.AxesSubplot at 0x7f23317086d
```




```
sns.heatmap(df.corr(), annot=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f233168cb10>
```




**Question 3 - Perform descriptive statistics on the dataset.**

SOLUTION:

```
df.describe()
```

OUTPUT:

Descriptive statistics of the dataset

```
[18] df.describe()
```

| | CreditScore | Gender | Age | Tenure | Balance | NumOfProducts | HasCrCard | IsActiveMember | EstimatedSalary | Exited |
|---|---|---|---|---|---|---|---|---|---|---|
| count | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.00000 | 10000.000000 | 10000.000000 | 10000.000000 |
| mean | 650.528800 | 0.545700 | 38.921800 | 5.012800 | 76485.889288 | 1.530200 | 0.70550 | 0.515100 | 100090.239881 | 0.203700 |
| std | 96.653299 | 0.497932 | 10.487806 | 2.892174 | 62397.405202 | 0.581654 | 0.45584 | 0.499797 | 57510.492818 | 0.402769 |
| min | 350.000000 | 0.000000 | 18.000000 | 0.000000 | 0.000000 | 1.000000 | 0.00000 | 0.000000 | 11.580000 | 0.000000 |
| 25% | 584.000000 | 0.000000 | 32.000000 | 3.000000 | 0.000000 | 1.000000 | 0.00000 | 0.000000 | 51002.110000 | 0.000000 |
| 50% | 652.000000 | 1.000000 | 37.000000 | 5.000000 | 97198.540000 | 1.000000 | 1.00000 | 1.000000 | 100193.915000 | 0.000000 |
| 75% | 718.000000 | 1.000000 | 44.000000 | 7.000000 | 127644.240000 | 2.000000 | 1.00000 | 1.000000 | 149388.247500 | 0.000000 |
| max | 850.000000 | 1.000000 | 92.000000 | 10.000000 | 250898.090000 | 4.000000 | 1.00000 | 1.000000 | 199992.480000 | 1.000000 |

**Question 4 – Handle the missing values**

SOLUTION:

```
df.duplicated().sum()

df.nunique()

df.info()
```

OUTPUT:

Handling missing values

```
[20] df.duplicated().sum()
0

df.isna().sum()

CreditScore        0
Gender             0
Age                0
Tenure             0
Balance            0
NumOfProducts      0
HasCrCard          0
IsActiveMember     0
EstimatedSalary    0
Exited             0
dtype: int64

[22] df.nunique()

CreditScore         460
Gender                2
Age                  70
Tenure               11
Balance            6382
NumOfProducts         4
HasCrCard             2
IsActiveMember        2
EstimatedSalary    9999
Exited                2
dtype: int64
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   CreditScore      10000 non-null  int64
 1   Gender           10000 non-null  int64
 2   Age              10000 non-null  int64
 3   Tenure           10000 non-null  int64
 4   Balance          10000 non-null  float64
 5   NumOfProducts    10000 non-null  int64
 6   HasCrCard        10000 non-null  int64
 7   IsActiveMember   10000 non-null  int64
 8   EstimatedSalary  10000 non-null  float64
 9   Exited           10000 non-null  int64
dtypes: float64(2), int64(8)
memory usage: 781.4 KB
```

**Question 5 - Find the outliers and replace the outliers**

**SOLUTION:**

out = df.drop(columns=['Gender', 'Tenure', 'HasCrCard', 'IsActiveMember', 'NumOfProducts', 'Exited']).quantile(q=[0.25, 0.50])

OUTPUT:

Handling outliers

```
[24] out = df.drop(columns=['Gender', 'Tenure', 'HasCrCard', 'IsActiveMember', 'NumOfProducts', 'Exited']).quantile(q=[0.25, 0.50])
     out
```

| | CreditScore | Age | Balance | EstimatedSalary |
|---|---|---|---|---|
| 0.25 | 584.0 | 32.0 | 0.00 | 51002.110 |
| 0.50 | 652.0 | 37.0 | 97198.54 | 100193.915 |

```
[25] Q1 = out.iloc[0]
     Q3 = out.iloc[1]
     iqr = Q3 - Q1
     iqr
```

```
CreditScore          68.000
Age                   5.000
Balance           97198.540
EstimatedSalary   49191.805
dtype: float64
```

```
upper = out.iloc[1] + 1.5*iqr
upper
```

```
CreditScore          754.0000
Age                   44.5000
Balance           242996.3500
EstimatedSalary   173981.6225
dtype: float64
```

```
[27] lower = out.iloc[0] - 1.5*iqr
     lower
```

```
CreditScore           482.0000
Age                    24.5000
Balance             -145797.8100
EstimatedSalary      -22785.5975
dtype: float64
```

Replace outliers

**Replace outliers**

SOLUTION:

```
df['CreditScore'] = np.where(df['CreditScore']>756, 650.5288, df['CreditScore'])
df['Age'] = np.where(df['Age']>62, 38.9218, df['Age'])
```

**Question 6 - Check for Categorical columns and perform encoding.**

SOLUTION:

```
df['Gender'].replace({'Male': 1, 'Female': 0}, inplace=True)

df.head(10)
```

OUTPUT:

Check for categorical columns and perform encoding

```
[19] df['Gender'].replace({'Male': 1, 'Female': 0}, inplace=True)
     df.head(10)
```

| | CreditScore | Gender | Age | Tenure | Balance | NumOfProducts | HasCrCard | IsActiveMember | EstimatedSalary | Exited |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 619 | 0 | 42 | 2 | 0.00 | 1 | 1 | 1 | 101348.88 | 1 |
| 1 | 608 | 0 | 41 | 1 | 83807.86 | 1 | 0 | 1 | 112542.58 | 0 |
| 2 | 502 | 0 | 42 | 8 | 159660.80 | 3 | 1 | 0 | 113931.57 | 1 |
| 3 | 699 | 0 | 39 | 1 | 0.00 | 2 | 0 | 0 | 93826.63 | 0 |
| 4 | 850 | 0 | 43 | 2 | 125510.82 | 1 | 1 | 1 | 79084.10 | 0 |
| 5 | 645 | 1 | 44 | 8 | 113755.78 | 2 | 1 | 0 | 149756.71 | 1 |
| 6 | 822 | 1 | 50 | 7 | 0.00 | 2 | 1 | 1 | 10062.80 | 0 |
| 7 | 376 | 0 | 29 | 4 | 115046.74 | 4 | 1 | 0 | 119346.88 | 1 |
| 8 | 501 | 1 | 44 | 4 | 142051.07 | 2 | 0 | 1 | 74940.50 | 0 |
| 9 | 684 | 1 | 27 | 2 | 134603.88 | 1 | 1 | 1 | 71725.73 | 0 |

**Question 7 – Split the data into dependent and independent variables.**

SOLUTION:

```
df = df.drop(columns=['RowNumber', 'CustomerId', 'Surname', 'Geography'])
```

df.head()

OUTPUT:

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

| | CreditScore | Gender | Age | Tenure | Balance | NumOfProducts | HasCrCard | IsActiveMember | EstimatedSalary |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 619.0000 | 0 | 42.0 | 2 | 0.00 | 1 | 1 | 1 | 101348.88 |
| 1 | 608.0000 | 0 | 41.0 | 1 | 83807.86 | 1 | 0 | 1 | 112542.58 |
| 2 | 502.0000 | 0 | 42.0 | 8 | 159660.80 | 3 | 1 | 0 | 113931.57 |
| 3 | 699.0000 | 0 | 39.0 | 1 | 0.00 | 2 | 0 | 0 | 93826.63 |
| 4 | 650.5288 | 0 | 43.0 | 2 | 125510.82 | 1 | 1 | 1 | 79084.10 |

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

0    1
1    0
2    1
3    0
4    0
Name: Exited, dtype: int64
```

**Question 8 – Scale the independent variables**

SOLUTION:

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

OUTPUT:

Scale the Independent variables

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

```
array([[-0.13284832, -1.09598752,  0.48205148, ...,  0.64609167,
         0.97024255,  0.02188649],
       [-0.28182929, -1.09598752,  0.36638802, ..., -1.54776799,
         0.97024255,  0.21653375],
       [-1.71746409, -1.09598752,  0.48205148, ...,  0.64609167,
        -1.03067011,  0.2406869 ],
       ...,
       [ 1.08608688, -1.09598752, -0.21192932, ..., -1.54776799,
         0.97024255, -1.00864308],
       [ 0.29416906,  0.91241915,  0.48205148, ...,  0.64609167,
        -1.03067011, -0.12523071],
       [ 0.29416906, -1.09598752, -1.13723705, ...,  0.64609167,
        -1.03067011, -1.07636976]])
```

**Question 9 - Split the data into training and testing**

SOLUTION:

```
from sklearn.model_selection import train_test_split

x_train, y_test = train_test_split(x, y, test_size=0.2, random_state=0)

print(x_train.shape)

print(x_test.shape)

print(y_train.shape)    print(y_test.shape)   OUTPUT:
```

Split into Training and Testing data

```
[34] 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)
```

```
[35] print(x_train.shape)
     print(x_test.shape)
     print(y_train.shape)
     print(y_test.shape)

     (8000, 9)
     (2000, 9)
     (8000,)
     (2000,)
```