

## Assignment -2

### Data Visualization and Pre-processing

|                     |                   |
|---------------------|-------------------|
| Assignment Date     | 24 September 2022 |
| Student Name        | SANJANA S         |
| Student Roll Number | 311519104050      |
| Maximum Marks       | 2 Marks           |

**To Perform Below Tasks to complete the assignment:-**

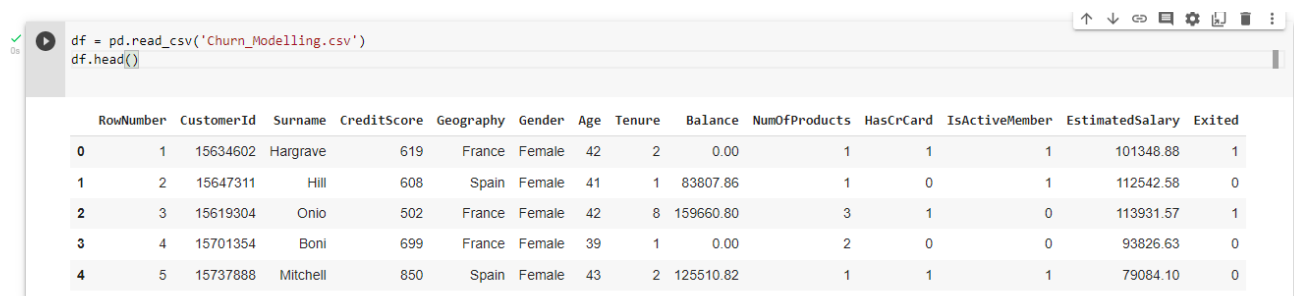
Step 1. Download the dataset: [Dataset](#)

Step 2. Load the dataset.

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

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

Output :



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

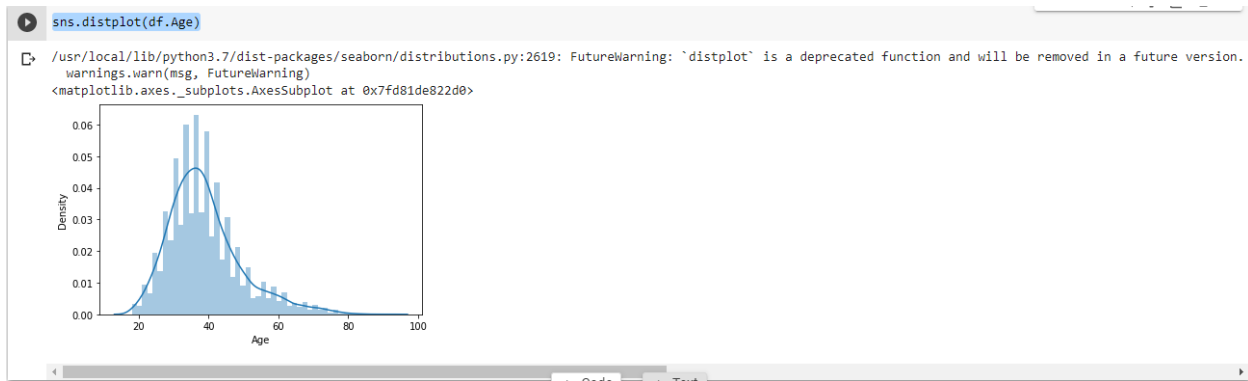
|   | RowNumber | CustomerId | Surname  | CreditScore | Geography | Gender | Age | Tenure | Balance   | NumOfProducts | HasCrCard | IsActiveMember | EstimatedSalary | Exited |
|---|-----------|------------|----------|-------------|-----------|--------|-----|--------|-----------|---------------|-----------|----------------|-----------------|--------|
| 0 | 1         | 15634602   | Hargrave | 619         | France    | Female | 42  | 2      | 0.00      | 1             | 1         | 1              | 101348.88       | 1      |
| 1 | 2         | 15647311   | Hill     | 608         | Spain     | Female | 41  | 1      | 83807.86  | 1             | 0         | 1              | 112542.58       | 0      |
| 2 | 3         | 15619304   | Onio     | 502         | France    | Female | 42  | 8      | 159660.80 | 3             | 1         | 0              | 113931.57       | 1      |
| 3 | 4         | 15701354   | Boni     | 699         | France    | Female | 39  | 1      | 0.00      | 2             | 0         | 0              | 93826.63        | 0      |
| 4 | 5         | 15737888   | Mitchell | 850         | Spain     | Female | 43  | 2      | 125510.82 | 1             | 1         | 1              | 79084.10        | 0      |

Step 3. Perform Below Visualizations.

- Univariate Analysis

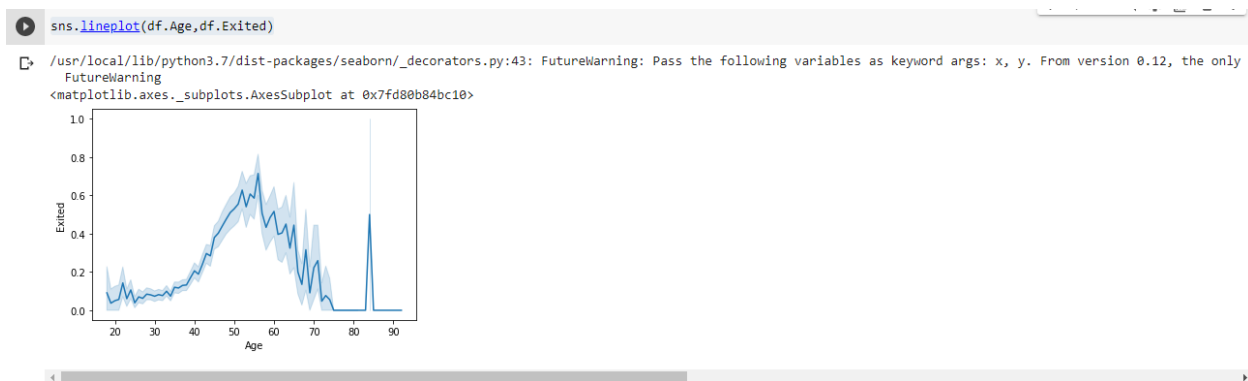
```
sns.distplot(df.Age)
```

Output :



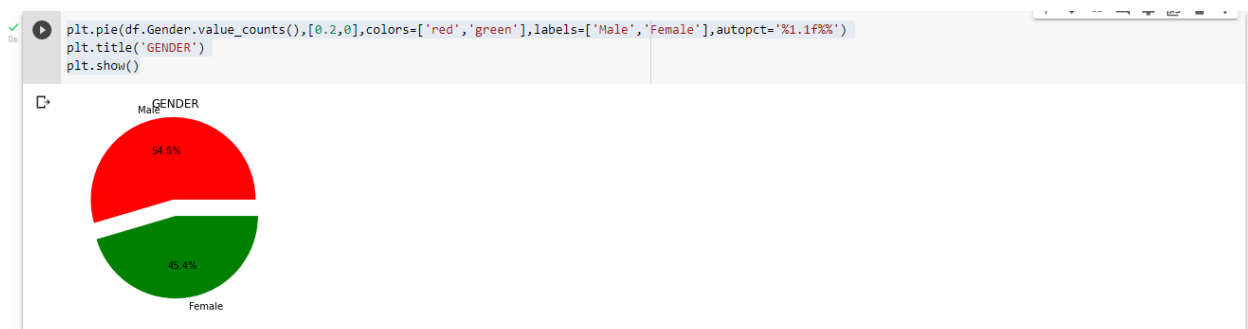
```
sns.lineplot(df.Age,df.Exited)
```

Output :



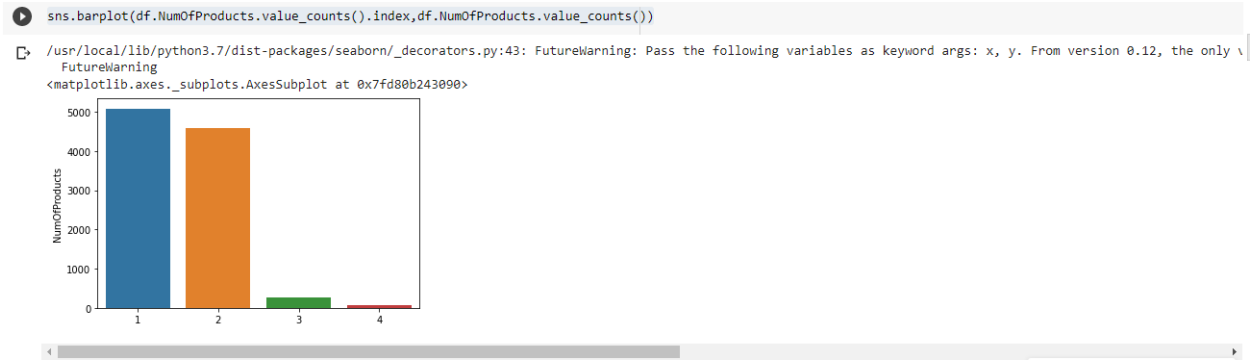
```
plt.pie(df.Gender.value_counts(),[0.2,0],colors=['red','green'],labels=['Male','Female'],autopct='%1.1f%%')  
plt.title('GENDER')  
plt.show()
```

Output :



```
sns.barplot(df.NumOfProducts.value_counts().index,df.NumOfProducts.value_counts())
```

Output :



## ● Bi - Variate Analysis

```
def countplot_2(x,hue,title=None,figsize=(6,5)):
    plt.figure(figsize=figsize)
    sns.countplot(data=df[[x,hue]],x=x,hue=hue)
    plt.title(title)
    plt.show()
```

countplot\_2('IsActiveMember','NumOfProducts','Credit Card Holders Product Details')

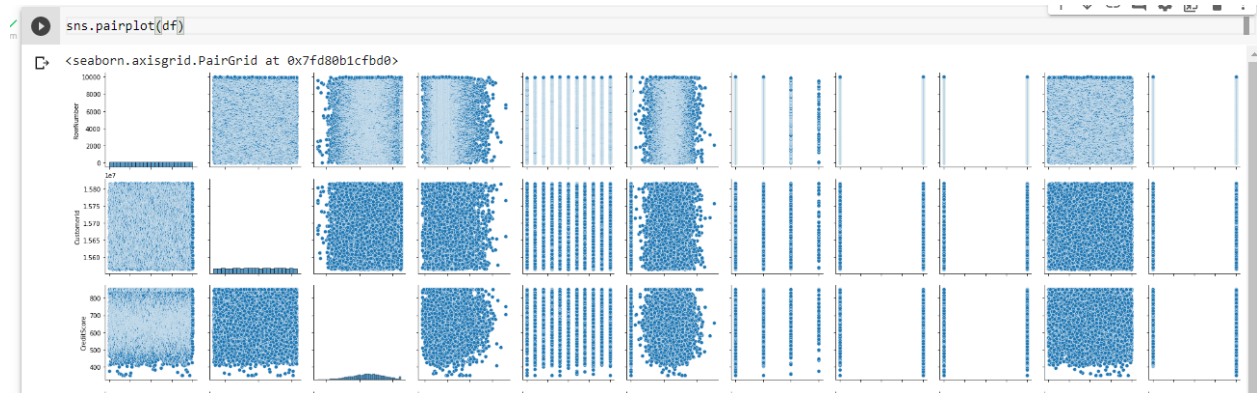
Output :



## ● Multi - Variate Analysis

```
sns.pairplot(df)
```

Output :



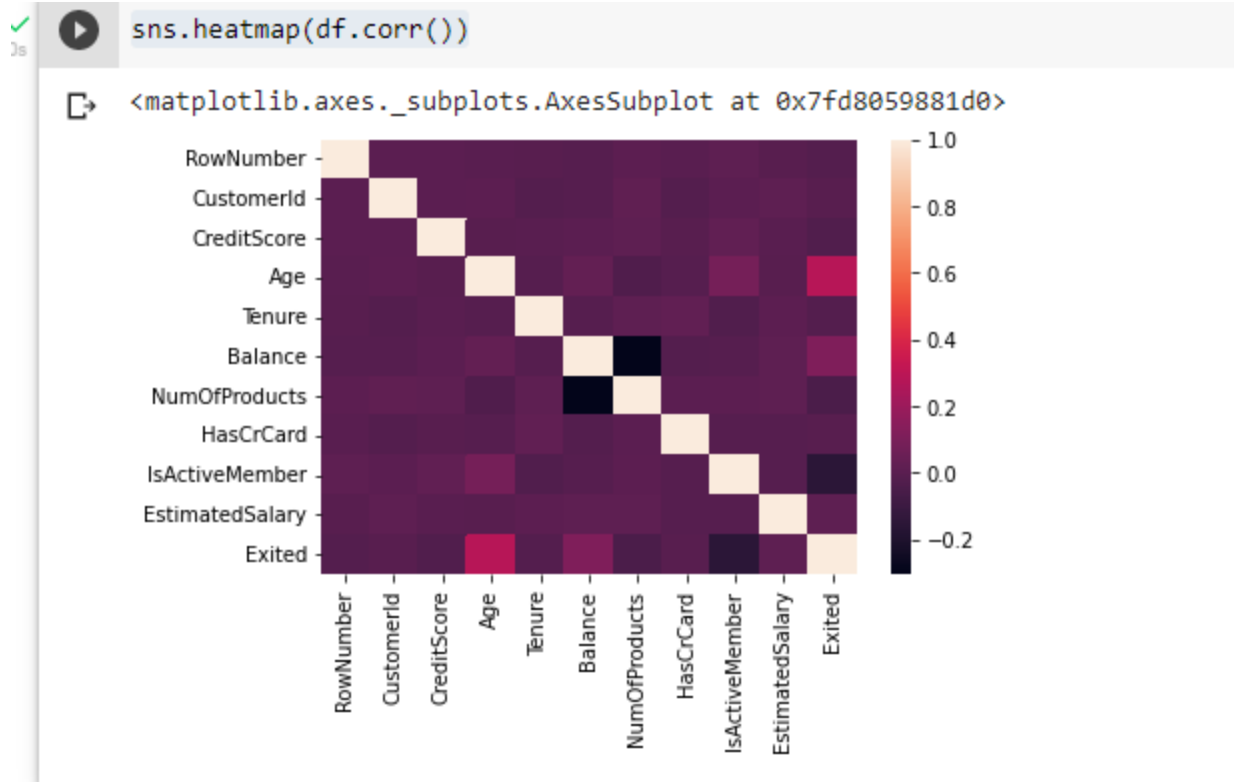
```
df.corr()
```

Output :

|                 | RowNumber | CustomerId | CreditScore | Age       | Tenure    | Balance   | NumOfProducts | HasCrCard | IsActiveMember | EstimatedSalary | Exited    |
|-----------------|-----------|------------|-------------|-----------|-----------|-----------|---------------|-----------|----------------|-----------------|-----------|
| RowNumber       | 1.000000  | 0.004202   | 0.005840    | 0.000783  | -0.006495 | -0.009067 | 0.007246      | 0.000599  | 0.012044       | -0.005988       | -0.016571 |
| CustomerId      | 0.004202  | 1.000000   | 0.005308    | 0.009497  | -0.014883 | -0.012419 | 0.016972      | -0.014025 | 0.001665       | 0.015271        | -0.006248 |
| CreditScore     | 0.005840  | 0.005308   | 1.000000    | -0.003965 | 0.000842  | 0.006268  | 0.012238      | -0.005458 | 0.025651       | -0.001384       | -0.027094 |
| Age             | 0.000783  | 0.009497   | -0.003965   | 1.000000  | -0.009997 | 0.028308  | -0.030680     | -0.011721 | 0.085472       | -0.007201       | 0.285323  |
| Tenure          | -0.006495 | -0.014883  | 0.000842    | -0.009997 | 1.000000  | -0.012254 | 0.013444      | 0.022583  | -0.028362      | 0.007784        | -0.014001 |
| Balance         | -0.009067 | -0.012419  | 0.006268    | 0.028308  | -0.012254 | 1.000000  | -0.304180     | -0.014858 | -0.010084      | 0.012797        | 0.118533  |
| NumOfProducts   | 0.007246  | 0.016972   | 0.012238    | -0.030680 | 0.013444  | -0.304180 | 1.000000      | 0.003183  | 0.009612       | 0.014204        | -0.047820 |
| HasCrCard       | 0.000599  | -0.014025  | -0.005458   | -0.011721 | 0.022583  | -0.014858 | 0.003183      | 1.000000  | -0.011866      | -0.009933       | -0.007138 |
| IsActiveMember  | 0.012044  | 0.001665   | 0.025651    | 0.085472  | -0.028362 | -0.010084 | 0.009612      | -0.011866 | 1.000000       | -0.011421       | -0.156128 |
| EstimatedSalary | -0.005988 | 0.015271   | -0.001384   | -0.007201 | 0.007784  | 0.012797  | 0.014204      | -0.009933 | -0.011421      | 1.000000        | 0.012097  |
| Exited          | -0.016571 | -0.006248  | -0.027094   | 0.285323  | -0.014001 | 0.118533  | -0.047820     | -0.007138 | -0.156128      | 0.012097        | 1.000000  |

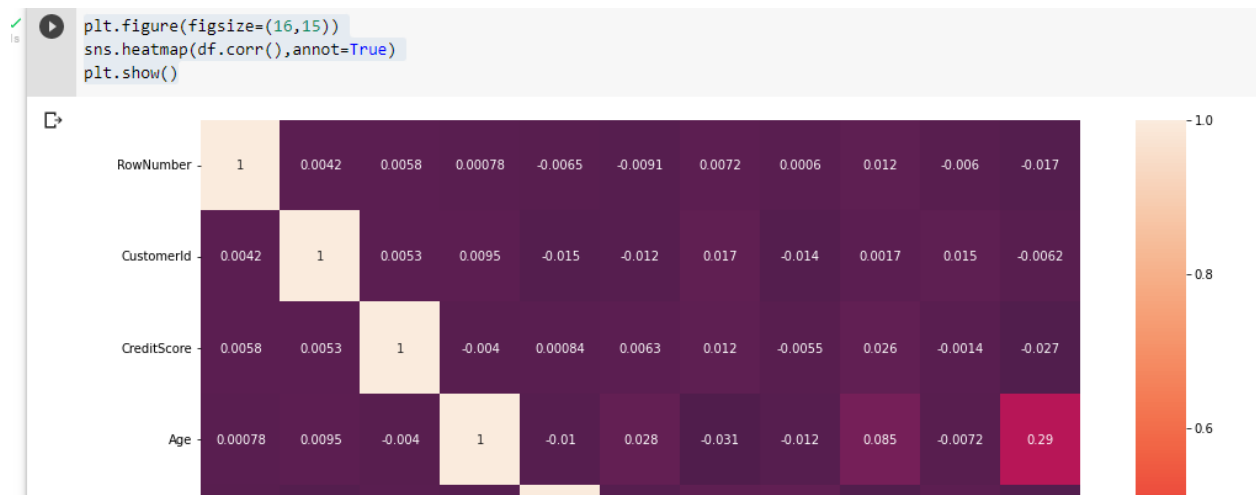
```
sns.heatmap(df.corr())
```

Output :



```
plt.figure(figsize=(16,15))
sns.heatmap(df.corr(),annot=True)
plt.show()
```

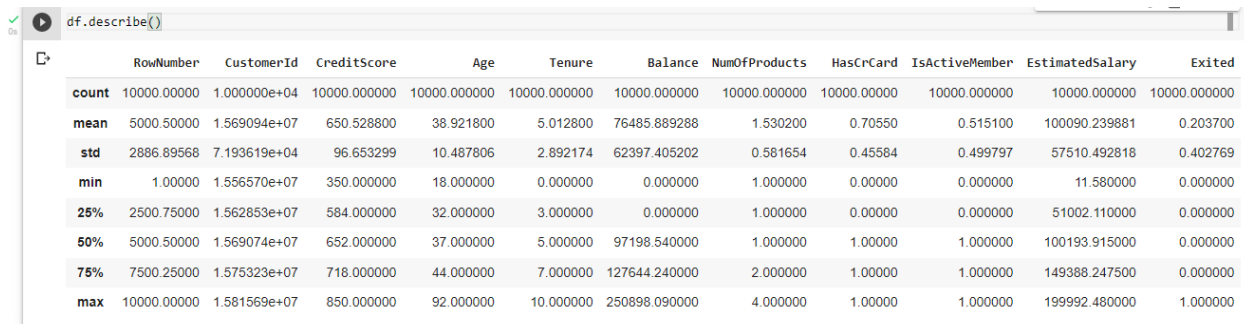
Output :



Step 4. Perform descriptive statistics on the dataset.

```
df.describe()
```

Output :

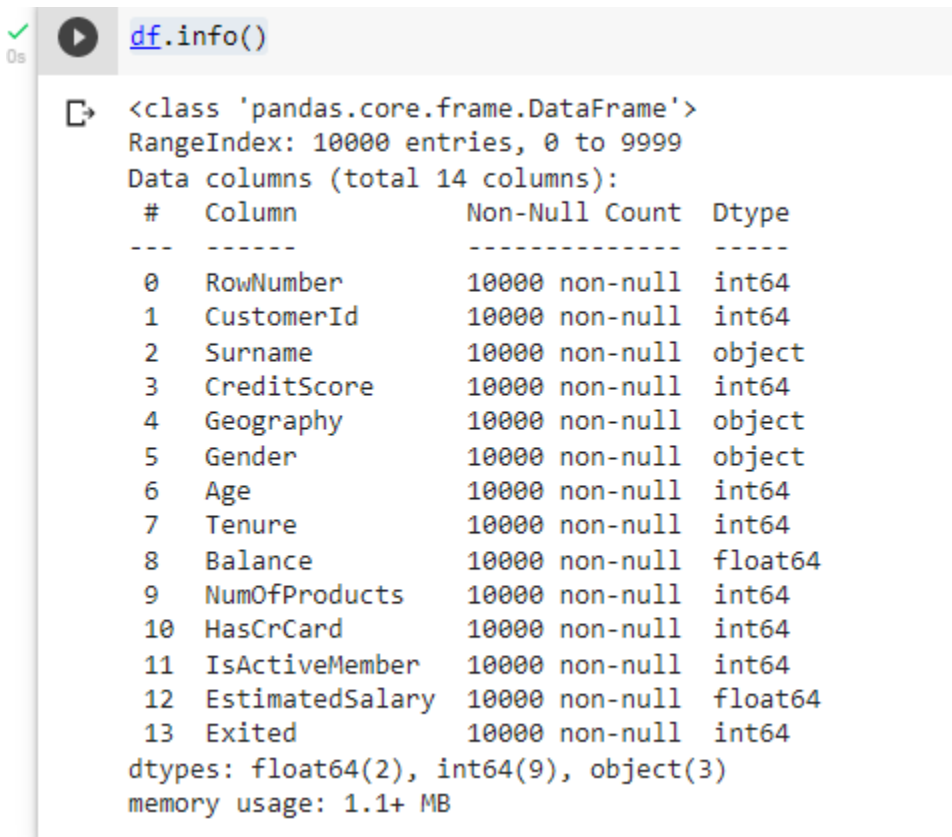


A Jupyter Notebook cell with the code `df.describe()` executed. The output is a summary statistics table for a DataFrame with 12 columns: RowNumber, CustomerId, CreditScore, Age, Tenure, Balance, NumOfProducts, HasCrCard, IsActiveMember, EstimatedSalary, and Exited. The table includes rows for count, mean, std, min, 25%, 50%, 75%, and max.

|       | RowNumber   | CustomerId   | CreditScore  | Age          | Tenure       | Balance       | NumOfProducts | HasCrCard   | IsActiveMember | EstimatedSalary | Exited       |
|-------|-------------|--------------|--------------|--------------|--------------|---------------|---------------|-------------|----------------|-----------------|--------------|
| count | 10000.00000 | 1.000000e+04 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000  | 10000.000000  | 10000.00000 | 10000.000000   | 10000.000000    | 10000.000000 |
| mean  | 5000.50000  | 1.569094e+07 | 650.528800   | 38.921800    | 5.012800     | 76485.889288  | 1.530200      | 0.70550     | 0.515100       | 100090.239881   | 0.203700     |
| std   | 2886.89568  | 7.193619e+04 | 96.653299    | 10.487806    | 2.892174     | 62397.405202  | 0.581654      | 0.45584     | 0.499797       | 57510.492818    | 0.402769     |
| min   | 1.00000     | 1.556570e+07 | 350.000000   | 18.000000    | 0.000000     | 0.000000      | 1.000000      | 0.00000     | 0.000000       | 11.580000       | 0.000000     |
| 25%   | 2500.75000  | 1.562853e+07 | 584.000000   | 32.000000    | 3.000000     | 0.000000      | 1.000000      | 0.00000     | 0.000000       | 51002.110000    | 0.000000     |
| 50%   | 5000.50000  | 1.569074e+07 | 652.000000   | 37.000000    | 5.000000     | 97198.540000  | 1.000000      | 1.00000     | 1.000000       | 100193.915000   | 0.000000     |
| 75%   | 7500.25000  | 1.575323e+07 | 718.000000   | 44.000000    | 7.000000     | 127644.240000 | 2.000000      | 1.00000     | 1.000000       | 149388.247500   | 0.000000     |
| max   | 10000.00000 | 1.581569e+07 | 850.000000   | 92.000000    | 10.000000    | 250898.090000 | 4.000000      | 1.00000     | 1.000000       | 199992.480000   | 1.000000     |

df.info()

Output :



A Jupyter Notebook cell with the code `df.info()` executed. The output shows the DataFrame's structure, including the number of entries (10000), the number of columns (14), and the data types of each column. It also shows the memory usage (1.1+ MB).

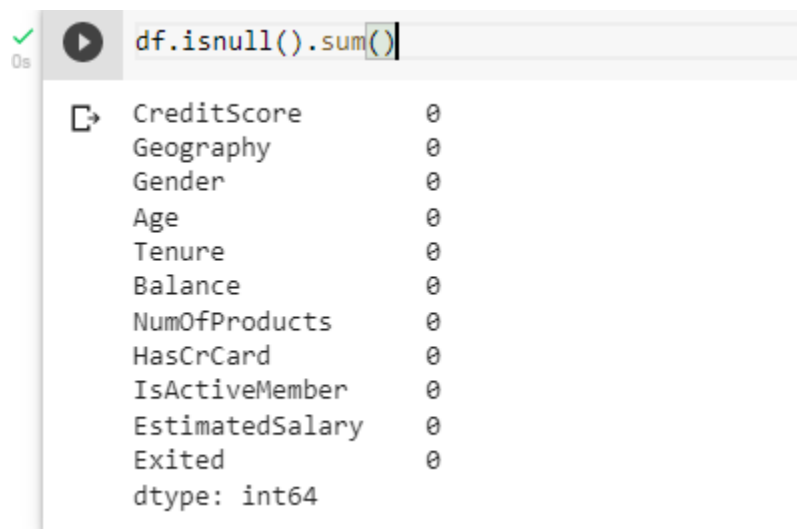
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 14 columns):
 #   Column                Non-Null Count  Dtype
---  ---
 0   RowNumber              10000 non-null  int64
 1   CustomerId             10000 non-null  int64
 2   Surname                10000 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
```

Step 5. Handle the Missing values.

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

```
df.isnull().sum()
```

Output :

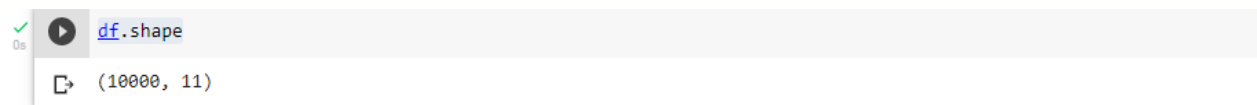


A screenshot of a Jupyter Notebook cell showing the command `df.isnull().sum()` and its output. The output is a table with 11 rows, each representing a column and its corresponding count of null values. All counts are 0. The data type is listed as `dtype: int64`.

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

```
df.shape
```

Output :



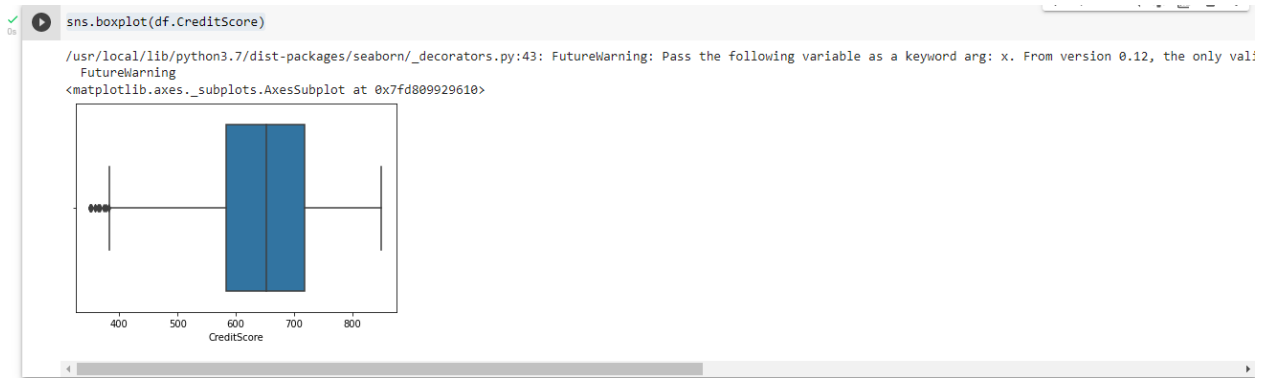
A screenshot of a Jupyter Notebook cell showing the command `df.shape` and its output. The output is a tuple `(10000, 11)`, indicating 10,000 rows and 11 columns.

```
(10000, 11)
```

Step 6. Find the outliers and replace the outliers

```
sns.boxplot(df.CreditScore)
```

Output :



```
Q1 = df.CreditScore.quantile(0.25)
```

```
Q3 = df.CreditScore.quantile(0.75)
```

```
IQR = Q3-Q1
```

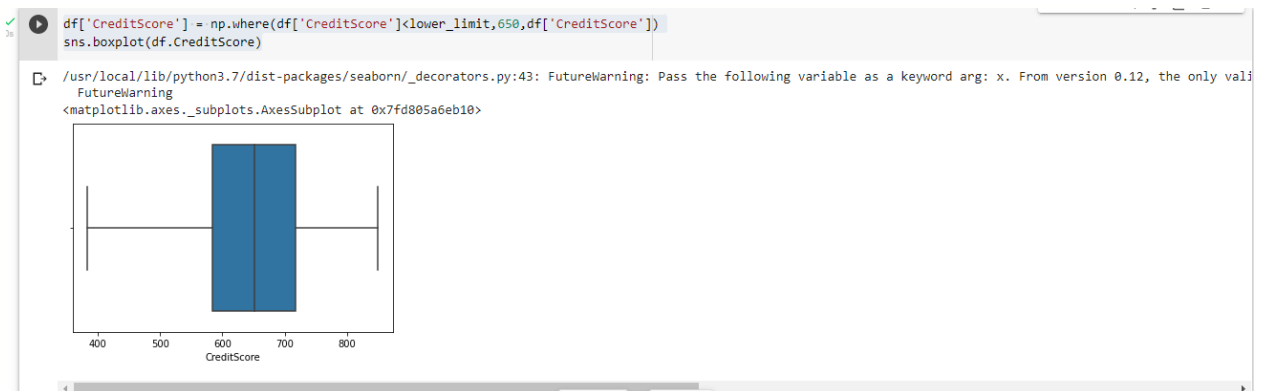
```
upper_limit = Q3 + (1.5*IQR)
```

```
lower_limit = Q1 - (1.5*IQR)
```

```
df['CreditScore'] = np.where(df['CreditScore']<lower_limit,650,df['CreditScore'])
```

```
sns.boxplot(df.CreditScore)
```

Output :



Step 7. Check for Categorical columns and perform encoding.

```
from sklearn.preprocessing import LabelEncoder
```

```
le = LabelEncoder()
```

```
df.Geography = le.fit_transform(df.Geography)
```

```
df.Gender = le.fit_transform(df.Gender)
```

```
df.head()
```

Output :



df.head()

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

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

```
X = df.drop(columns=['Exited'])
X.head()
```

Output :

X = df.drop(columns=['Exited'])  
X.head()

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

```
Y = df.Exited
Y.head()
```

Output :

Y = df.Exited  
Y.head()

|   |   |
|---|---|
| 0 | 1 |
| 1 | 0 |
| 2 | 1 |
| 3 | 0 |
| 4 | 0 |

Name: Exited, dtype: int64

Step 9. Scale the independent variables

```
from sklearn.preprocessing import MinMaxScaler
scale = MinMaxScaler()
X_scaled = pd.DataFrame(scale.fit_transform(X), columns=X.columns)
```

Step 10. Split the data into training and testing

```
from sklearn.model_selection import train_test_split
x_train , y_train , x_test , y_test = train_test_split(X_scaled,Y,test_size=0.2,random_state=0)
```

Output :

✓  
0s



`X_scaled.shape`

`(10000, 10)`

✓  
0s

`[40] x_train.shape`

`(8000, 10)`