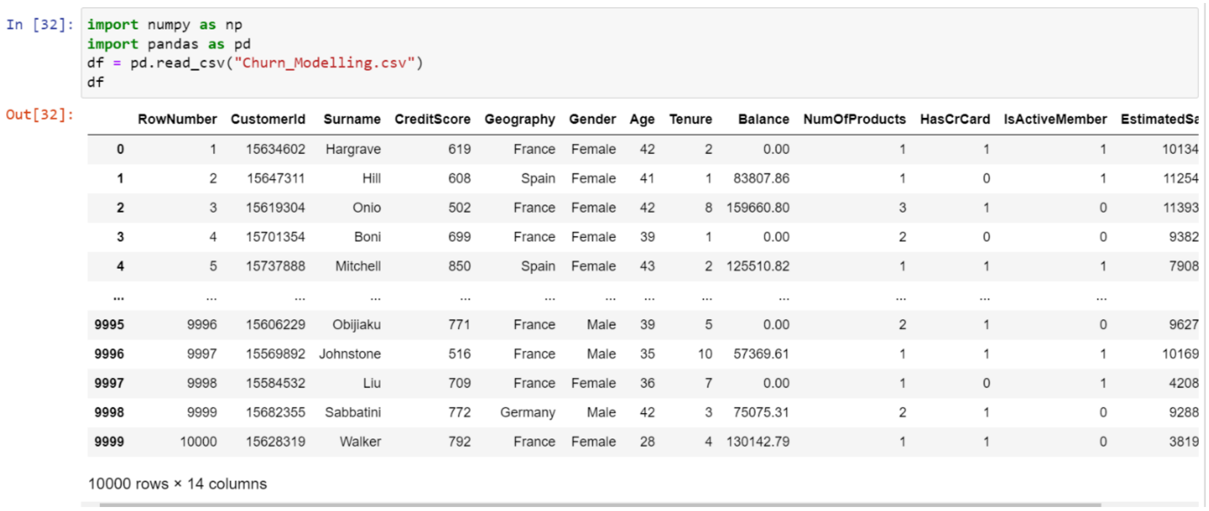
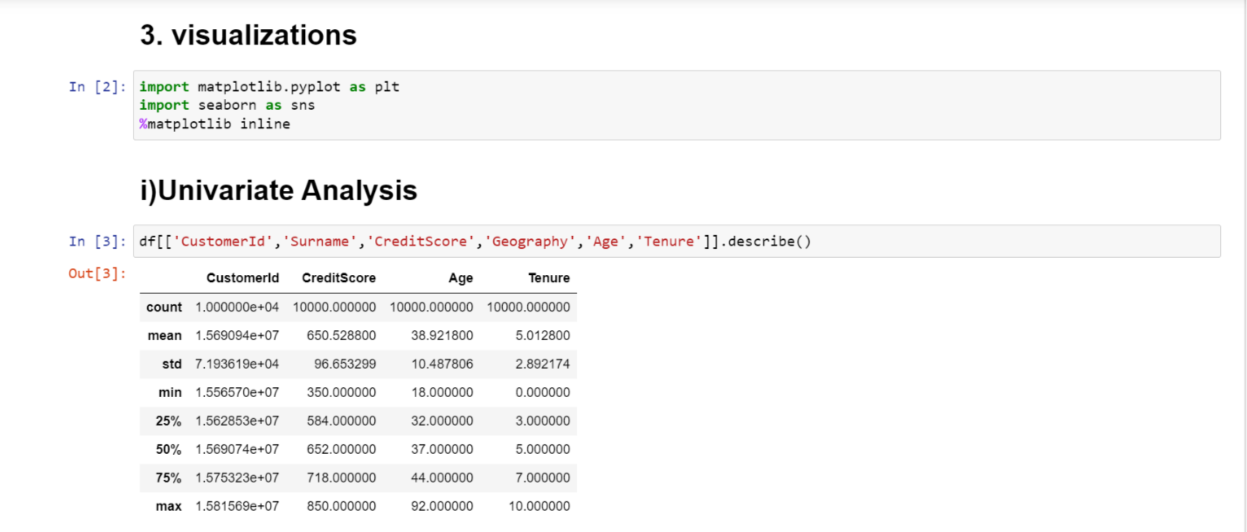


```
In [32]: import numpy as np
         import pandas as pd
         df = pd.read_csv("Churn_Modelling.csv")
         df
```

Out[32]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSa
0	1	15634602	Hargrave	619	France	Female	42	2	0.00	1	1	1	10134
1	2	15647311	Hill	608	Spain	Female	41	1	83807.86	1	0	1	11254
2	3	15619304	Onio	502	France	Female	42	8	159660.80	3	1	0	11393
3	4	15701354	Boni	699	France	Female	39	1	0.00	2	0	0	9382
4	5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	1	1	7908
...	...	...	...	...	...	...	...	...	...	...	...	...	
9995	9996	15606229	Obijiaku	771	France	Male	39	5	0.00	2	1	0	9627
9996	9997	15569892	Johnstone	516	France	Male	35	10	57369.61	1	1	1	10169
9997	9998	15584532	Liu	709	France	Female	36	7	0.00	1	0	1	4208
9998	9999	15682355	Sabbatini	772	Germany	Male	42	3	75075.31	2	1	0	9288
9999	10000	15628319	Walker	792	France	Female	28	4	130142.79	1	1	0	3819

10000 rows × 14 columns



3. visualizations

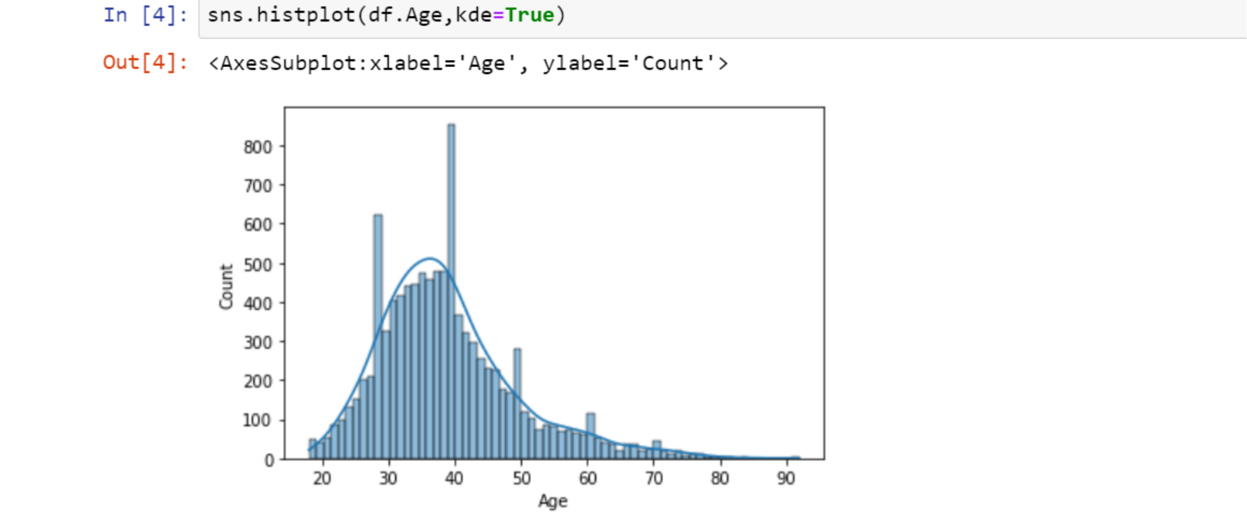
```
In [2]: import matplotlib.pyplot as plt
        import seaborn as sns
        %matplotlib inline
```

i)Univariate Analysis

```
In [3]: df[['CustomerId','Surname','CreditScore','Geography','Age','Tenure']].describe()
```

Out[3]:

	CustomerId	CreditScore	Age	Tenure
count	1.000000e+04	10000.000000	10000.000000	10000.000000
mean	1.569094e+07	650.528800	38.921800	5.012800
std	7.193619e+04	96.653299	10.487806	2.892174
min	1.556570e+07	350.000000	18.000000	0.000000
25%	1.562853e+07	584.000000	32.000000	3.000000
50%	1.569074e+07	652.000000	37.000000	5.000000
75%	1.575323e+07	718.000000	44.000000	7.000000
max	1.581569e+07	850.000000	92.000000	10.000000

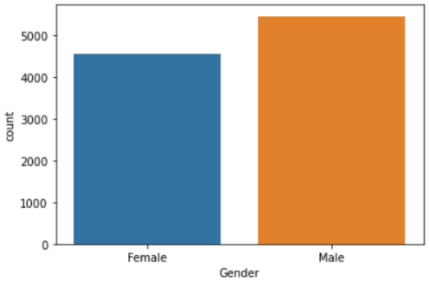


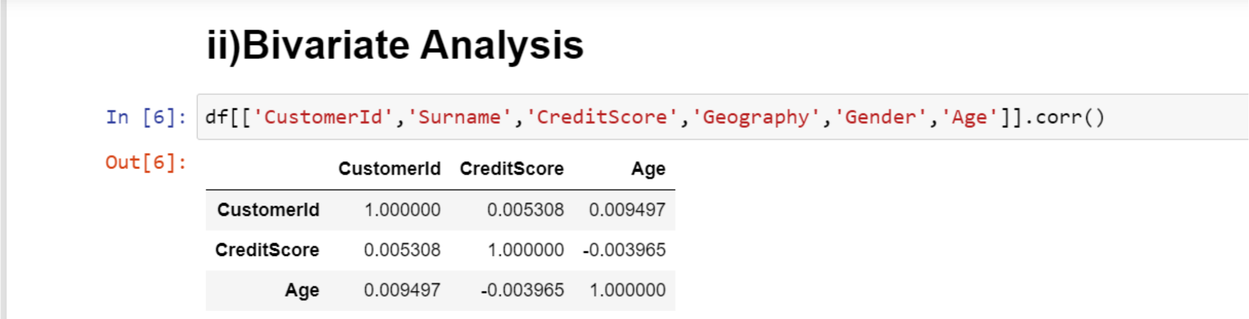
```
In [4]: sns.histplot(df.Age,kde=True)

Out[4]: <AxesSubplot:xlabel='Age', ylabel='Count'>
```

In [5]:
```
# plot count plot for the gender column
sns.countplot(df.Gender)
```

```
C:\Users\User\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword ar
g: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit key
word will result in an error or misinterpretation.
  warnings.warn(
```

Out[5]:
```
<AxesSubplot:xlabel='Gender', ylabel='count'>
```





ii)Bivariate Analysis

In [6]:
```
df[['CustomerId','Surname','CreditScore','Geography','Gender','Age']].corr()
```

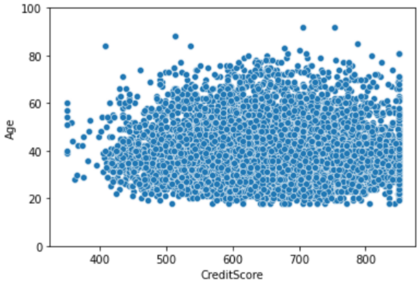
Out[6]:

	CustomerId	CreditScore	Age
CustomerId	1.000000	0.005308	0.009497
CreditScore	0.005308	1.000000	-0.003965
Age	0.009497	-0.003965	1.000000

In [9]:
```
sns.scatterplot(df.CreditScore,df.Age)
plt.ylim(0,100)
```

```
C:\Users\User\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variables as keyword arg
s: x, y. 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(
```

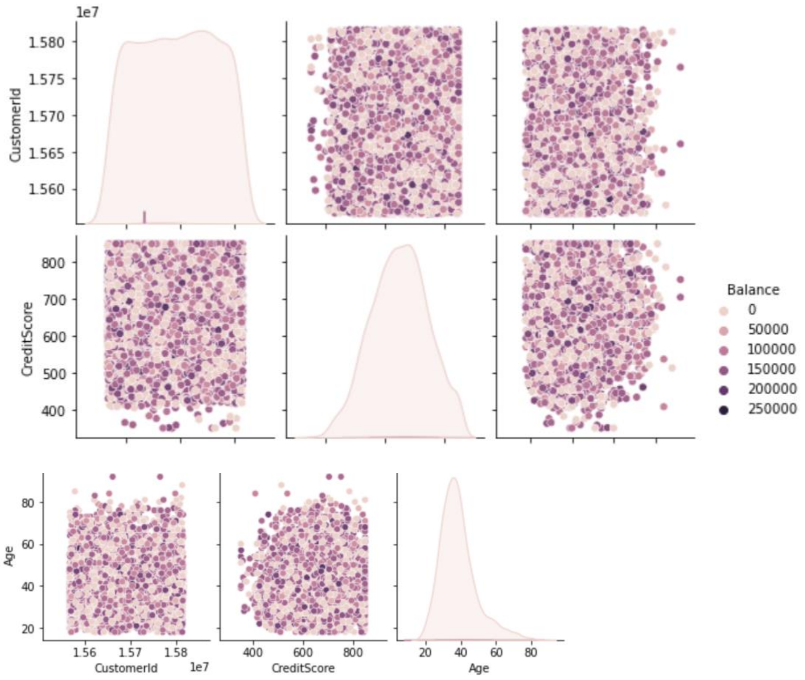
Out[9]:
```
(0.0, 100.0)
```



iii)Multivariate Analysis

```
In [8]: sns.pairplot(data =df[['CustomerId','Geography','Gender','CreditScore','Age','Balance']],hue = 'Balance')
```

```
Out[8]: <seaborn.axisgrid.PairGrid at 0x29ee2462310>
```



4. Descriptive Statistics

```
In [9]: #mode
        df['Age'].mode()
```

```
Out[9]: 0    37
        Name: Age, dtype: int64
```

```
In [10]: #calculation of the mean (for Age)
         df["Age"].mean()
```

```
Out[10]: 38.9218
```

```
In [11]: #calculation of the mean and round the result(for Age)
         round(df["Age"].mean(), 2)
```

```
Out[11]: 38.92
```

```
In [12]: #calculation of the median(for Age)
         df["Age"].median()
```

```
Out[12]: 37.0
```

```
In [33]: df.columns
```

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

In [14]:
```
df["NumOfProducts"].value_counts()
```

Out[14]:
```
1    5084
2    4590
3     266
4      60
Name: NumOfProducts, dtype: int64
```

In [34]:
```
df.dtypes
```

Out[34]:
```
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
```

In [35]:
```
df.head()
```

Out[35]:

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

In [17]:
```
df.describe()
```

Out[17]:

	RowNumber	CustomerId	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	
count	10000.00000	1.000000e+04	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.00000	10000.000000	10000.000000	1(
mean	5000.50000	1.569094e+07	650.528800	38.921800	5.012800	76485.889288	1.530200	0.70550	0.515100	100090.239881	
std	2886.89568	7.193619e+04	96.653299	10.487806	2.892174	62397.405202	0.581654	0.45584	0.499797	57510.492818	
min	1.00000	1.556570e+07	350.000000	18.000000	0.000000	0.000000	1.000000	0.00000	0.000000	11.580000	
25%	2500.75000	1.562853e+07	584.000000	32.000000	3.000000	0.000000	1.000000	0.00000	0.000000	51002.110000	
50%	5000.50000	1.569074e+07	652.000000	37.000000	5.000000	97198.540000	1.000000	1.00000	1.000000	100193.915000	
75%	7500.25000	1.575323e+07	718.000000	44.000000	7.000000	127644.240000	2.000000	1.00000	1.000000	149388.247500	
max	10000.00000	1.581569e+07	850.000000	92.000000	10.000000	250898.090000	4.000000	1.00000	1.000000	199992.480000	

5. Handling Missing values

In [18]:
```
df.isna().any()
```

Out[18]:
```
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
```

In [19]:
```
df.isnull().sum()
```

Out[19]:
```
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
```

In [21]:
```
df.notnull()
```

Out[21]:

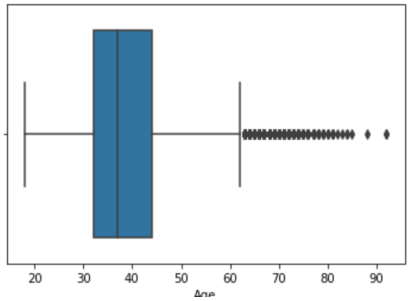
	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSala
0	True	True	True	True	True	True	True	True	True	True	True	True	Tru
1	True	True	True	True	True	True	True	True	True	True	True	True	Tru
2	True	True	True	True	True	True	True	True	True	True	True	True	Tru
3	True	True	True	True	True	True	True	True	True	True	True	True	Tru
4	True	True	True	True	True	True	True	True	True	True	True	True	Tru
...	...	...	...	...	...	...	...	...	...	...	...	...	
9995	True	True	True	True	True	True	True	True	True	True	True	True	Tru
9996	True	True	True	True	True	True	True	True	True	True	True	True	Tru
9997	True	True	True	True	True	True	True	True	True	True	True	True	Tru
9998	True	True	True	True	True	True	True	True	True	True	True	True	Tru
9999	True	True	True	True	True	True	True	True	True	True	True	True	Tru

10000 rows × 14 columns

6. Finding and replacing the outliers

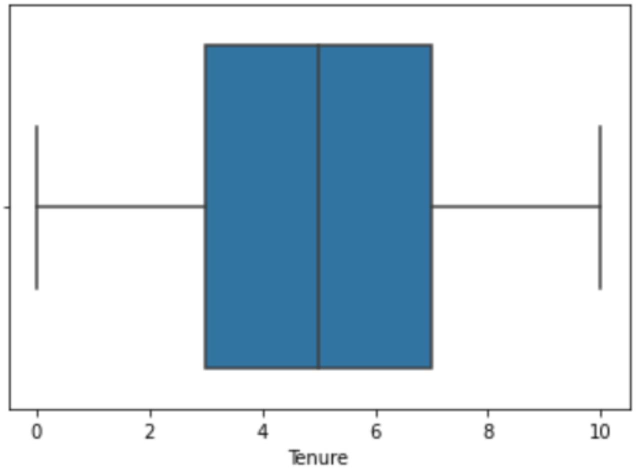
In [22]:
```
import seaborn as sns
sns.boxplot(x=df['Age'])
```

Out[22]:
```
<AxesSubplot:xlabel='Age'>
```



```
In [23]: sns.boxplot(x=df['Tenure'])
```

```
Out[23]: <AxesSubplot:xlabel='Tenure'>
```



7.Check for categorical columns and perform encoding

```
In [24]: import pandas as pd
         df = pd.read_csv("Churn_Modelling.csv", header=None)
```

```
In [25]: cols = df.columns
         num_cols = df._get_numeric_data().columns
```

```
In [26]: num_cols
```

```
Out[26]: Int64Index([], dtype='int64')
```

```
In [27]: list(set(cols) - set(num_cols))
```

```
Out[27]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]
```

8.Split the data into dependent and independent variables

```
In [36]: 
         x =df.drop('Exited',axis=1)
         y=df['Exited']
```

```
In [37]: x.head()
```

Out[37]:

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

```
In [38]: y.head()
```

```
Out[38]: 0    1
         1    0
         2    1
         3    0
         4    0
         Name: Exited, dtype: int64
```

9. Scale the independent varaibles

In [39]:
```
from sklearn import linear_model
from sklearn.preprocessing import StandardScaler
scale = StandardScaler()
```

In [40]:
```
X = df[['Balance', 'Tenure']]

scaledX = scale.fit_transform(X)
print(scaledX)
```

```
[[-1.22584767 -1.04175968]
 [ 0.11735002 -1.38753759]
 [ 1.33305335  1.03290776]
 ...
 [-1.22584767  0.68712986]
 [-0.02260751 -0.69598177]
 [ 0.85996499 -0.35020386]]
```

10.Split the data into training and testing

In [41]:
```
from sklearn.model_selection import train_test_split
```

In [42]:
```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2,random_state=0)
```

In [43]:
```
print('X Train shape:{},Y.Train SHape:{}'.format(x_train.shape,y_train.shape))
```

```
X Train shape:(8000, 13),Y.Train SHape:(8000,)
```

In [44]:
```
print('X Test Shape :{},Y Test SHape:{}'.format(x_test.shape,y_test.shape))
```

```
X Test Shape :(2000, 13),Y Test SHape:(2000,)
```