Assignment -2

Assignment Date	17 September 2022
Student Name	Sivagama Sundari. V
Student Roll Number	211419104255
Maximum Marks	2 Marks

1. Download the dataset:

In [1]:
```
#Downloaded

#importing the libraries

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

import warnings
warnings.filterwarnings('ignore')
```

2. Load the dataset.

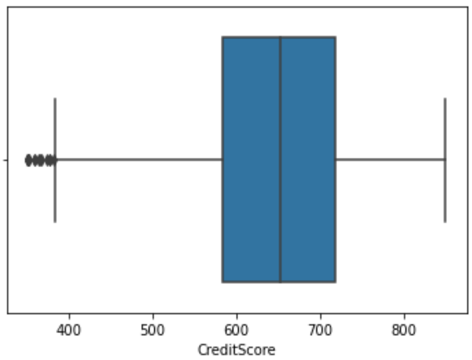
In [2]:
```
#loading the dataset

d = pd.read_csv(r'Downloads/Churn_Modelling.csv')
```

3. Perform Below Visualizations.

• Univariate Analysis

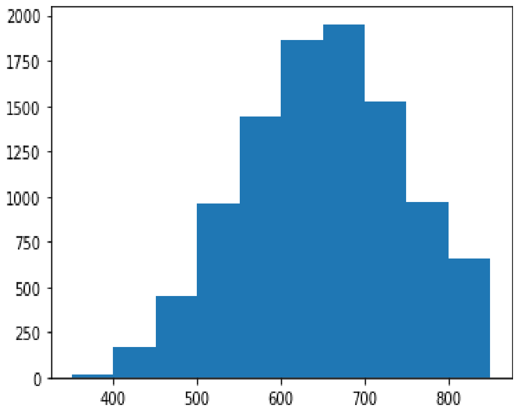
In [3]:
```
#Boxplot

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

Out[3]: <AxesSubplot:xlabel='CreditScore'>



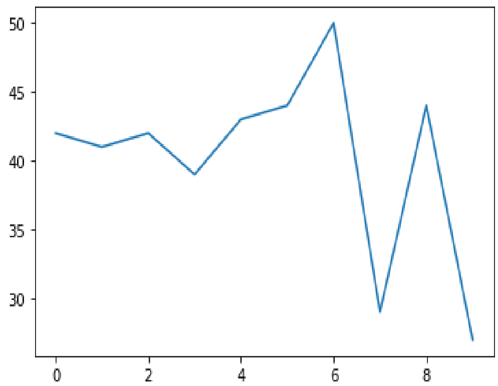
In [4]:
```
#histogram

plt.hist(d['CreditScore'])
```

Out[4]:
```
(array([  19.,  166.,  447.,  958., 1444., 1866., 1952., 1525.,  968.,
         655.]),
 array([350., 400., 450., 500., 550., 600., 650., 700., 750., 800., 850.]),
 <BarContainer object of 10 artists>)
```



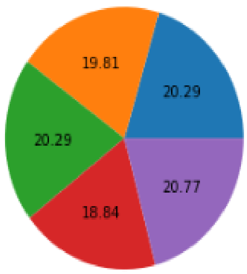
In [5]:
```
#line plot

plt.plot(d['Age'].head(10))
```

Out[5]:
```
[<matplotlib.lines.Line2D at 0x1fd895767c0>]
```



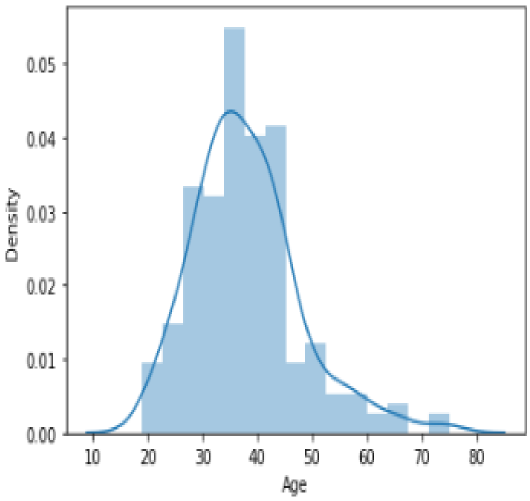
In [6]:
```
#piechart

plt.pie(d['Age'].head(),autopct='%.2f')
```

Out[6]:
```
([<matplotlib.patches.Wedge at 0x1fd895d4f10>,
  <matplotlib.patches.Wedge at 0x1fd895e1640>,
  <matplotlib.patches.Wedge at 0x1fd895e1d60>,
  <matplotlib.patches.Wedge at 0x1fd895ee4c0>,
  <matplotlib.patches.Wedge at 0x1fd895eebe0>],
 [Text(0.8839942345509236, 0.654640506904917, ''),
  Text(-0.3525952068146547, 1.0419580702366729, ''),
  Text(-1.09987331875942, -0.01669379169450419, ''),
  Text(-0.35259525559223215, -1.0419580537304987, ''),
  Text(0.8739574598774371, -0.6679808068534441, '')],
 [Text(0.48217867339141285, 0.3570766401299547, '20.29'),
  Text(-0.19232465826253894, 0.5683407655836397, '19.81'),
  Text(-0.5999309011415017, -0.009105704560638648, '20.29'),
  Text(-0.19232468486849025, -0.5683407565802719, '18.84'),
  Text(0.47670406902405654, -0.364353167374605
8, '20.77')])
```



In [7]:
```
#distplot

sns.distplot(d['Age'].head(200))
```

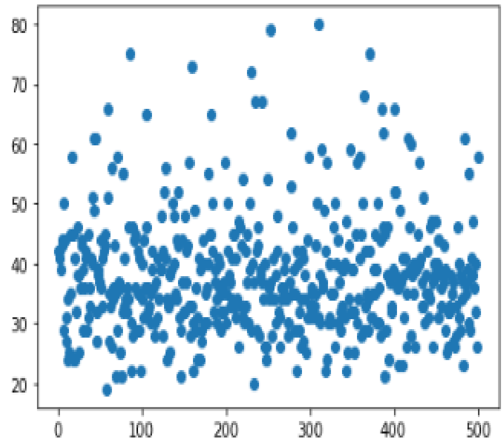
Out[7]:
```
<AxesSubplot:xlabel='Age', ylabel='Density'>
```



• Bi - Variate Analysis

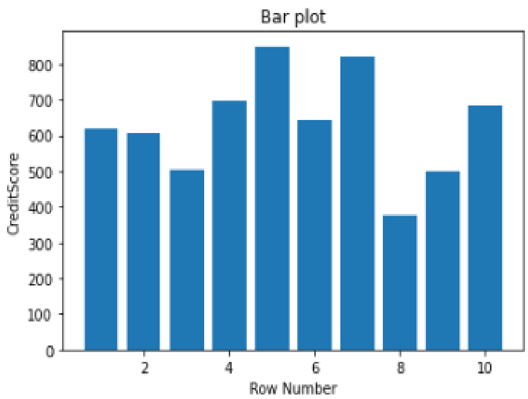
In [8]:
```
#scatter plot

plt.scatter(d['RowNumber'].head(500),d['Age'].head(500))
```

Out[8]: <matplotlib.collections.PathCollection at 0x1fd89b27910>



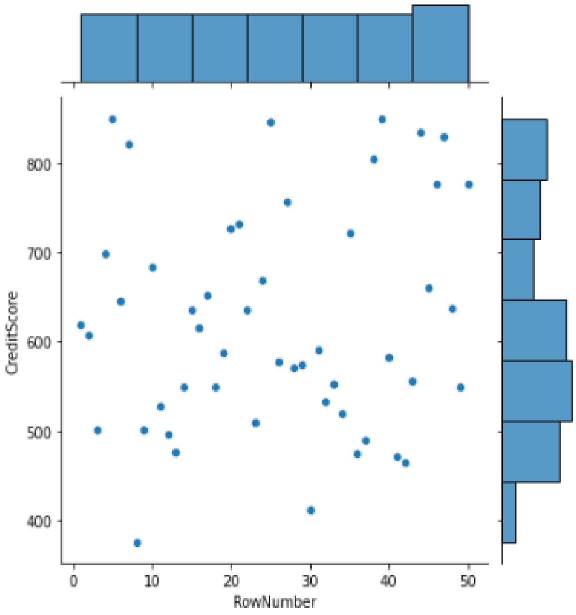
In [9]:
```
#bar plot

plt.bar(d['RowNumber'].head(10),d['CreditScore'].head(10))

#labelling of x,y and result

plt.title('Bar plot')
plt.xlabel('Row Number')
plt.ylabel('CreditScore')
```

Out[9]: Text(0, 0.5, 'CreditScore')



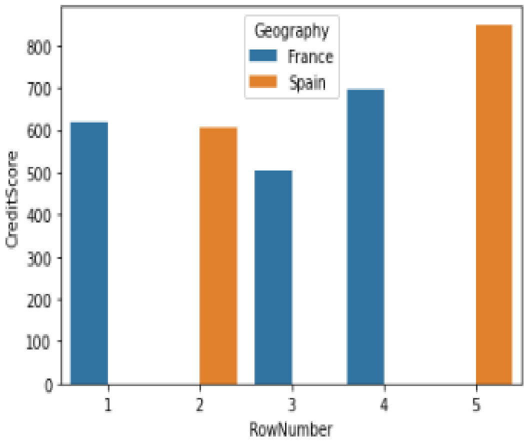
In [10]:
```
#joint plot

sns.jointplot(d['RowNumber'].head(50),d['CreditScore'].head(50))
```

Out[10]: <seaborn.axisgrid.JointGrid at 0x1fd89bc0b80>



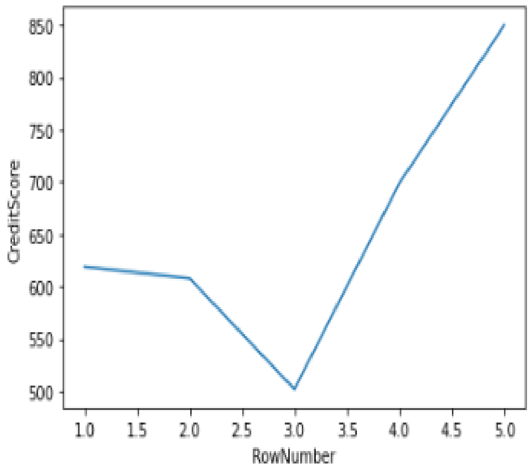
In [11]:
```
#bar plot

sns.barplot('RowNumber','CreditScore',hue='Geography',data=d.head())
```

Out[11]: <AxesSubplot:xlabel='RowNumber', ylabel='CreditScore'>



In [12]:
```
sns.lineplot(d['RowNumber'].head(),d['CreditScore'].head())
```

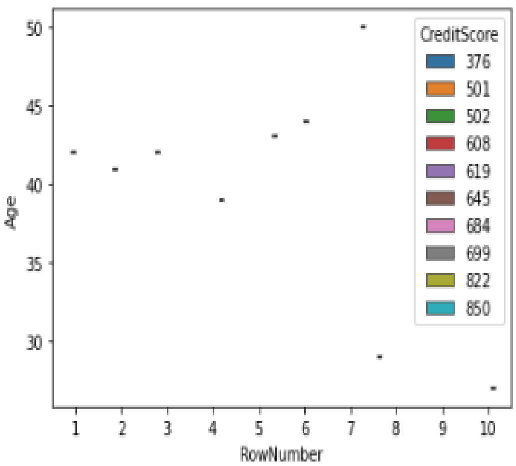
Out[12]: <AxesSubplot:xlabel='RowNumber', ylabel='CreditScore'>



• Multi - Variate Analysis

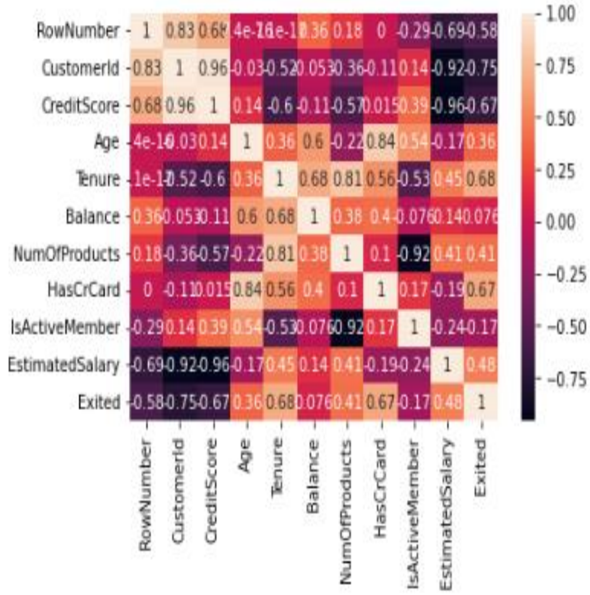
In [13]:
```
#boxplot

sns.boxplot(d['RowNumber'].head(10),d['Age'].head(10),d['CreditScore'].head(10))
```

Out[13]: <AxesSubplot:xlabel='RowNumber', ylabel='Age'>



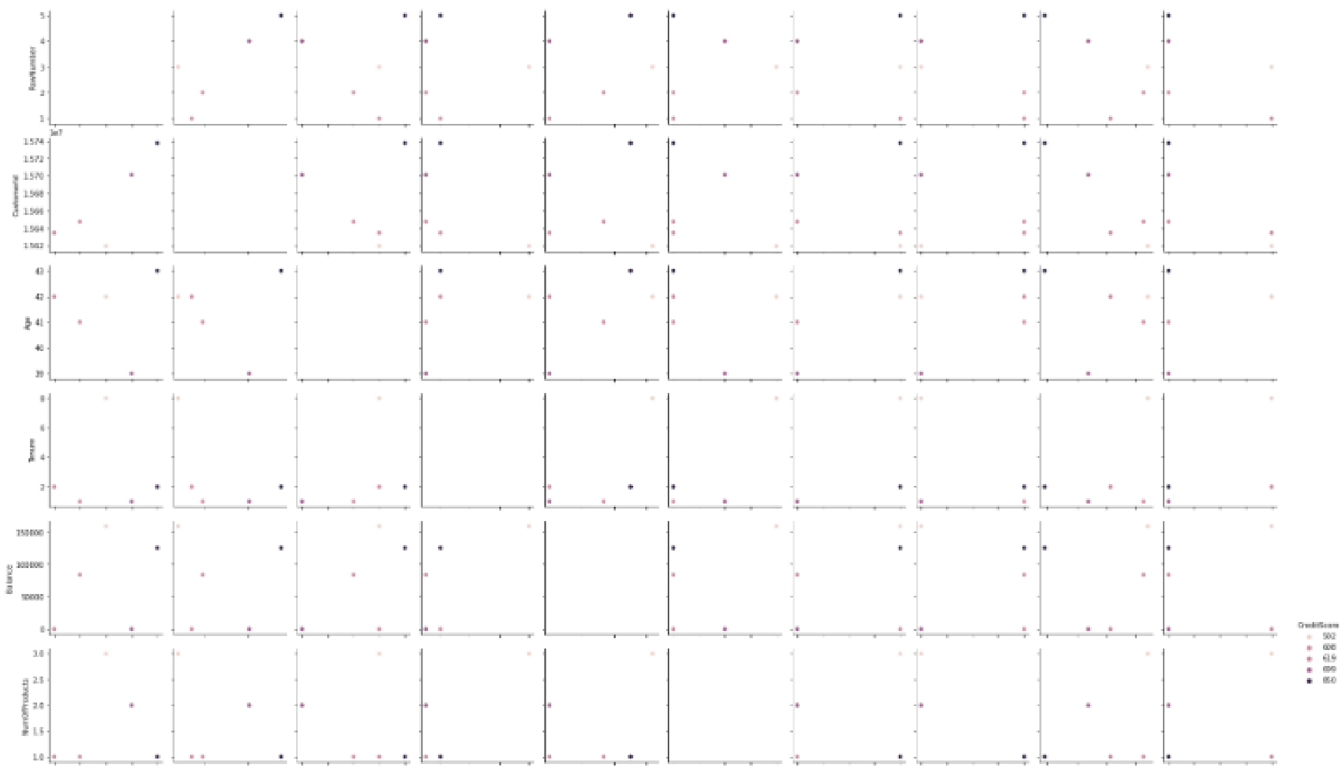
In [14]:
```python
#heat map

sns.heatmap(d.head().corr(),annot=True)
```

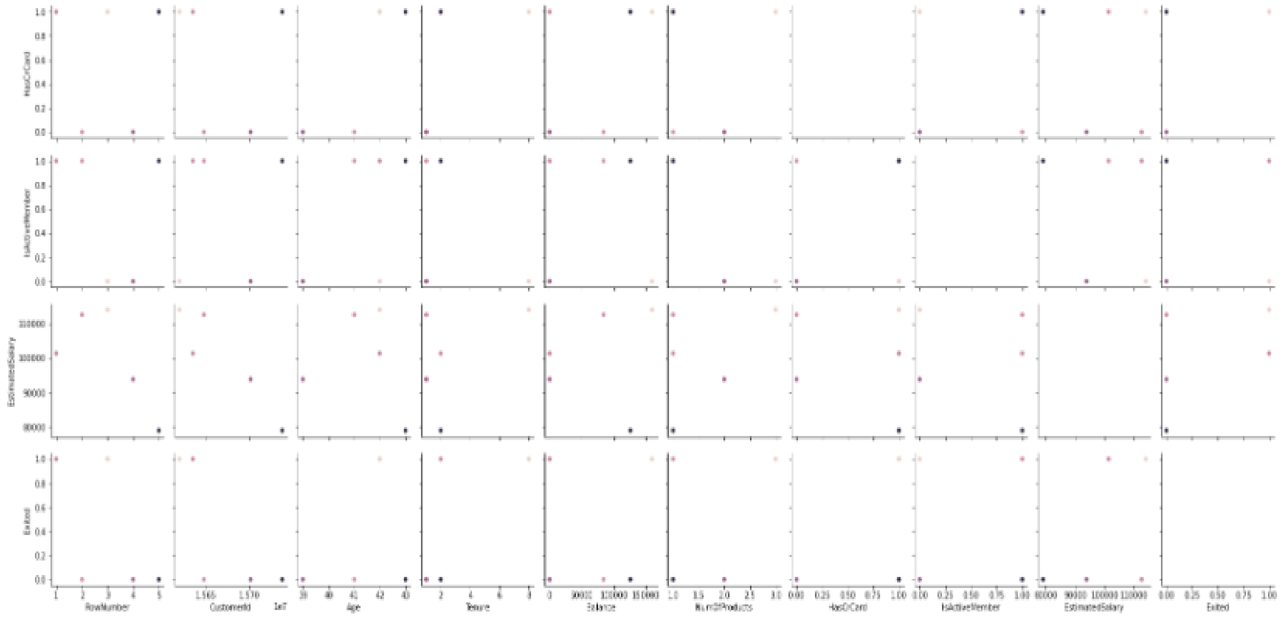
Out[14]: <AxesSubplot:>



In [15]:
```python
#pair plot

sns.pairplot(d.head(),hue='CreditScore')
```

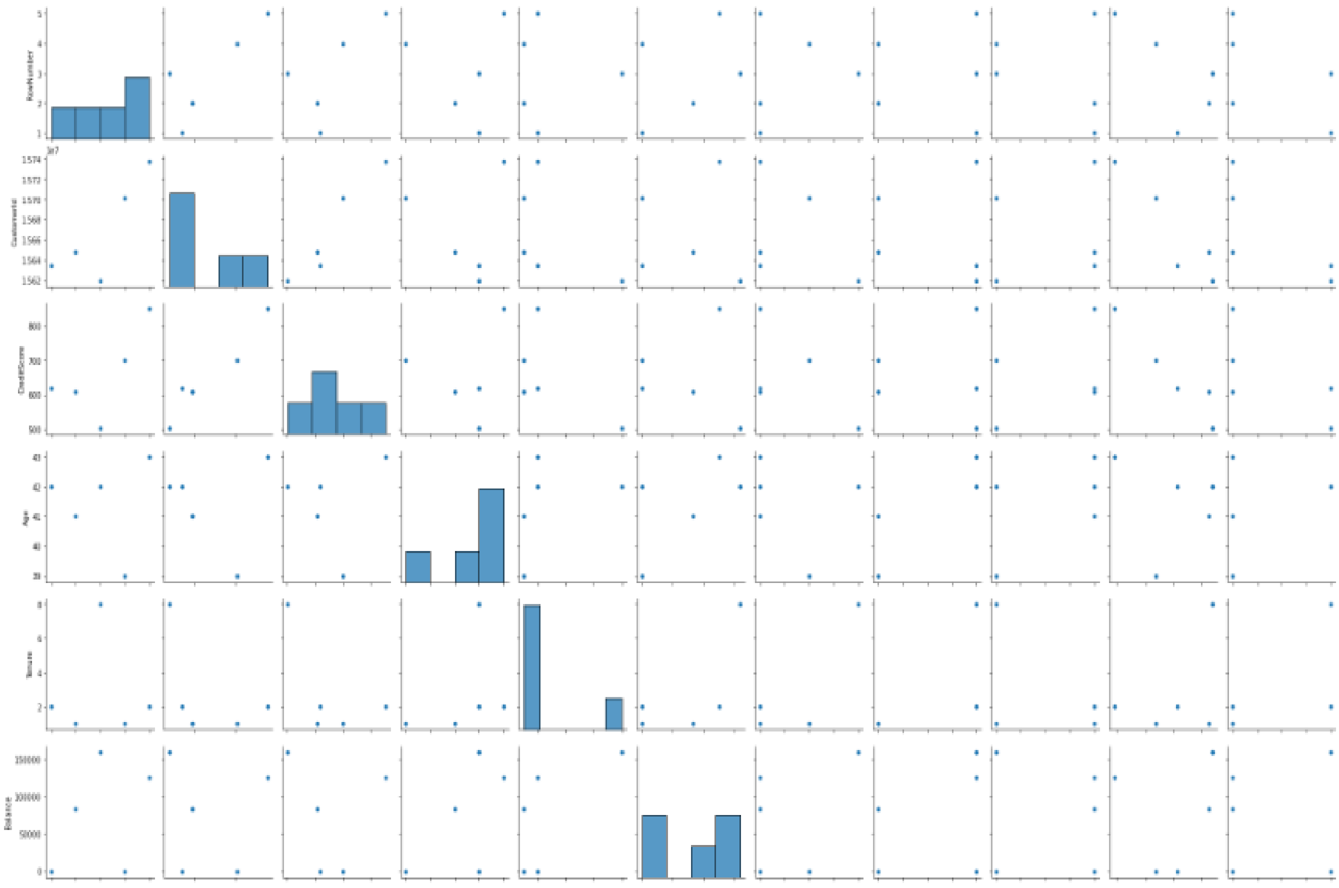
Out[15]: <seaborn.axisgrid.PairGrid at 0x2a8e36ae2e0>

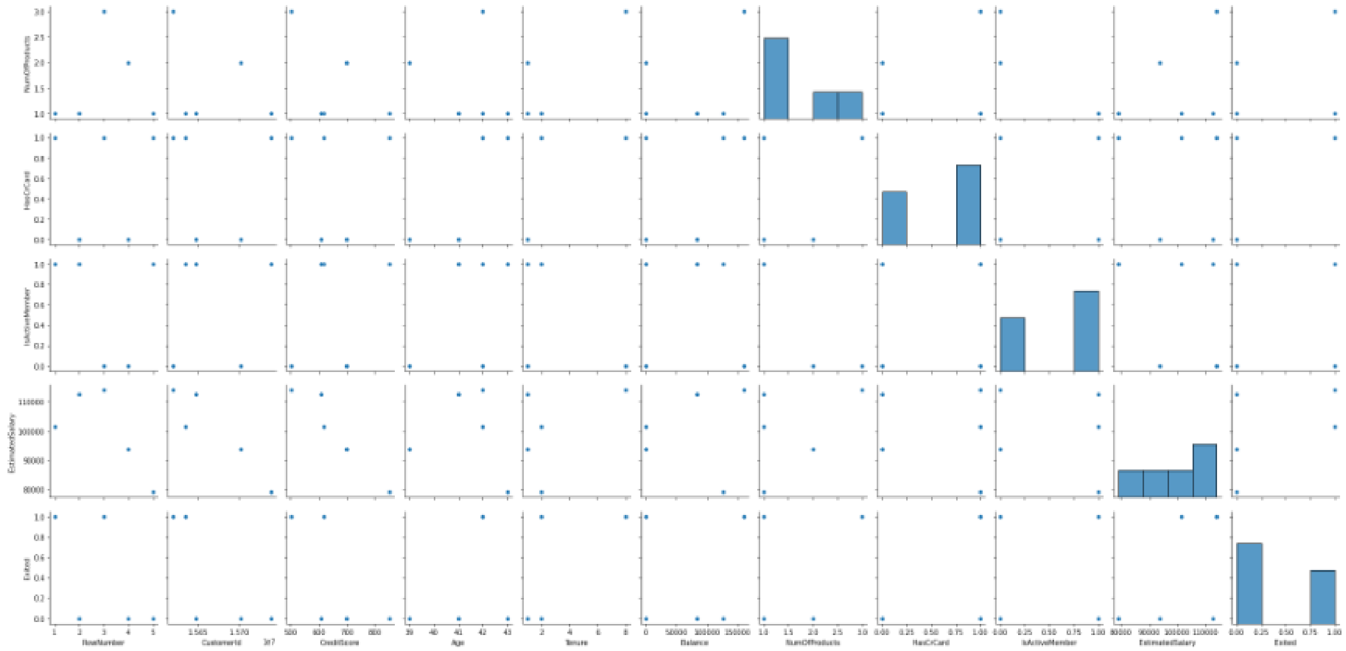




In [16]:
```
sns.pairplot(d.head())
```

Out[16]: <seaborn.axisgrid.PairGrid at 0x2a8e3329580>





4. Perform descriptive statistics on the dataset.

In [17]:
```
#mean

d.mean()
```

Out[17]:
```
RowNumber          5.000500e+03
CustomerId         1.569094e+07
CreditScore        6.505288e+02
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
```

In [18]:
```
#median

d.median()
```

Out[18]:
```
RowNumber          5.000500e+03
CustomerId         1.569074e+07
CreditScore        6.520000e+02
Age                3.700000e+01
Tenure             5.000000e+00
Balance            9.719854e+04
NumOfProducts      1.000000e+00
HasCrCard          1.000000e+00
IsActiveMember     1.000000e+00
EstimatedSalary    1.001939e+05
Exited             0.000000e+00
dtype: float64
```

In [19]:

```
#mode

d.mode()
```

Out[19]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSal
0	1	15565701	Smith	850.0	France	Male	37.0	2.0	0.0	1.0	1.0	1.0	24924
1	2	15565706	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
2	3	15565714	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
3	4	15565779	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
4	5	15565796	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
...	...	...	...	...	...	...	...	...	...	...	...	...	
9995	9996	15815628	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
9996	9997	15815645	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
9997	9998	15815656	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
9998	9999	15815660	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
9999	10000	15815690	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N

10000 rows × 14 columns

In [20]:

```
d.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 [21]:

```
d.shape
```

Out[21]:

```
(10000, 14)
```

In [22]:
```
#kurtosis

d.kurt()
```

Out[22]:
```
RowNumber         -1.200000
CustomerId        -1.196113
CreditScore       -0.425726
Age                1.395347
Tenure            -1.165225
Balance           -1.489412
NumOfProducts      0.582981
HasCrCard         -1.186973
IsActiveMember    -1.996747
EstimatedSalary   -1.181518
Exited             0.165671
dtype: float64
```

In [23]:
```
#skewness

d.skew()
```

Out[23]:
```
RowNumber          0.000000
CustomerId         0.001149
CreditScore       -0.071607
Age                1.011320
Tenure             0.010991
Balance           -0.141109
NumOfProducts      0.745568
HasCrCard         -0.901812
IsActiveMember    -0.060437
EstimatedSalary    0.002085
Exited             1.471611
dtype: float64
```

In [24]:
```
#standard deviation

d.std()
```

Out[24]:
```
RowNumber           2886.895680
CustomerId         71936.186123
CreditScore           96.653299
Age                   10.487806
Tenure                 2.892174
Balance            62397.405202
NumOfProducts          0.581654
HasCrCard              0.455840
IsActiveMember         0.499797
EstimatedSalary    57510.492818
Exited                 0.402769
dtype: float64
```

In [25]:
```
#variance

d.var()
```

Out[25]:
```
RowNumber          8.334167e+06
CustomerId         5.174815e+09
CreditScore        9.341860e+03
Age                1.099941e+02
Tenure             8.364673e+00
Balance            3.893436e+09
NumOfProducts      3.383218e-01
HasCrCard          2.077905e-01
IsActiveMember     2.497970e-01
EstimatedSalary    3.307457e+09
Exited             1.622225e-01
dtype: float64
```

In [26]:
```
#statistical analysis

d.describe()
```

Out[26]:

	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	

In [27]:
```
#finding unique values for columns

d['Gender'].unique()
```

Out[27]:
```
array(['Female', 'Male'], dtype=object)
```

In [28]:
```
d['Geography'].unique()
```

Out[28]:
```
array(['France', 'Spain', 'Germany'], dtype=object)
```

In [29]:
```
quantile= d['Age'].quantile(q=[0.75, 0.25])
quantile
```

Out[29]:
```
0.75    44.0
0.25    32.0
Name: Age, dtype: float64
```

5. Handle the Missing values.

In [30]:
```
#finding missing values

d.isna()
```

Out[30]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSa
0	False	False	False	False	False	False	False	False	False	False	False	False	F
1	False	False	False	False	False	False	False	False	False	False	False	False	F
2	False	False	False	False	False	False	False	False	False	False	False	False	F
3	False	False	False	False	False	False	False	False	False	False	False	False	F
4	False	False	False	False	False	False	False	False	False	False	False	False	F
...	...	...	...	...	...	...	...	...	...	...	...	...	
9995	False	False	False	False	False	False	False	False	False	False	False	False	F
9996	False	False	False	False	False	False	False	False	False	False	False	False	F
9997	False	False	False	False	False	False	False	False	False	False	False	False	F
9998	False	False	False	False	False	False	False	False	False	False	False	False	F
9999	False	False	False	False	False	False	False	False	False	False	False	False	F

10000 rows × 14 columns

In [31]:
```
d.isna().any()
```

Out[31]:
```
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 [32]:
```
d.isna().sum()
```

Out[32]:
```
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 [33]:
```
d.isna().any().sum()
```

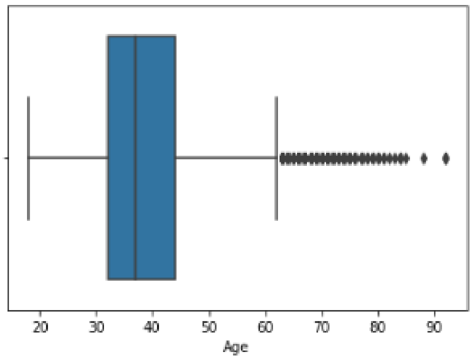
Out[33]:
```
0
```

In [34]:
```
#no missing values
```

6. Find the outliers and replace the outliers

In [35]:
```
#finding outliers

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

Out[35]:
```
<AxesSubplot:xlabel='Age'>
```



In [36]:
```
#handling outliers

qnt=d.quantile(q=[0.25,0.75])
qnt
```

Out[36]:

	RowNumber	CustomerId	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
0.25	2500.75	15628528.25	584.0	32.0	3.0	0.00	1.0	0.0	0.0	51002.1100	0.0
0.75	7500.25	15753233.75	718.0	44.0	7.0	127644.24	2.0	1.0	1.0	149388.2475	0.0

In [37]:
```
iqr=qnt.loc[0.75]-qnt.loc[0.25]

iqr
```

Out[37]:
```
RowNumber            4999.5000
CustomerId         124705.5000
CreditScore           134.0000
Age                    12.0000
Tenure                  4.0000
Balance            127644.2400
NumOfProducts           1.0000
HasCrCard               1.0000
IsActiveMember          1.0000
EstimatedSalary     98386.1375
Exited                  0.0000
dtype: float64
```

In [38]:
```
lower=qnt.loc[0.25]-(1.5*iqr)
lower
```

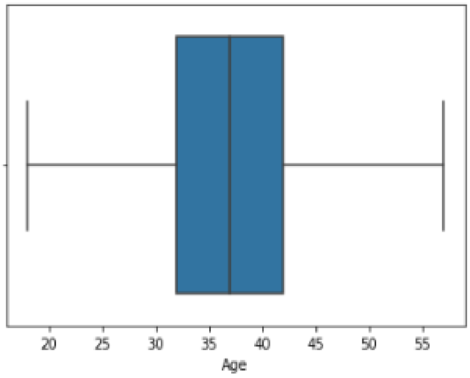
Out[38]:
```
RowNumber         -4.998500e+03
CustomerId         1.544147e+07
CreditScore        3.830000e+02
Age                1.400000e+01
Tenure            -3.000000e+00
Balance           -1.914664e+05
NumOfProducts     -5.000000e-01
HasCrCard         -1.500000e+00
IsActiveMember    -1.500000e+00
EstimatedSalary   -9.657710e+04
Exited             0.000000e+00
dtype: float64
```

In [39]:
```
upper=qnt.loc[0.75]+(1.5*iqr)
upper
```

Out[39]:
```
RowNumber          1.499950e+04
CustomerId         1.594029e+07
CreditScore        9.190000e+02
Age                6.200000e+01
Tenure             1.300000e+01
Balance            3.191106e+05
NumOfProducts      3.500000e+00
HasCrCard          2.500000e+00
IsActiveMember     2.500000e+00
EstimatedSalary    2.969675e+05
Exited             0.000000e+00
dtype: float64
```

In [40]:
```
#replacing outliers

d['Age']=np.where(d['Age']>57,39,d['Age'])
sns.boxplot(d['Age'])
```

Out[40]:
```
<AxesSubplot:xlabel='Age'>
```



7. Check for Categorical columns and perform encoding.

In [41]:
```
#checking for categorical columns

d.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 [42]:
```
d['Geography'].unique()
```

Out[42]:
```
array(['France', 'Spain', 'Germany'], dtype=object)
```

In [43]:
```
d['Gender'].unique()
```

Out[43]:
```
array(['Female', 'Male'], dtype=object)
```

In [44]:
```
d['Surname'].unique()
```

Out[44]:
```
array(['Hargrave', 'Hill', 'Onio', ..., 'Kashiwagi', 'Aldridge',
       'Burbidge'], dtype=object)
```

In [45]:
```
#one hot encoding

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

Out[45]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedS
0	1	15634602	Hargrave	619	France	0	42	2	0.00	1	1	1	1013
1	2	15647311	Hill	608	Spain	0	41	1	83807.86	1	0	1	1125
2	3	15619304	Onio	502	France	0	42	8	159660.80	3	1	0	1139
3	4	15701354	Boni	699	France	0	39	1	0.00	2	0	0	938
4	5	15737888	Mitchell	850	Spain	0	43	2	125510.82	1	1	1	790
...	...	...	...	...	...	...	...	...	...	...	...	...	
9995	9996	15606229	Obijiaku	771	France	1	39	5	0.00	2	1	0	962
9996	9997	15569892	Johnstone	516	France	1	35	10	57369.61	1	1	1	1016
9997	9998	15584532	Liu	709	France	0	36	7	0.00	1	0	1	420
9998	9999	15682355	Sabbatini	772	Germany	1	42	3	75075.31	2	1	0	928
9999	10000	15628319	Walker	792	France	0	28	4	130142.79	1	1	0	381

10000 rows × 14 columns

In [46]:
```
#using dummy variables to encode

d=pd.get_dummies(d,columns=['Geography'])
d
```

Out[46]:

	RowNumber	CustomerId	Surname	CreditScore	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
0	1	15634602	Hargrave	619	0	42	2	0.00	1	1	1	101348.88	1
1	2	15647311	Hill	608	0	41	1	83807.86	1	0	1	112542.58	0
2	3	15619304	Onio	502	0	42	8	159660.80	3	1	0	113931.57	1
3	4	15701354	Boni	699	0	39	1	0.00	2	0	0	93826.63	0
4	5	15737888	Mitchell	850	0	43	2	125510.82	1	1	1	79084.10	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
9995	9996	15606229	Obijiaku	771	1	39	5	0.00	2	1	0	96270.64	0
9996	9997	15569892	Johnstone	516	1	35	10	57369.61	1	1	1	101699.77	0
9997	9998	15584532	Liu	709	0	36	7	0.00	1	0	1	42085.58	1
9998	9999	15682355	Sabbatini	772	1	42	3	75075.31	2	1	0	92888.52	1
9999	10000	15628319	Walker	792	0	28	4	130142.79	1	1	0	38190.78	0

10000 rows × 16 columns

In [47]:
```
d=pd.get_dummies(d,columns=['Surname'])
d
```

Out[47]:

	RowNumber	CustomerId	CreditScore	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	...	Surname_Zinachukwudi	Surna
0	1	15634602	619	0	42	2	0.00	1	1	1	...	0	
1	2	15647311	608	0	41	1	83807.86	1	0	1	...	0	
2	3	15619304	502	0	42	8	159660.80	3	1	0	...	0	
3	4	15701354	699	0	39	1	0.00	2	0	0	...	0	
4	5	15737888	850	0	43	2	125510.82	1	1	1	...	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
9995	9996	15606229	771	1	39	5	0.00	2	1	0	...	0	
9996	9997	15569892	516	1	35	10	57369.61	1	1	1	...	0	
9997	9998	15584532	709	0	36	7	0.00	1	0	1	...	0	
9998	9999	15682355	772	1	42	3	75075.31	2	1	0	...	0	
9999	10000	15628319	792	0	28	4	130142.79	1	1	0	...	0	

10000 rows × 2947 columns

8. Split the data into dependent and independent variables.

In [48]:
```
d.head()
```

Out[48]:

	RowNumber	CustomerId	CreditScore	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	...	Surname_Zinachukwudi	Surname_
0	1	15634602	619	0	42	2	0.00	1	1	1	...	0	
1	2	15647311	608	0	41	1	83807.86	1	0	1	...	0	
2	3	15619304	502	0	42	8	159660.80	3	1	0	...	0	
3	4	15701354	699	0	39	1	0.00	2	0	0	...	0	
4	5	15737888	850	0	43	2	125510.82	1	1	1	...	0	

5 rows × 2947 columns

In [49]:
```
x=d.iloc[ : ].values
y=d.iloc[:,2].values
print(x)
```

```
[[1.0000000e+00 1.5634602e+07 6.1900000e+02 ... 0.0000000e+00
  0.0000000e+00 0.0000000e+00]
 [2.0000000e+00 1.5647311e+07 6.0800000e+02 ... 0.0000000e+00
  0.0000000e+00 0.0000000e+00]
 [3.0000000e+00 1.5619304e+07 5.0200000e+02 ... 0.0000000e+00
  0.0000000e+00 0.0000000e+00]
 ...
 [9.9980000e+03 1.5584532e+07 7.0900000e+02 ... 0.0000000e+00
  0.0000000e+00 0.0000000e+00]
 [9.9990000e+03 1.5682355e+07 7.7200000e+02 ... 0.0000000e+00
  0.0000000e+00 0.0000000e+00]
 [1.0000000e+04 1.5628319e+07 7.9200000e+02 ... 0.0000000e+00
  0.0000000e+00 0.0000000e+00]]
```

In [50]:
```
y
```
Out[50]:
```
array([619, 608, 502, ..., 709, 772, 792], dtype=int64)
```

In [51]:
```
x=d.drop(columns= ['EstimatedSalary']).values
y=d['EstimatedSalary'].values
x
```
Out[51]:
```
array([[1.0000000e+00, 1.5634602e+07, 6.1900000e+02, ..., 0.0000000e+00,
        0.0000000e+00, 0.0000000e+00],
       [2.0000000e+00, 1.5647311e+07, 6.0800000e+02, ..., 0.0000000e+00,
        0.0000000e+00, 0.0000000e+00],
       [3.0000000e+00, 1.5619304e+07, 5.0200000e+02, ..., 0.0000000e+00,
        0.0000000e+00, 0.0000000e+00],
       ...,
       [9.9980000e+03, 1.5584532e+07, 7.0900000e+02, ..., 0.0000000e+00,
        0.0000000e+00, 0.0000000e+00],
       [9.9990000e+03, 1.5682355e+07, 7.7200000e+02, ..., 0.0000000e+00,
        0.0000000e+00, 0.0000000e+00],
       [1.0000000e+04, 1.5628319e+07, 7.9200000e+02, ..., 0.0000000e+00,
        0.0000000e+00, 0.0000000e+00]])
```

In [52]:
```
y
```
Out[52]:
```
array([101348.88, 112542.58, 113931.57, ...,  42085.58,  92888.52,
        38190.78])
```

9. Scale the independent variables

In [53]:
```
from sklearn.preprocessing import scale  #StandardScaler
# Scale - Similar to std
```

In [54]:
```
#Scaling the independent variables

x = scale(x)
x
```
Out[54]:
```
array([[-1.73187761, -0.78321342, -0.32622142, ..., -0.0100005 ,
        -0.01414355, -0.01414355],
       [-1.7315312 , -0.60653412, -0.44003595, ..., -0.0100005 ,
        -0.01414355, -0.01414355],
       [-1.73118479, -0.99588476, -1.53679418, ..., -0.0100005 ,
        -0.01414355, -0.01414355],
       ...,
       [ 1.73118479, -1.47928179,  0.60498839, ..., -0.0100005 ,
        -0.01414355, -0.01414355],
       [ 1.7315312 , -0.11935577,  1.25683526, ..., -0.0100005 ,
        -0.01414355, -0.01414355],
       [ 1.73187761, -0.87055909,  1.46377078, ..., -0.0100005 ,
        -0.01414355, -0.01414355]])
```

In [55]:
```
x.mean()
```
Out[55]:
```
2.348215911527609e-18
```

In [56]:
```
x.std()
```
Out[56]:
```
1.0000000000000102
```

10. Split the data into training and testing

In [57]:
```
from sklearn.model_selection import train_test_split
```

In [58]:
```
#spliting data to train and test

x_train, x_test, y_train, y_test = train_test_split(x,y, test_size = 0.2)
print(x_train.shape, x_test.shape)
```
```
(8000, 2946) (2000, 2946)
```