Assignment 4

ASSIGNMENT 4 - Customer Segmentation Analysis

1.Dataset - “Mall_customers.csv”.

2.Load the Dataset into tool.

[1]:
```
#importing lbraries
import pandas as pd
#load the dataset
df=pd.read_csv("Mall_customers.csv")
df
```

[1]:

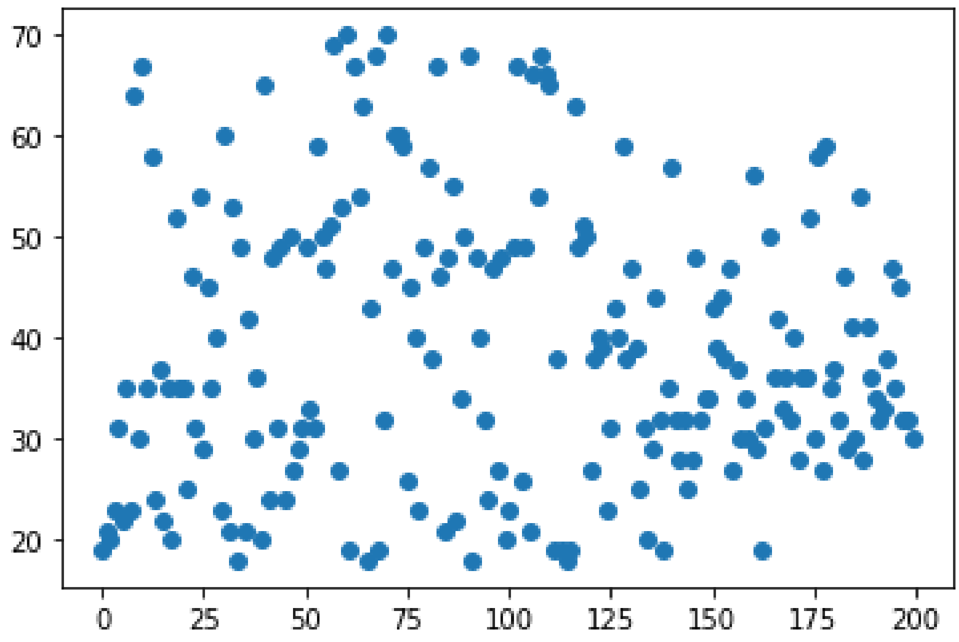
	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40
..	...	...	...	...	...
195	196	Female	35	120	79
196	197	Female	45	126	28
197	198	Male	32	126	74
198	199	Male	32	137	18
199	200	Male	30	137	83

[200 rows x 5 columns]

3. Perform Below Visualizations.

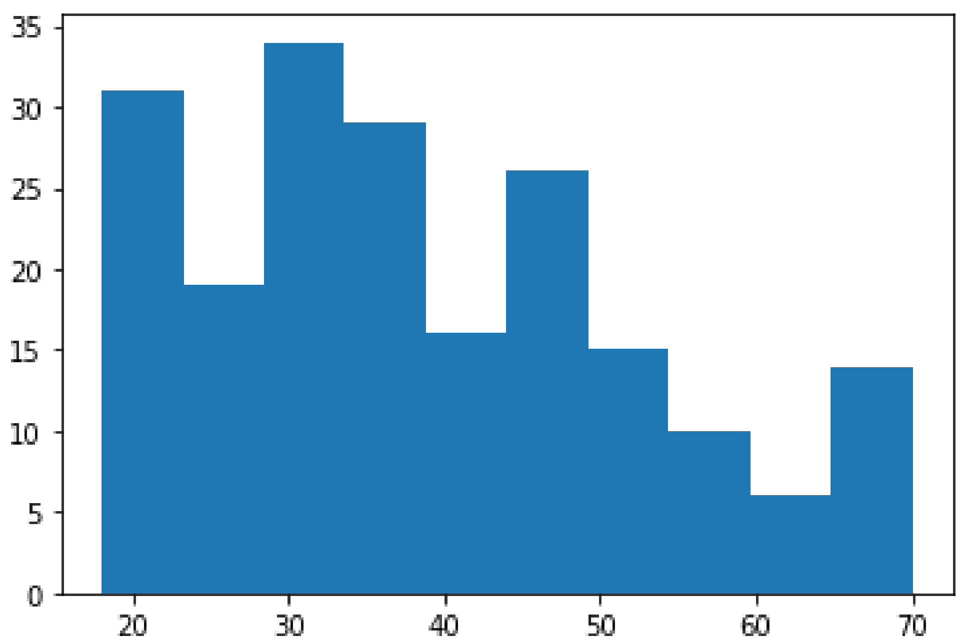
3.1 Univariate Analysis

[2]:
```
#scatterplot
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
#load the dataset
df=pd.read_csv("Mall_customers.csv")
plt.scatter(df.index,df['Age'])
plt.show()
```



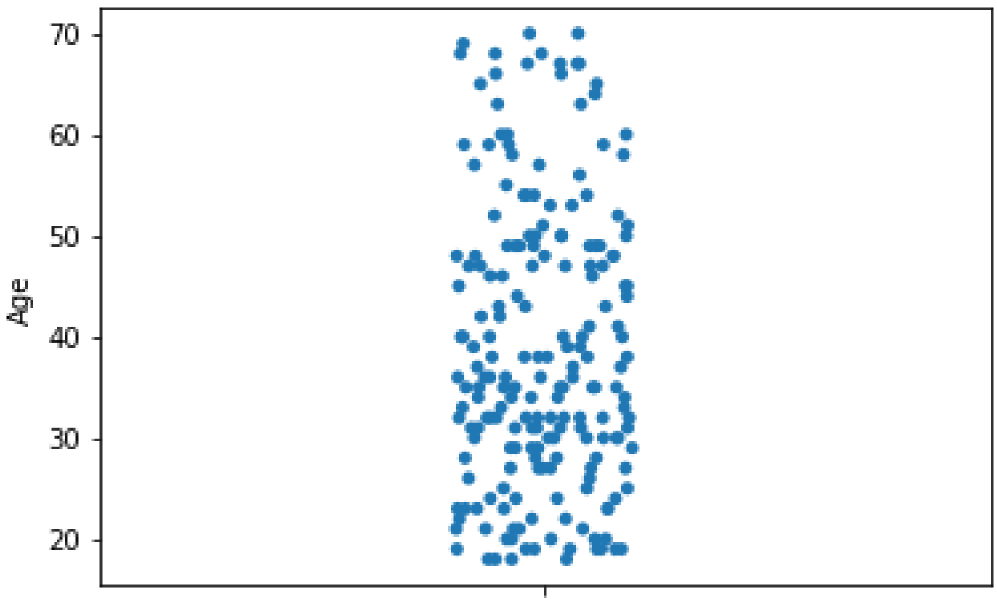
[3]:
```
plt.hist(df["Age"])
```

[3]: (array([31., 19., 34., 29., 16., 26., 15., 10., 6., 14.]),
array([18. , 23.2, 28.4, 33.6, 38.8, 44. , 49.2, 54.4, 59.6, 64.8, 70.]),
<BarContainer object of 10 artists>)

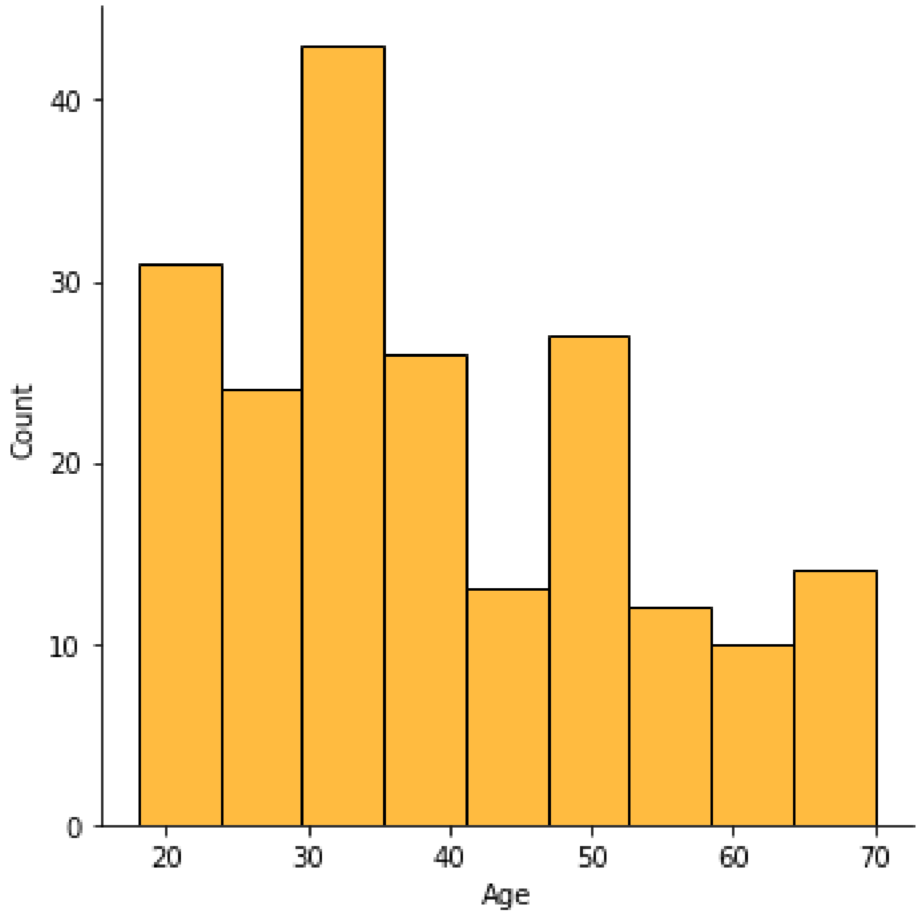


3.2 Bi - Variate Analysis

[4]:
```
#strip plot
sns.stripplot(y=df["Age"])
```
[4]: <AxesSubplot:ylabel='Age'>



[5]:
```
import seaborn as sns
sns.displot(df["Age"], color="orange")
```
[5]: <seaborn.axisgrid.FacetGrid at 0x291d12450a0>

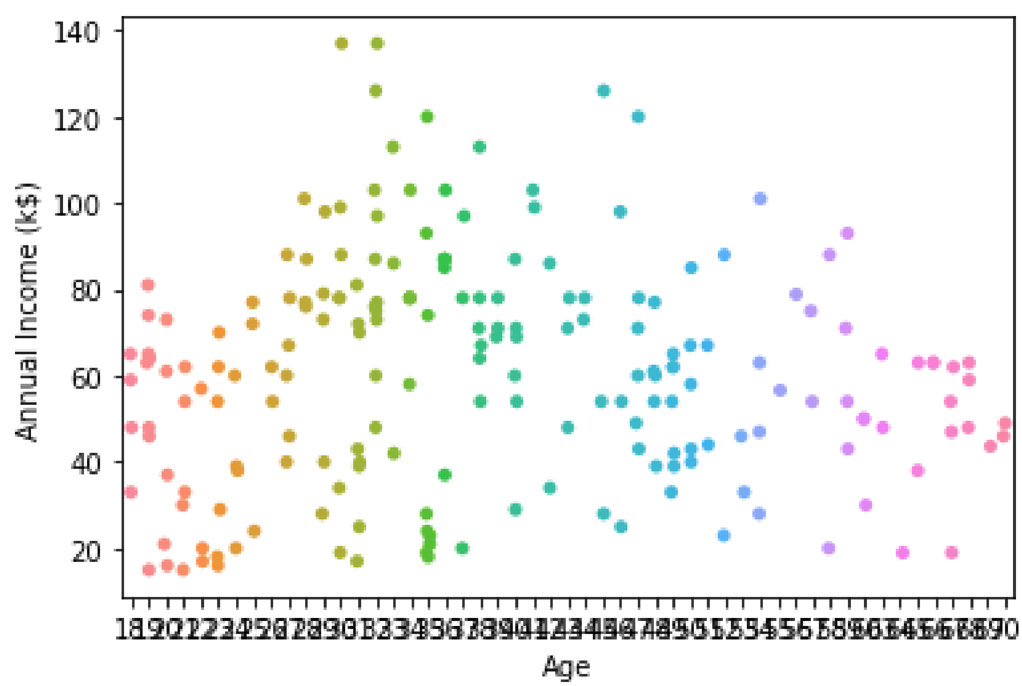


[6]:
```
plt.scatter(df["Gender"],df["Annual Income (k$)"])
```

[6]: <matplotlib.collections.PathCollection at 0x291d13969d0>

[7]:
```
#strip plot
sns.stripplot(x=df['Age'],y=df['Annual Income (k$)'])
```

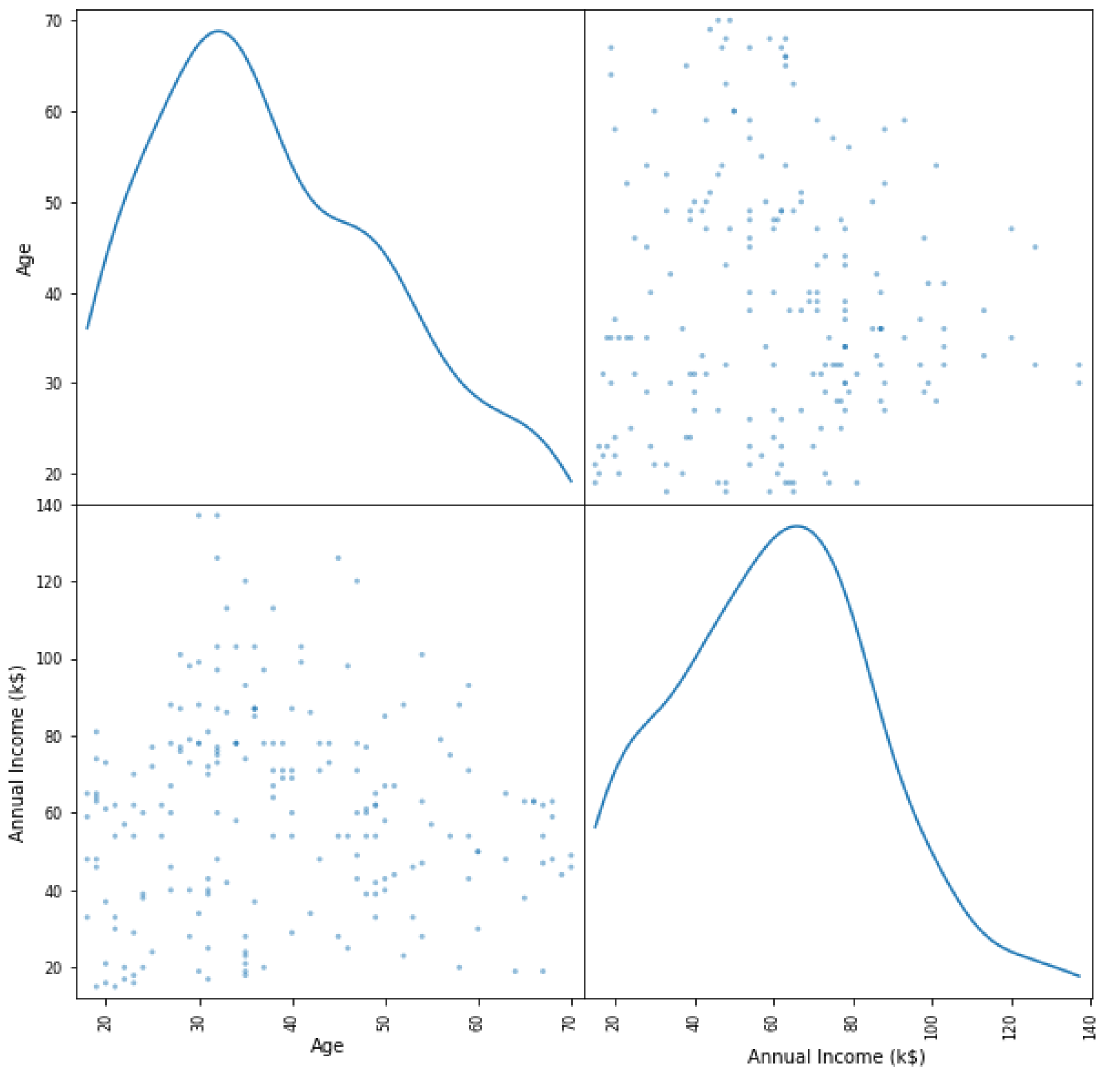
[7]: <AxesSubplot:xlabel='Age', ylabel='Annual Income (k$)'>



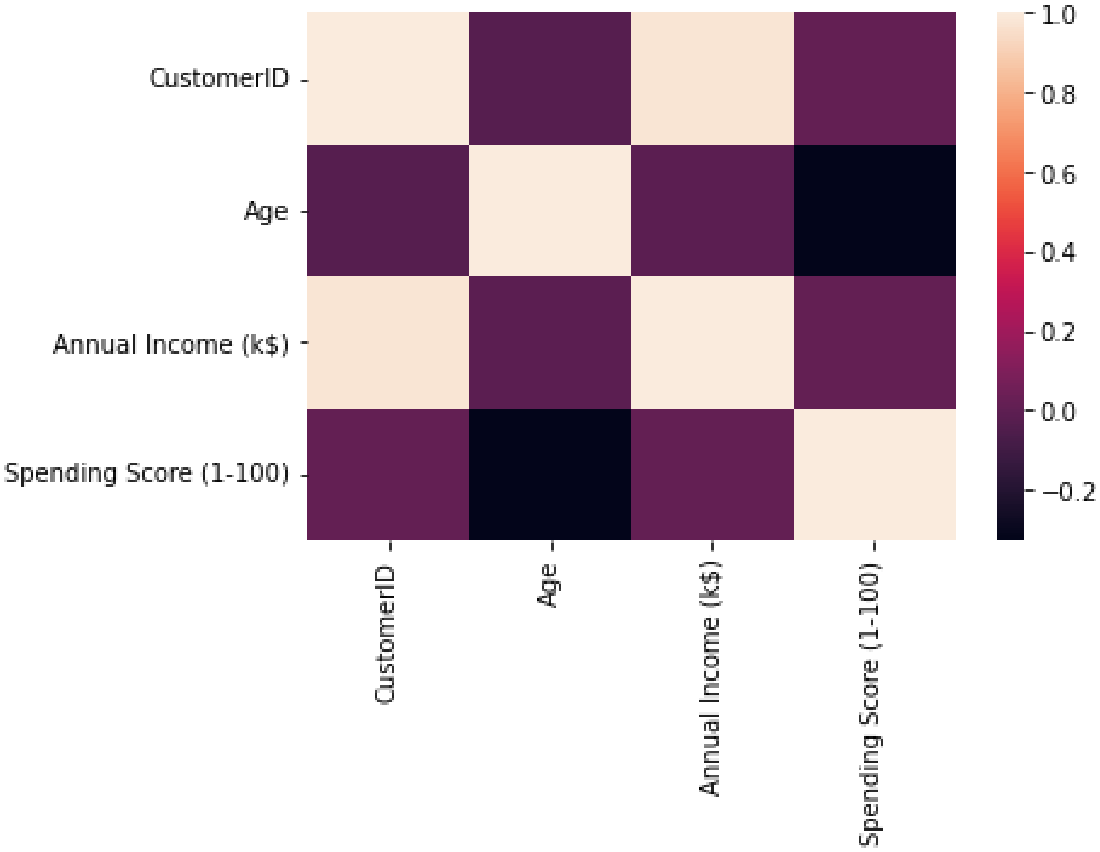
3.3 Multivariate Analysis

[8]:
```
pd.plotting.scatter_matrix(df.loc[:,"Age":"Annual Income (k$)"] , diagonal
↪="kde", figsize=(10,10))
```

[8]:
```
array([[<AxesSubplot:xlabel='Age', ylabel='Age'>,
        <AxesSubplot:xlabel='Annual  Income  (k$)',  ylabel='Age'>],[<AxesSubplot:xlabel='Age',
      ylabel='Annual  Income  (k$)'>,
        <AxesSubplot:xlabel='Annual Income (k$)', ylabel='Annual Income(k$)'>]],
    dtype=object)
```



[9]:
```
sns.heatmap(df.corr())
```
[9]: <AxesSubplot:>



4. Perform descriptive statistics on the dataset.

[10]:
```
df.describe()
```
[10]:

	CustomerID	Age	Annual Income (k$)	Spending Score (1-100)
count	200.000000	200.000000	200.000000	200.000000
mean	100.500000	38.850000	60.560000	50.200000
std	57.879185	13.969007	26.264721	25.823522
min	1.000000	18.000000	15.000000	1.000000
25%	50.750000	28.750000	41.500000	34.750000
50%	100.500000	36.000000	61.500000	50.000000
75%	150.250000	49.000000	78.000000	73.000000

[11]:
```
df.describe().T
```

[11]:

	count	mean	std	min	25%	50%	75%	\
CustomerID	200.0	100.50	57.879185	1.0	50.75	100.5	150.25	
Age	200.0	38.85	13.969007	18.0	28.75	36.0	49.00	
Annual Income (k$)	200.0	60.56	26.264721	15.0	41.50	61.5	78.00	
Spending Score (1-100)	200.0	50.20	25.823522	1.0	34.75	50.0	73.00	

	max
CustomerID	200.0
Age	70.0
Annual Income (k$)	137.0
Spending Score (1-100)	99.0

[12]:

```
df.head()
```

[12]:

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40

[13]:

```
df.head(10)
```

[13]:

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40
5	6	Female	22	17	76
6	7	Female	35	18	6
7	8	Female	23	18	94
8	9	Male	64	19	3
9	10	Female	30	19	72

[14]:

```
df.tail()
```

[14]:

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)
195	196	Female	35	120	79
196	197	Female	45	126	28
197	198	Male	32	126	74
198	199	Male	32	137	18
199	200	Male	30	137	83

[15]:

```
df.tail(10)
```

[15]:

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)
190	191	Female	34	103	23

191	192	Female	32	103	69
192	193	Male	33	113	8
193	194	Female	38	113	91
194	195	Female	47	120	16
195	196	Female	35	120	79
196	197	Female	45	126	28
197	198	Male	32	126	74
198	199	Male	32	137	18
199	200	Male	30	137	83

[16]:
```
df.info()
```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199 Data
columns (total 5 columns):

#	Column	Non-Null Count	Dtype
0	CustomerID	200 non-null	int64
1	Gender	200 non-null	object
2	Age	200 non-null	int64
3	Annual Income (k$)	200 non-null	int64
4	Spending Score (1-100)	200 non-null	int64

dtypes: int64(4), object(1)
[16]: memory usage: 7.9+ KB

[17]: (
```
df.shape
```

[18]:
```
df.median()
```

C:\Users\janar vijay\AppData\Local\Temp\ipykernel_8400\530051474.py:1: FutureWarning: Dropping of nuisance columns in DataFrame reductions (with'numeric_only=None') is deprecated; in a future version this will raise TypeError. Select only valid columns before calling the reduction.
df.median()

[18]:

CustomerID	100.5
Age	36.0
Annual Income (k$)	61.5
Spending Score (1-100)	50.0

dtype: float64

[19]:
```
df.mode()
```

[19]:

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Female	32.0	54.0	42.0
1	2	NaN	NaN	78.0	NaN

```
2                3     NaN   NaN              NaN                  NaN
3                4     NaN   NaN              NaN                  NaN
4                5     NaN   NaN              NaN                  NaN
..             ...   ...   ...              ...                  ...
195            196     NaN   NaN              NaN                  NaN
196            197     NaN   NaN              NaN                  NaN
197            198     NaN   NaN              NaN                  NaN
198            199     NaN   NaN              NaN                  NaN
199            200     NaN   NaN              NaN                  NaN

[200 rows x 5 columns]
```

5.Handle the Missing values.

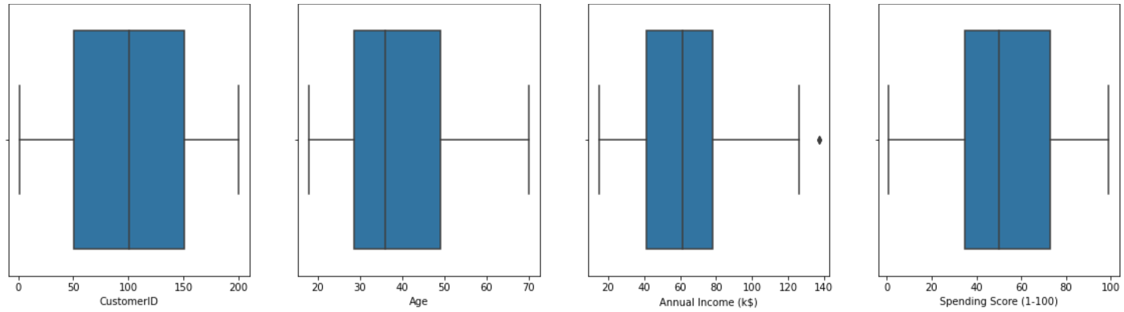
[20]:
```
df.isna().sum()
```

[20]:
```
CustomerID                0
Gender                    0
Age                       0
Annual Income (k$)        0
Spending Score (1-100)    0
dtype: int64
```

6. Find the outliers and replace the outliers

[21]:
```
fig,ax=plt.subplots(figsize=(25,5))

plt.subplot(1, 5, 2)
sns.boxplot(x=df['Age'])

plt.subplot(1, 5, 3)
sns.boxplot(x=df['Annual Income (k$)'])

plt.subplot(1, 5, 4)
sns.boxplot(x=df['Spending Score (1-100)'])

plt.subplot(1, 5, 1)
sns.boxplot(x=df['CustomerID'])
```

[21]: <



Handling Outlier

[22]:
```
quant=df.quantile(q=[0.25,0.75])
quant
```

[22]:

	CustomerID	Age	Annual Income (k$)	Spending Score (1-100)
0.25	50.75	28.75	41.5	34.75
0.75	150.25	49.00	78.0	73.00

[23]:
```
quant.loc[0.75]
```

[23]:
```
CustomerID                150.25
Age                        49.00
Annual Income (k$)         78.00
Spending Score (1-100)     73.00
Name: 0.75, dtype: float64
```

[24]:
```
quant.loc[0.25]
```

[24]:
```
CustomerID                50.75
Age                       28.75
Annual Income (k$)        41.50
Spending Score (1-100)    34.75
Name: 0.25, dtype: float64
```

[25]:
```
iqr=quant.loc[0.75]-quant.loc[0.25]
iqr
```

[25]:
```
CustomerID                99.50
Age                       20.25
Annual Income (k$)        36.50
Spending Score (1-100)    38.25
dtype: float64
```

[26]:
```
lower=quant.loc[0.25]-(1.5 *iqr)
lower
```

[26]:
```
CustomerID               -98.500
Age                       -1.625
Annual Income (k$)       -13.250
Spending Score (1-100)   -22.625
dtype: float64
```

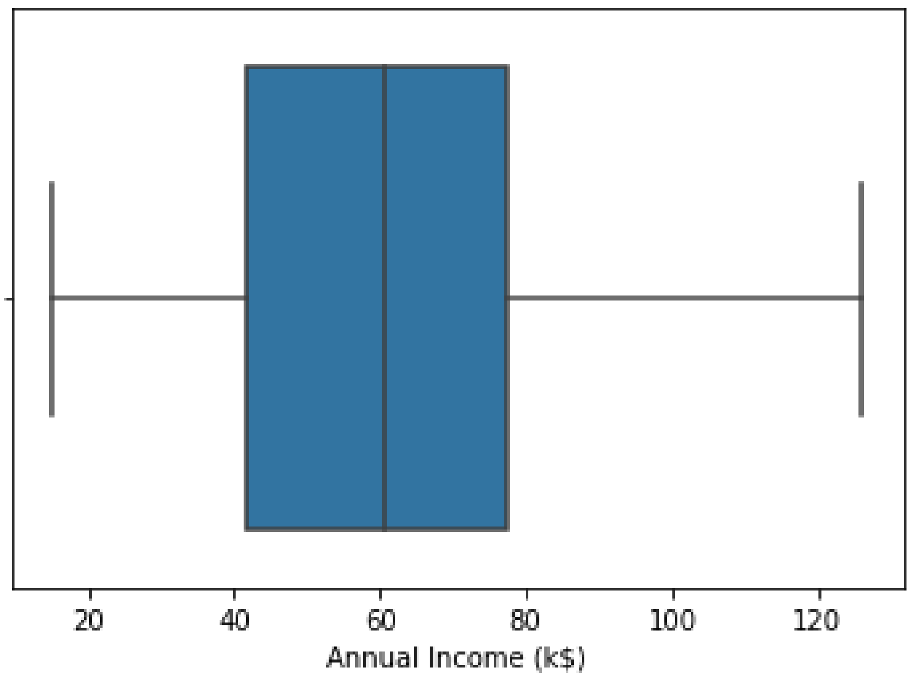
[27]:
```
upper=quant.loc[0.75]+(1.5 *iqr)
upper
```

[27]:
```
CustomerID               299.500
Age                       79.375
Annual Income (k$)       132.750
Spending Score (1-100)   130.375
dtype: float64
```

[28]:
```
import numpy as np
df['Annual Income (k$)']= np.where(df['Annual Income (k$)']>132,60,df['Annual␣
↪Income (k$)'])
```

[29]:
```
sns.boxplot(x=df['Annual Income (k$)'])
```

[29]: <AxesSubplot:xlabel='Annual Income (k$)'>



7. Check for Categorical columns and perform encoding.

```
df.info()
```

[30]:
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0  to  199 Data
columns  (total  5  columns):
 #    Column                  Non-Null Count    Dtype
---   -------                 ---------------   ------
 0    CustomerID              200 non-null      int64
 1    Gender                  200 non-null      object
 2    Age                     200 non-null      int64
 3    Annual Income (k$)      200 non-null      int64
 4    Spending Score (1-100)  200 non-null      int64
dtypes:  int64(4),  object(1)
memory  usage:  7.9+ KB
```

[31]:
```
df["Gender"].unique()
```

[31]: array(['Male', 'Female'], dtype=object)

encoding

[32]:
```
df["Gender"].replace({"Male":1,"Female":0},inplace=True)
df
```

[32]:

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)
0	1	1	19	15	39
1	2	1	21	15	81
2	3	0	20	16	6
3	4	0	23	16	77
4	5	0	31	17	40
..	...	...	...	...	...
195	196	0	35	120	79
196	197	0	45	126	28
197	198	1	32	126	74
198	199	1	32	60	18
199	200	1	30	60	83

[200 rows x 5 columns]

8. Scaling the data

[33]:
```
df.drop("CustomerID", axis=1, inplace = True)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0  to  199 Data
columns  (total  4  columns):
```

```
 #     Column                   Non-Null Count   Dtype
-----  ---------                --------------   ----------
 0     Gender                   200 non-null     object
 1     Age                      200 non-null     int64
 2     Annual    Income (k$)    200 non-null     int64
 3     Spending Score (1-100)   200 non-null     int64
dtypes: int64(3), object(1)
memory usage: 6.4+ KB
```

[34]:
```
df.isnull().sum()
```

[34]:
```
Gender                    0
Age                       0
Annual Income (k$)        0
Spending Score (1-100)    0
dtype: int64
```

[35]:
```
x = df.iloc[:,[2,3]].values
x
```

[35]:
```
array([[ 15,  39],
       [ 15,  81],
       [ 16,   6],
       [ 16,  77],
       [ 17,  40],
       [ 17,  76],
       [ 18,   6],
       [ 18,  94],
       [ 19,   3],
       [ 19,  72],
       [ 19,  14],
       [ 19,  99],
       [ 20,  15],
       [ 20,  77],
       [ 20,  13],
       [ 20,  79],
       [ 21,  35],
       [ 21,  66],
       [ 23,  29],
       [ 23,  98],
       [ 24,  35],
       [ 24,  73],
       [ 25,   5],
       [ 25,  73],
       [ 28,  14],
       [ 28,  82],
       [ 28,  32],
       [ 28,  61],
```

```
[ 29, 31],
[ 29, 87],
[ 30,  4],
[ 30, 73],
[ 33,  4],
[ 33, 92],
[ 33, 14],
[ 33, 81],
[ 34, 17],
[ 34, 73],
[ 37, 26],
[ 37, 75],
[ 38, 35],
[ 38, 92],
[ 39, 36],
[ 39, 61],
[ 39, 28],
[ 39, 65],
[ 40, 55],
[ 40, 47],
[ 40, 42],
[ 40, 42],
[ 42, 52],
[ 42, 60],
[ 43, 54],
[ 43, 60],
[ 43, 45],
[ 43, 41],
[ 44, 50],
[ 44, 46],
[ 46, 51],
[ 46, 46],
[ 46, 56],
[ 46, 55],
[ 47, 52],
[ 47, 59],
[ 48, 51],
[ 48, 59],
[ 48, 50],
[ 48, 48],
[ 48, 59],
[ 48, 47],
[ 49, 55],
[ 49, 42],
[ 50, 49],
[ 50, 56],
[ 54, 47],
```

[54, 54],
[54, 53],
[54, 48],
[54, 52],
[54, 42],
[54, 51],
[54, 55],
[54, 41],
[54, 44],
[54, 57],
[54, 46],
[57, 58],
[57, 55],
[58, 60],
[58, 46],
[59, 55],
[59, 41],
[60, 49],
[60, 40],
[60, 42],
[60, 52],
[60, 47],
[60, 50],
[61, 42],
[61, 49],
[62, 41],
[62, 48],
[62, 59],
[62, 55],
[62, 56],
[62, 42],
[63, 50],
[63, 46],
[63, 43],
[63, 48],
[63, 52],
[63, 54],
[64, 42],
[64, 46],
[65, 48],
[65, 50],
[65, 43],
[65, 59],
[67, 43],
[67, 57],
[67, 56],
[67, 40],

[69, 58],
[69, 91],
[70, 29],
[70, 77],
[71, 35],
[71, 95],
[71, 11],
[71, 75],
[71, 9],
[71, 75],
[72, 34],
[72, 71],
[73, 5],
[73, 88],
[73, 7],
[73, 73],
[74, 10],
[74, 72],
[75, 5],
[75, 93],
[76, 40],
[76, 87],
[77, 12],
[77, 97],
[77, 36],
[77, 74],
[78, 22],
[78, 90],
[78, 17],
[78, 88],
[78, 20],
[78, 76],
[78, 16],
[78, 89],
[78, 1],
[78, 78],
[78, 1],
[78, 73],
[79, 35],
[79, 83],
[81, 5],
[81, 93],
[85, 26],
[85, 75],
[86, 20],
[86, 95],
[87, 27],

```
       [ 87,  63],
       [ 87,  13],
       [ 87,  75],
       [ 87,  10],
       [ 87,  92],
       [ 88,  13],
       [ 88,  86],
       [ 88,  15],
       [ 88,  69],
       [ 93,  14],
       [ 93,  90],
       [ 97,  32],
       [ 97,  86],
       [ 98,  15],
       [ 98,  88],
       [ 99,  39],
       [ 99,  97],
       [101,  24],
       [101,  68],
       [103,  17],
       [103,  85],
       [103,  23],
       [103,  69],
       [113,   8],
       [113,  91],
       [120,  16],
       [120,  79],
       [126,  28],
       [126,  74],
       [137,  18],
       [137,  83]], dtype=int64)
```

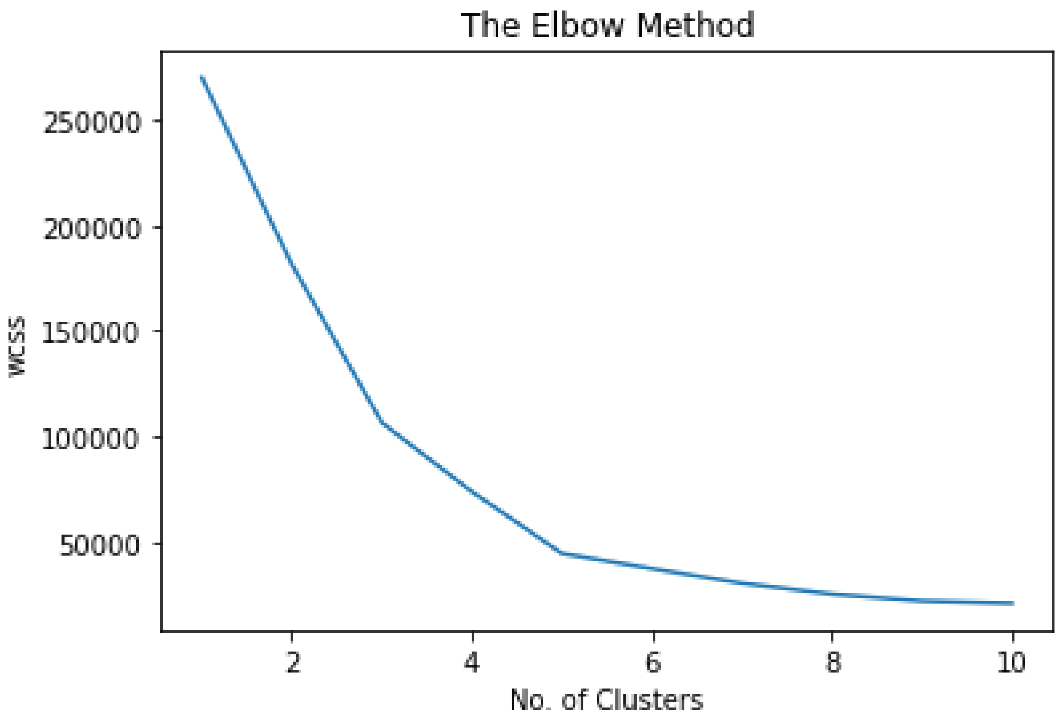
9. Perform any of the clustering algorithms

[36]:

```
from sklearn.cluster import KMeans

wcss = []
for i in range(1, 11):
    km = KMeans(n_clusters = i, init = 'k-means++', max_iter = 300, n_init = 10,␣
 ↪random_state = 0)
    km.fit(x)
    wcss.append(km.inertia_)

plt.plot(range(1, 11), wcss)
plt.title('The Elbow Method')
plt.xlabel('No. of Clusters')
plt.ylabel('wcss')
```

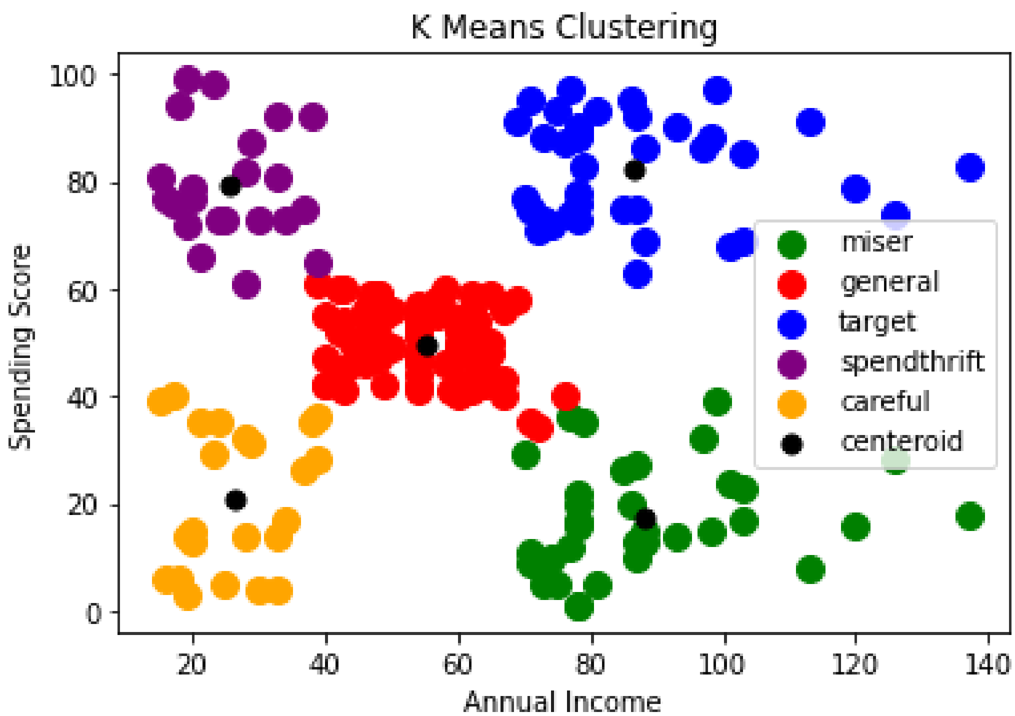
```
plt.show()
```

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\cluster_kmeans.py:1036: UserWarning: KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting theenvironment variable OMP_NUM_THREADS=1.
  warnings.warn(



[37]:
```
km = KMeans(n_clusters = 5, init = 'k-means++', max_iter = 300, n_init = 10,␣
↪random_state = 0)
y_means = km.fit_predict(x)

plt.scatter(x[y_means == 0, 0], x[y_means == 0, 1], s = 100, c = 'green', label␣
↪= 'miser')
plt.scatter(x[y_means == 1, 0], x[y_means == 1, 1], s = 100, c = 'red', label =␣
↪'general')
plt.scatter(x[y_means == 2, 0], x[y_means == 2, 1], s = 100, c = 'blue', label␣
↪= 'target')
plt.scatter(x[y_means == 3, 0], x[y_means == 3, 1], s = 100, c = 'purple',␣
↪label = 'spendthrift')
plt.scatter(x[y_means == 4, 0], x[y_means == 4, 1], s = 100, c = 'orange',␣
↪label = 'careful')
```

```
plt.scatter(km.cluster_centers_[:,0], km.cluster_centers_[:, 1], s = 50, c =␣
 ↪"black" , label = "centeroid")

plt.title("K Means Clustering")
plt.xlabel("Annual Income")
plt.ylabel("Spending Score")
plt.legend()
plt.show()
```



10. Add the cluster data with the primary dataset

[38]:
```
df["clust"]=mb
```

[39]:
```
df.head()
```

[39]:

	Gender	Age	Annual Income (k$)	Spending Score (1-100)	clust
0	Male	19	15	39	0
1	Male	21	15	81	0
2	Female	20	16	6	3
3	Female	23	16	77	1
4	Female	31	17	40	1

[40]:
```
df.tail()
```

[40]: Gender Age Annual Income (k$) Spending Score (1-100) clust195
Female 35 120 79 1

11. Split the data into dependent and independent variables

```
from sklearn.model_selection import train_test_split
```

[41]:
```
x = df.iloc[:, 1:5]
```

[42]:
```
y = df.iloc[:, -1]
y
```

1	21	15	81	0
2	20	16	6	3
3	23	16	77	1
4	31	17	40	1
..	...	...	...	...
195	35	120	79	1
196	45	126	28	3
197	32	126	74	0
198	32	137	18	2
199	30	137	83	0

[42]:

```
0       0
```

[43]:

```
1       0
2       3
3       1
4       1
        ..
195     1
196     3
197     0
198     2
199     0
Name: clust, Length: 200, dtype: int32
```

11.Build the Model

[44]:
```
from sklearn.linear_model import LinearRegression
```

[45]:
```
model=LinearRegression()
```

[46]:
```
model.fit(x_train,y_train)
```

112. Split the data into training and testing

```
from sklearn.model_selection import train_test_split
```

[47]:
```
X = df.iloc[:, 1:6]
X
```

[48]:

[48]:

	Age	Annual Income (k$)	Spending Score (1-100)	clust
0	19	15	39	0
1	21	15	81	0
2	20	16	6	3
3	23	16	77	1
4	31	17	40	1
..	...	...	...	...
195	35	120	79	1
196	45	126	28	3
197	32	126	74	0
198	32	137	18	2
199	30	137	83	0

[200 rows x 4 columns]

[49]:
```
y = df.iloc[:, -1]
y
```

[49]:
```
0      0
1      0
2      3
3      1
4      1
      ..
195    1
196    3
197    0
198    2
199    0
Name: clust, Length: 200, dtype: int32
```

[50]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.25,random_state
↪=0)
```

[51]:

```
print(X_train)
```

```
     Age  Annual  Income (k$)  Spending  Score  (1-100)  clust
71    47                   49                        42      3
124   23                   70                        29      1
184   41                   99                        39      3
97    27                   60                        50      1
149   34                   78                        90      0
..   ...                  ...                ...    ...
67    68                   48                        48      3
192   33                  113                         8      2
117   49                   65                        59      3
47    27                   40                        47      1
172   36                   87                        10      2

[150 rows x 4 columns]
```

[52]:

```
print(X_test)
```

```
     Age  Annual  Income  (k$)  Spending  Score  (1-100)  clust
18    52                    23                        29      2
170   40                    87                        13      2
107   54                    63                        46      2
98    48                    61                        42      2
177   27                    88                        69      0
182   46                    98                        15      2
5     22                    17                        76      1
146   48                    77                        36      2
12    58                    20                        15      3
152   44                    78                        20      3
61    19                    46                        55      0
125   31                    70                        77      1
180   37                    97                        32      3
154   47                    78                        16      3
80    57                    54                        51      2
7     23                    18                        94      1
33    18                    33                        92      0
130   47                    71                         9      2
37    30                    34                        73      1
74    59                    54                        47      2
183   29                    98                        88      1
145   28                    77                        97      0
45    24                    39                        65      1
159   30                    78                        73      1
60    70                    46                        56      2
123   39                    69                        91      0
179   35                    93                        90      0
185   30                    99                        97      0
```

```
122     40              69              58      1
44      49              39              28      3
16      35              21              35      3
55      47              43              41      2
150     43              78              17      2
111     19              63              54      1
22      46              25               5      3
189     36             103              85      1
129     38              71              75      0
4       31              17              40      1
83      46              54              44      3
106     66              63              50      3
134     20              73               5      2
66      43              48              50      3
26      45              28              32      3
113     19              64              46      0
168     36              87              27      3
63      54              47              59      3
8       64              19               3      2
75      26              54              54      0
118     51              67              43      3
143     32              76              87      1
```

[53]:
```
print(y_train)
```

```
71      3
124     1
184     3
97      1
149     0
       ..
67      3
192     2
117     3
47      1
172     2
Name: clust, Length: 150, dtype: int32
```

[54]:
```
print(y_test)
```

```
18      2
170     2
107     2
98      2
177     0
182     2
5       1
146     2
12      3
```

```
152    3
61     0
125    1
180    3
154    3
80     2
7      1
33     0
130    2
37     1
74     2
183    1
145    0
45     1
159    1
60     2
123    0
179    0
185    0
122    1
44     3
16     3
55     2
150    2
111    1
22     3
189    1
129    0
4      1
83     3
106    3
134    2
66     3
26     3
113    0
168    3
63     3
8      2
75     0
118    3
143    1
Name: clust, dtype: int32
```

13.Build the Model

[55]:
```python
from sklearn.linear_model import LinearRegression
```

[56]:
```
model=LinearRegression()
```

[57]:
```
model.fit(X_train,y_train)
```

[57]: LinearRegression()

14. Train the Model

[58]:
```
y_predict_train = model.predict(X_train)
y_predict_train
```

[58]:
```
array([ 3.00000000e+00,  1.00000000e+00,  3.00000000e+00,  1.00000000e+00,
       -1.35981248e-15,  3.00000000e+00,  2.00000000e+00,  3.00000000e+00,
        3.00000000e+00,  3.00000000e+00, -5.08338388e-16,  3.00000000e+00,
        1.00000000e+00,  1.00000000e+00,  1.00000000e+00,  2.00000000e+00,
        3.00000000e+00,  1.00000000e+00,  3.78230988e-15,  3.00000000e+00,
        1.00000000e+00,  3.00000000e+00, -1.03981941e-15,  3.00000000e+00,
        2.00000000e+00,  3.00000000e+00,  2.00000000e+00,  2.00000000e+00,
        6.18236322e-15, -8.17774807e-16, -5.92319283e-15,  3.00000000e+00,
        2.00000000e+00,  1.00000000e+00,  2.00000000e+00,  3.00000000e+00,
        3.00000000e+00,  2.00000000e+00,  3.00000000e+00, -9.83833688e-15,
        1.00000000e+00,  2.00000000e+00,  3.00000000e+00,  3.00000000e+00,
        3.00000000e+00,  2.00000000e+00,  1.00000000e+00,  2.00000000e+00,
        3.00000000e+00,  3.00000000e+00,  2.00000000e+00,  1.00000000e+00,
        3.00000000e+00,  7.41004011e-15,  1.00000000e+00,  2.00000000e+00,
        3.78930723e-16,  2.00000000e+00,  1.00000000e+00,  5.15288973e-15,
        1.00000000e+00,  1.00000000e+00,  1.00000000e+00,  2.00000000e+00,
        1.00000000e+00,  1.00000000e+00,  6.93337166e-15,  3.00000000e+00,
        1.00000000e+00,  2.00000000e+00,  7.74883999e-15,  1.00000000e+00,
        2.00000000e+00,  1.00000000e+00,  1.00000000e+00,  3.00000000e+00,
        1.00000000e+00,  3.00000000e+00,  2.00000000e+00,  2.00000000e+00,
        3.00000000e+00,  2.46954765e-15,  1.00000000e+00,  3.00000000e+00,
        1.00000000e+00, -3.39743545e-15,  2.00000000e+00,  1.17664588e-16,
       -1.14612341e-14,  1.00000000e+00,  3.00000000e+00,  2.00000000e+00,
        1.00000000e+00,  2.00000000e+00,  2.44913294e-15,  8.61660370e-15,
       -6.14537022e-16,  2.00000000e+00,  2.00000000e+00,  1.00000000e+00,
        1.00000000e+00, -2.76227299e-15,  7.18065464e-15,  3.00000000e+00,
        1.00000000e+00,  2.00000000e+00,  1.00000000e+00,  2.00000000e+00,
        3.00000000e+00,  3.00000000e+00,  1.00000000e+00, -1.31325008e-15,
        1.00000000e+00,  2.00000000e+00,  6.71878339e-17,  2.00000000e+00,
        2.00000000e+00,  1.00000000e+00, -3.64884352e-15,  1.00000000e+00,
        1.00000000e+00,  4.74717047e-17,  2.00000000e+00,  3.00000000e+00,
        1.00000000e+00,  3.00000000e+00,  1.00000000e+00,  3.00000000e+00,
        2.00000000e+00,  6.64274206e-15,  1.00000000e+00,  1.74720410e-15,
        3.00000000e+00,  3.00000000e+00,  3.00000000e+00,  1.00000000e+00,
        1.00000000e+00,  3.00000000e+00,  1.00000000e+00,  2.00000000e+00,
        1.00000000e+00,  3.00000000e+00,  6.97741685e-15,  1.00000000e+00,
```

```
1.69959961e-16,  3.00000000e+00,  2.00000000e+00,  3.00000000e+00,
1.00000000e+00,  2.00000000e+00])
```

15. Test the Model

[59]:
```python
y_predict = model.predict(X_test)
y_predict
```

[59]:
```
array([ 2.00000000e+00,  2.00000000e+00,  2.00000000e+00,  2.00000000e+00,
       -3.75091707e-15,  2.00000000e+00,  1.00000000e+00,  2.00000000e+00,
        3.00000000e+00,  3.00000000e+00,  2.69159225e-15,  1.00000000e+00,
        3.00000000e+00,  3.00000000e+00,  2.00000000e+00,  1.00000000e+00,
        5.86306874e-15,  2.00000000e+00,  1.00000000e+00,  2.00000000e+00,
        1.00000000e+00, -1.12147825e-15,  1.00000000e+00,  1.00000000e+00,
        2.00000000e+00,  2.68817729e-16, -3.83739957e-15, -4.74438481e-15,
        1.00000000e+00,  3.00000000e+00,  3.00000000e+00,  2.00000000e+00,
        2.00000000e+00,  1.00000000e+00,  3.00000000e+00,  1.00000000e+00,
       -5.28753097e-16,  1.00000000e+00,  3.00000000e+00,  3.00000000e+00,
        2.00000000e+00,  3.00000000e+00,  3.00000000e+00, -5.55810093e-16,
        3.00000000e+00,  3.00000000e+00,  2.00000000e+00,  1.47447202e-15,
        3.00000000e+00,  1.00000000e+00])
```

16. Measure the performance using Evaluation Metrics.

[60]:
```python
from sklearn.metrics import mean_squared_error
import math
print(mean_squared_error(y_test, y_predict))
print(math.sqrt(mean_squared_error(y_test, y_predict)))
```

```
1.7123447122139227e-29
4.138048709493307e-15
```