Problem Statement :Customer Segmentation Analysis

1.Download the dataset

```
import pandas as pd
import numpy as np
import seaborn as sns
from matplotlib import pyplot as plt
from sklearn.preprocessing import scale
import warnings
warnings.filterwarnings('ignore')
```

2.load the dataset into the tool

```
data=pd.read_csv("Mall_Customers.csv")
```

```
data.head()
```

	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

```
data.shape
```

(200, 5)

```
data.size
```

Out[49]:
1000
In [50]:

```
data.info()
```

```
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
```

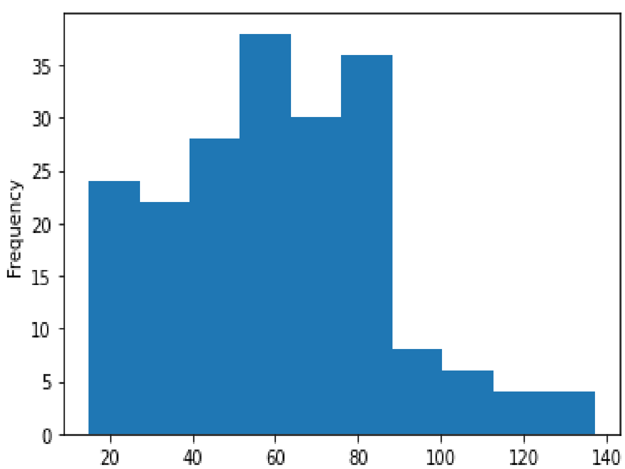
Perform Below Visulizations

Univariate Analysis

In [51]:

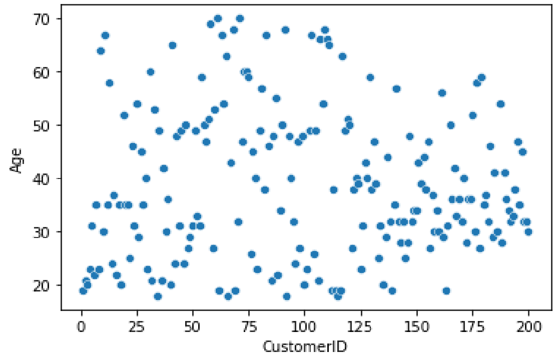
```
data.hist(figsize=(20,10), grid=False, layout=(2,4),bins=30)
plt.show()
```

```
data["Annual Income (k$)"].plot(kind='hist')
```



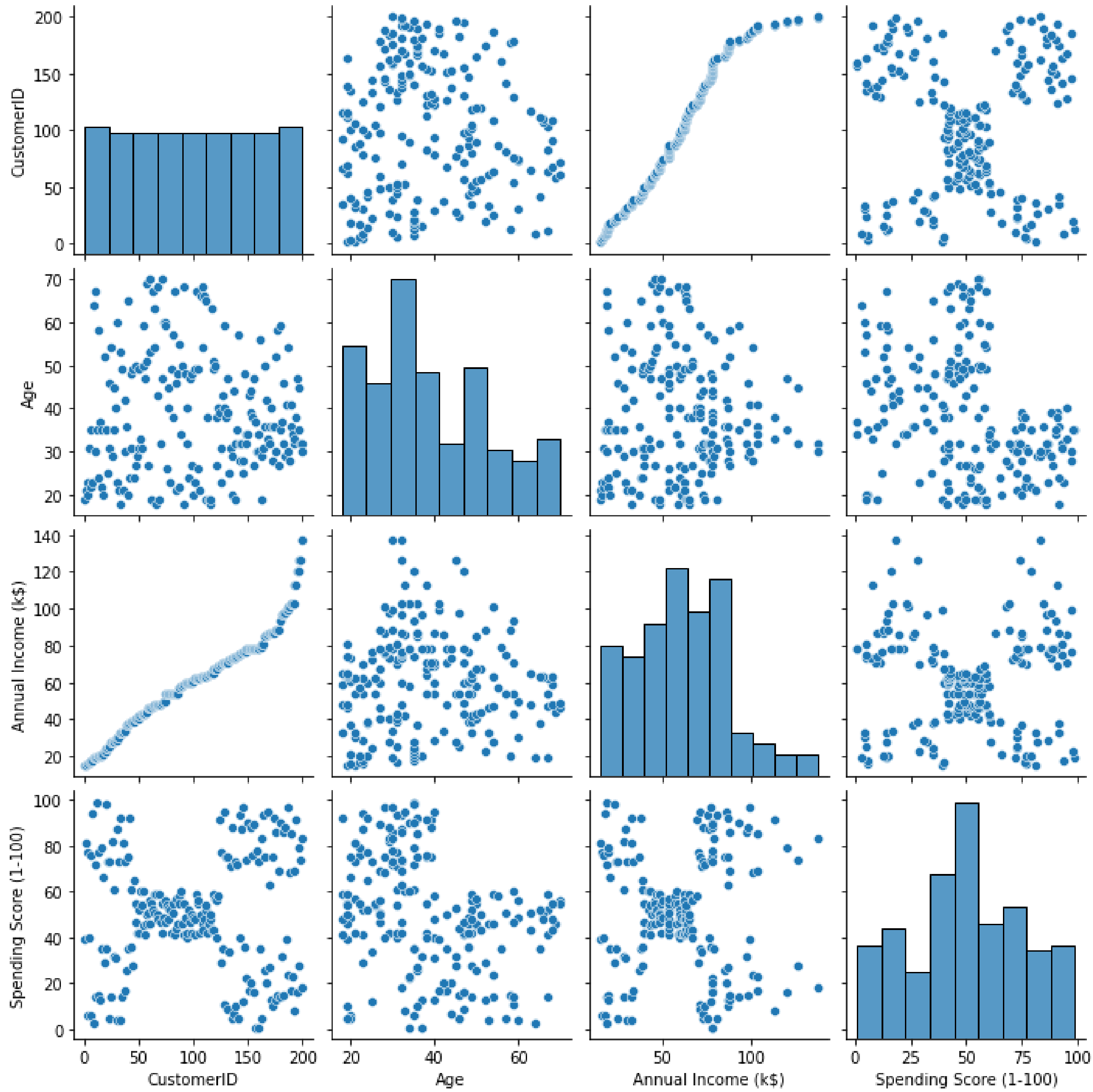
Bi-variate Analysis

```
sns.scatterplot(data.CustomerID,data.Age)
```



Multi -Variate Analysis

sns.pairplot(data)



4.Perform descriptive statistics on the dataset

data.describe()

	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
max	200.000000	70.000000	137.000000	99.000000

5. Check for Missing values and deal with them

data.isna().sum()

6. Find the outliers and replace them outliers.

In [56]:
data.skew()

Out[56]:
CustomerID 0.000000
Age 0.485569
Annual Income (k$) 0.321843
Spending Score (1-100) -0.047220
dtype: float64
In [57]:

```
sns.boxplot(x=data['Age'],data=data)
```

7.Check for Categorical columns and perform encoding

In [61]:

```
data.info
```

Out[61]:

In [62]:

```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
data['Gender']=le.fit_transform(data['Gender'])
data.head()
```

Out[62]:

	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

In [63]:

```
data["Gender"].unique()
```

Out[63]:

```
array([1, 0])
```

8. Scaling the Data

In [64]:

```
x=data.drop(columns=['Gender','Age'])
print(x)
```

```
    CustomerID  Annual Income (k$)  Spending Score (1-100)
0            1                  15                      39
1            2                  15                      81
```

```
2           3               16                  6
3           4               16                 77
4           5               17                 40
..        ...             ...               ...
195       196             120                79
196       197             126                28
197       198             126                74
198       199             137                18
199       200             137                83

[200 rows x 3 columns]
```
In [65]:
```
S=scale(x)
print(S)
```
```
[[-1.7234121  -1.73899919 -0.43480148]
 [-1.70609137 -1.73899919  1.19570407]
 [-1.68877065 -1.70082976 -1.71591298]
 [-1.67144992 -1.70082976  1.04041783]
 [-1.6541292  -1.66266033 -0.39597992]
 [-1.63680847 -1.66266033  1.00159627]
 [-1.61948775 -1.62449091 -1.71591298]
 [-1.60216702 -1.62449091  1.70038436]
[-1.5848463  -1.58632148 -1.83237767]
 [-1.56752558 -1.58632148  0.84631002]
 [-1.55020485 -1.58632148 -1.4053405 ]
 [-1.53288413 -1.58632148  1.89449216]
 [-1.5155634  -1.54815205 -1.36651894]
 [-1.49824268 -1.54815205  1.04041783]
 [-1.48092195 -1.54815205 -1.44416206]
 [-1.46360123 -1.54815205  1.11806095]
 [-1.4462805  -1.50998262 -0.59008772]
 [-1.42895978 -1.50998262  0.61338066]
 [-1.41163905 -1.43364376 -0.82301709]
 [-1.39431833 -1.43364376  1.8556706 ]
 [-1.3769976  -1.39547433 -0.59008772]
 [-1.35967688 -1.39547433  0.88513158]
 [-1.34235616 -1.3573049  -1.75473454]
 [-1.32503543 -1.3573049   0.88513158]
 [-1.30771471 -1.24279661 -1.4053405 ]
 [-1.29039398 -1.24279661  1.23452563]
 [-1.27307326 -1.24279661 -0.7065524 ]
 [-1.25575253 -1.24279661  0.41927286]
 [-1.23843181 -1.20462718 -0.74537397]
 [-1.22111108 -1.20462718  1.42863343]
 [-1.20379036 -1.16645776 -1.7935561 ]
 [-1.18646963 -1.16645776  0.88513158]
```

```
[-1.16914891 -1.05194947 -1.7935561 ]
[-1.15182818 -1.05194947  1.62274124]
[-1.13450746 -1.05194947 -1.4053405 ]
[-1.11718674 -1.05194947  1.19570407]
[-1.09986601 -1.01378004 -1.28887582]
[-1.08254529 -1.01378004  0.88513158]
[-1.06522456 -0.89927175 -0.93948177]
[-1.04790384 -0.89927175  0.96277471]
[-1.03058311 -0.86110232 -0.59008772]
[-1.01326239 -0.86110232  1.62274124]
[-0.99594166 -0.82293289 -0.55126616]
[-0.97862094 -0.82293289  0.41927286]
[-0.96130021 -0.82293289 -0.86183865]
[-0.94397949 -0.82293289  0.5745591 ]
[-0.92665877 -0.78476346  0.18634349]
[-0.90933804 -0.78476346 -0.12422899]
[-0.89201732 -0.78476346 -0.3183368 ]
[-0.87469659 -0.78476346 -0.3183368 ]
[-0.85737587 -0.70842461  0.06987881]
[-0.84005514 -0.70842461  0.38045129]
[-0.82273442 -0.67025518  0.14752193]
[-0.80541369 -0.67025518  0.38045129]
[-0.78809297 -0.67025518 -0.20187212]
[-0.77077224 -0.67025518 -0.35715836]
[-0.75345152 -0.63208575 -0.00776431]
[-0.73613079 -0.63208575 -0.16305055]
[-0.71881007 -0.55574689  0.03105725]
[-0.70148935 -0.55574689 -0.16305055]
[-0.68416862 -0.55574689  0.22516505]
[-0.6668479  -0.55574689  0.18634349]
[-0.64952717 -0.51757746  0.06987881]
[-0.63220645 -0.51757746  0.34162973]
[-0.61488572 -0.47940803  0.03105725]
[-0.597565   -0.47940803  0.34162973]
[-0.58024427 -0.47940803 -0.00776431]
[-0.56292355 -0.47940803 -0.08540743]
[-0.54560282 -0.47940803  0.34162973]
[-0.5282821  -0.47940803 -0.12422899]
[-0.51096138 -0.4412386   0.18634349]
[-0.49364065 -0.4412386  -0.3183368 ]
[-0.47631993 -0.40306917 -0.04658587]
[-0.4589992  -0.40306917  0.22516505]
[-0.44167848 -0.25039146 -0.12422899]
[-0.42435775 -0.25039146  0.14752193]
[-0.40703703 -0.25039146  0.10870037]
[-0.3897163  -0.25039146 -0.08540743]
```

```
[-0.37239558 -0.25039146  0.06987881]
[-0.35507485 -0.25039146 -0.3183368 ]
[-0.33775413 -0.25039146  0.03105725]
[-0.3204334  -0.25039146  0.18634349]
[-0.30311268 -0.25039146 -0.35715836]
[-0.28579196 -0.25039146 -0.24069368]
[-0.26847123 -0.25039146  0.26398661]
[-0.25115051 -0.25039146 -0.16305055]
[-0.23382978 -0.13588317  0.30280817]
[-0.21650906 -0.13588317  0.18634349]
[-0.19918833 -0.09771374  0.38045129]
[-0.18186761 -0.09771374 -0.16305055]
[-0.16454688 -0.05954431  0.18634349]
[-0.14722616 -0.05954431 -0.35715836]
[-0.12990543 -0.02137488 -0.04658587]
[-0.11258471 -0.02137488 -0.39597992]
[-0.09526399 -0.02137488 -0.3183368 ]
[-0.07794326 -0.02137488  0.06987881]
[-0.06062254 -0.02137488 -0.12422899]
[-0.04330181 -0.02137488 -0.00776431]
[-0.02598109  0.01679455 -0.3183368 ]
[-0.00866036  0.01679455 -0.04658587]
[ 0.00866036  0.05496398 -0.35715836]
[ 0.02598109  0.05496398 -0.08540743]
[ 0.04330181  0.05496398  0.34162973]
[ 0.06062254  0.05496398  0.18634349]
[ 0.07794326  0.05496398  0.22516505]
[ 0.09526399  0.05496398 -0.3183368 ]
[ 0.11258471  0.09313341 -0.00776431]
[ 0.12990543  0.09313341 -0.16305055]
[ 0.14722616  0.09313341 -0.27951524]
[ 0.16454688  0.09313341 -0.08540743]
[ 0.18186761  0.09313341  0.06987881]
[ 0.19918833  0.09313341  0.14752193]
[ 0.21650906  0.13130284 -0.3183368 ]
[ 0.23382978  0.13130284 -0.16305055]
[ 0.25115051  0.16947227 -0.08540743]
[ 0.26847123  0.16947227 -0.00776431]
[ 0.28579196  0.16947227 -0.27951524]
[ 0.30311268  0.16947227  0.34162973]
[ 0.3204334   0.24581112 -0.27951524]
[ 0.33775413  0.24581112  0.26398661]
[ 0.35507485  0.24581112  0.22516505]
[ 0.37239558  0.24581112 -0.39597992]
[ 0.3897163   0.32214998  0.30280817]
[ 0.40703703  0.32214998  1.58391968]
```

[0.42435775 0.36031941 -0.82301709]
[0.44167848 0.36031941 1.04041783]
[0.4589992   0.39848884 -0.59008772]
[0.47631993 0.39848884 1.73920592]
[0.49364065 0.39848884 -1.52180518]
[0.51096138 0.39848884 0.96277471]
[0.5282821   0.39848884 -1.5994483]
[0.54560282 0.39848884 0.96277471]
[0.56292355 0.43665827 -0.62890928]
[0.58024427 0.43665827 0.80748846]
[0.597565   0.4748277 -1.75473454]
[0.61488572 0.4748277   1.46745499]
[0.63220645 0.4748277 -1.67709142]
[0.64952717 0.4748277   0.88513158]
[0.6668479   0.51299713 -1.56062674]
[0.68416862 0.51299713 0.84631002]
[0.70148935 0.55116656 -1.75473454]
[0.71881007 0.55116656 1.6615628]
[0.73613079 0.58933599 -0.39597992]
[0.75345152 0.58933599 1.42863343]
[0.77077224 0.62750542 -1.48298362]
[0.78809297 0.62750542 1.81684904]
[0.80541369 0.62750542 -0.55126616]
[0.82273442 0.62750542 0.92395314]
[0.84005514 0.66567484 -1.09476801]
[0.85737587 0.66567484 1.54509812]
[0.87469659 0.66567484 -1.28887582]
[0.89201732 0.66567484 1.46745499]
[0.90933804 0.66567484 -1.17241113]
[0.92665877 0.66567484 1.00159627]
[0.94397949 0.66567484 -1.32769738]
[0.96130021 0.66567484 1.50627656]
[0.97862094 0.66567484 -1.91002079]
[0.99594166 0.66567484 1.07923939]
[1.01326239 0.66567484 -1.91002079]
[1.03058311 0.66567484 0.88513158]
[1.04790384 0.70384427 -0.59008772]
[1.06522456 0.70384427 1.27334719]
[1.08254529 0.78018313 -1.75473454]
[1.09986601 0.78018313 1.6615628]
[1.11718674 0.93286085 -0.93948177]
[1.13450746 0.93286085 0.96277471]
[1.18646963 1.00919971 -0.90066021]
[1.20379036 1.00919971 0.49691598]
[1.22111108 1.00919971 -1.44416206]
[1.23843181 1.00919971 0.96277471]

```
 [ 1.25575253  1.00919971 -1.56062674]
 [ 1.27307326  1.00919971  1.62274124]
 [ 1.29039398  1.04736914 -1.44416206]
 [ 1.30771471  1.04736914  1.38981187]
 [ 1.32503543  1.04736914 -1.36651894]
 [ 1.34235616  1.04736914  0.72984534]
 [ 1.35967688  1.23821628 -1.4053405 ]
 [ 1.3769976   1.23821628  1.54509812]
 [ 1.39431833  1.390894   -0.7065524 ]
 [ 1.41163905  1.390894    1.38981187]
 [ 1.42895978  1.42906343 -1.36651894]
 [ 1.4462805   1.42906343  1.46745499]
 [ 1.46360123  1.46723286 -0.43480148]
 [ 1.48092195  1.46723286  1.81684904]
 [ 1.49824268  1.54357172 -1.01712489]
 [ 1.5155634   1.54357172  0.69102378]
 [ 1.53288413  1.61991057 -1.28887582]
 [ 1.55020485  1.61991057  1.35099031]
 [ 1.56752558  1.61991057 -1.05594645]
 [ 1.5848463   1.61991057  0.72984534]
 [ 1.60216702  2.00160487 -1.63826986]
 [ 1.61948775  2.00160487  1.58391968]
 [ 1.63680847  2.26879087 -1.32769738]
 [ 1.6541292   2.26879087  1.11806095]
 [ 1.67144992  2.49780745 -0.86183865]
 [ 1.68877065  2.49780745  0.92395314]
 [ 1.70609137  2.91767117 -1.25005425]
 [ 1.7234121   2.91767117  1.27334719]]
```

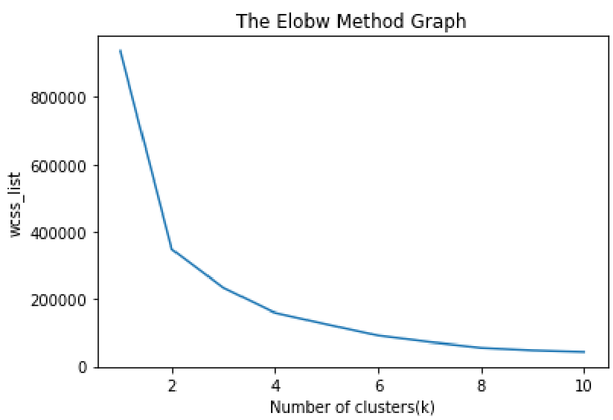
9.Perform any of the clustering algorithms

In [66]:

```
#finding optimal number of clusters using the elbow method
from sklearn.cluster import KMeans
wcss_list= []  #Initializing the list for the values of WCSS

#Using for loop for iterations from 1 to 10.
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state= 42)
    kmeans.fit(x)
    wcss_list.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss_list)
plt.title('The Elobw Method Graph')
plt.xlabel('Number of clusters(k)')
```

```
plt.ylabel('wcss_list')
plt.show()
```



10. Add the cluster data with the primary dataset

In [67]:
```
kmeans = KMeans(n_clusters=5, init='k-means++', random_state= 42)
clus= kmeans.fit_predict(S)
```

In [68]:
```
clus
```

Out[68]:
```
array([2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3,
       2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3,
       2, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 1, 4, 1, 4, 1, 4, 1, 4,
       1, 4, 1, 4, 1, 4, 1, 4, 1, 4, 1, 4, 1, 4, 1, 4, 1, 4, 1, 4, 1, 4,
       1, 4, 1, 4, 1, 4, 1, 4, 1, 4, 1, 4, 1, 4, 1, 4, 1, 4, 1, 4, 1, 4,
       1, 4, 1, 4, 1, 4, 1, 4, 1, 4, 1, 4, 1, 4, 1, 4, 1, 4, 1, 4, 1, 4,
       1, 4], dtype=int32)
```

11. Split the data into dependent and independent variables

In [69]:
```
x=data.drop(columns=['Annual Income (k$)'],axis=1)
print(x)
```

```
     CustomerID  Gender  Age  Spending Score (1-100)
0             1       1   19                      39
1             2       1   21                      81
2             3       0   20                       6
3             4       0   23                      77
4             5       0   31                      40
..          ...     ...  ...                     ...
195         196       0   35                      79
196         197       0   45                      28
197         198       1   32                      74
198         199       1   32                      18
199         200       1   30                      83

[200 rows x 4 columns]
```

In [70]:

```
y=data['Annual Income (k$)']
y
```

Out[70]:

```
0       15
1       15
2       16
3       16
4       17
      ...
195    120
196    126
197    126
198    137
199    137
Name: Annual Income (k$), Length: 200, dtype: int64
```

12. Split the data into training and testing

In [71]:

```
from sklearn.model_selection import train_test_split
x_train, x_test,y_train,y_test = train_test_split(x,y, test_size = 0.3,random_state=1)
```

In [72]:

```
x_train
```

Out[72]:

	CustomerID	**Gender**	**Age**	**Spending Score (1-100)**
116	117	0	63	43

	CustomerID	Gender	Age	Spending Score (1-100)
67	68	0	68	48
78	79	0	23	52
42	43	1	48	36
17	18	1	20	66
...	...	...	...	...
133	134	0	31	71
137	138	1	32	73
72	73	0	60	49
140	141	0	57	5
37	38	0	30	73

140 rows × 4 columns

x_train.shape

Out[73]:
(140, 4)
In [74]:
x_test

Out[74]:

	CustomerID	Gender	Age	Spending Score (1-100)
58	59	0	27	51
40	41	0	65	35
34	35	0	49	14
102	103	1	67	59
184	185	0	41	39
198	199	1	32	18
95	96	1	24	52
4	5	0	31	40
29	30	0	23	87
168	169	0	36	27
171	172	1	28	75
18	19	1	52	29
11	12	0	35	99
89	90	0	50	46
110	111	1	65	52

	CustomerID	Gender	Age	Spending Score (1-100)
118	119	0	51	43
159	160	0	30	73
35	36	0	21	81
136	137	0	44	7
59	60	1	53	46
51	52	1	33	60
16	17	0	35	35
44	45	0	49	28
94	95	0	32	42
31	32	0	21	73
162	163	1	19	5
38	39	0	36	26
28	29	0	40	31
193	194	0	38	91
27	28	1	35	61
47	48	0	27	47

	CustomerID	Gender	Age	Spending Score (1-100)
165	166	0	36	75
194	195	0	47	16
177	178	1	27	69
176	177	1	58	15
97	98	0	27	50
174	175	0	52	13
73	74	0	60	56
69	70	0	32	47
172	173	1	36	10
108	109	1	68	43
107	108	1	54	46
189	190	0	36	85
14	15	1	37	13
56	57	0	51	50
19	20	0	35	98
114	115	0	18	48

	CustomerID	Gender	Age	Spending Score (1-100)
39	40	0	20	75
185	186	1	30	97
124	125	0	23	29
98	99	1	48	42
123	124	1	39	91
119	120	0	50	57
53	54	1	59	60
33	34	1	18	92
179	180	1	35	90
181	182	0	32	86
106	107	0	66	50
199	200	1	30	83
138	139	1	19	10

```
x_test.shape
```

Out[75]:
(60, 4)
In [76]:

```
y_train
```

Out[76]:
116   65

```
67     48
78     54
42     39
17     21
       ..
133    72
137    73
72     50
140    75
37     34
Name: Annual Income (k$), Length: 140, dtype: int64
```

13.Build the model

In [77]:
```
from sklearn.linear_model import LinearRegression
LR = LinearRegression()
```

14.Train the Model

In [78]:
```
LR.fit(x_train,y_train)
```

Out[78]:
```
LinearRegression()
```

15.Test the model

In [79]:
```
pred=LR.predict(x_test)
```

In [80]:
```
pred
```

Out[80]:
```
array([ 41.79651469,  35.44897396,  32.32182941,  62.15230947,
        97.15499   , 102.74527464,  57.52904542,  18.50596884,
        28.90050195,  90.05616474,  90.63951146,  25.17877999,
        21.47607213,  56.15450717,  65.58284431,  68.81365504,
        85.74449988,  31.45756756,  76.51559556,  42.98039276,
        38.70178627,  23.89238204,  36.61730406,  57.67164216,
        29.74845621,  86.65460588,  33.53032334,  29.31235764,
       100.75984295,  28.3364555 ,  37.02836966,  88.57006476,
       101.81449573,  93.23392219,  94.16104415,  58.75918464,
        93.31570423,  49.53263905,  46.78164703,  91.618992  ,
        64.85923756,  63.89021447,  98.96847593,  22.93975353,
        41.82689378,  24.95860094,  65.82297944,  33.18229176,
        96.7187877 ,  70.4300092 ,  59.76768524,  70.1173078 ,
```

```
       69.1581952 ,  40.54244593,  30.19338393,  94.32293272,
       95.33656664,  64.12923371, 102.85955135,  76.19945402])
pred.astype(int)
```

Out[81]:

```
array([ 41,  35,  32,  62,  97, 102,  57,  18,  28,  90,  90,  25,  21,
        56,  65,  68,  85,  31,  76,  42,  38,  23,  36,  57,  29,  86,
        33,  29, 100,  28,  37,  88, 101,  93,  94,  58,  93,  49,  46,
        91,  64,  63,  98,  22,  41,  24,  65,  33,  96,  70,  59,  70,
        69,  40,  30,  94,  95,  64, 102,  76])
```

In [82]:

```
y_test
```

Out[82]:

```
58      46
40      38
34      33
102     62
184     99
198    137
95      60
4      17
29      29
168     87
171     87
18      23
11      19
89      58
110     63
118     67
159     78
35      33
136     73
59      46
51      42
16      21
44      39
94      60
31      30
162     81
38      37
28      29
193    113
27      28
47      40
165     85
194    120
177     88
```

```
176     88
97      60
174     88
73      50
69      48
172     87
108     63
107     63
189    103
14      20
56      44
19      23
114     65
39      37
185     99
124     70
98      61
123     69
119     67
53      43
33      33
179     93
181     97
106     63
199    137
138     74
Name: Annual Income (k$), dtype: int64
```

16.Measure the performance using Evaluation Metrics.

In [83]:

```python
from sklearn.metrics import r2_score
score=r2_score(pred,y_test)
```

In [84]:

```python
score
```

Out[84]:

```
0.9234274149757858
```