Assignment -4

Python Programming

Assignment Date	21 October 2022
Student Name	HARAN PRANAV.B.S
Student Roll Number	311419205011
Maximum Marks	2 Marks

Customer Segmentation Analysis

In [14]:

```
## import required libraries

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

In [15]:

```
## loading the dataset

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

Out[15]:

	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

In [16]:

```
df.shape
```

Out[16]: (200, 5)

In [17]:

```
df.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
```

In [10]:

```
df.isnull().any()
```

Out[10]:

```
CustomerID                False
Gender                    False
Age                       False
Annual Income (k$)        False
Spending Score (1-100)    False
dtype: bool
```

In [18]:

```
df.describe()
```

Out[18]:

	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

Univariate Analysis

In [10]:

```
df['Age'].mean()
```

Out[10]: 38.85

In [11]:

```
df['Age'].median()
```

Out[11]: 36.0

In [12]:

```
df['Age'].std()
```

Out[12]: 13.969007331558883

In [13]:

```
df['Annual Income (k$)'].value_counts()
```

Out[13]:

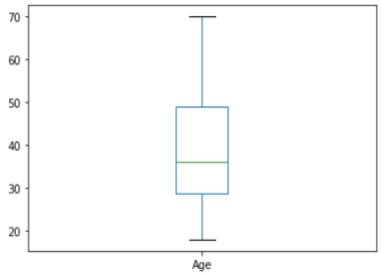
```
54     12
78     12
48      6
71      6
63      6
       ..
58      2
59      2
16      2
64      2
137     2
Name: Annual Income (k$), Length: 64, dtype: int64
```

Visualization

In [14]:

```
df.boxplot(column=['Age'], grid=False)
```

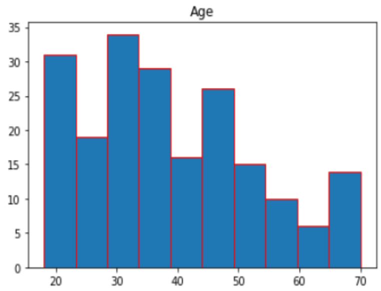
Out[14]:



In [15]:

```
df.hist(column='Age', grid=False, edgecolor='Red')
```

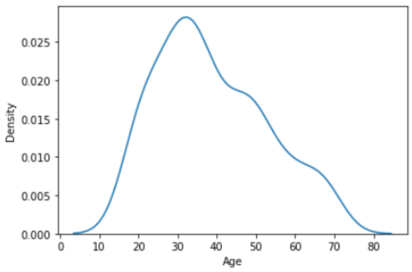
Out[15]: array([[]], dtype=object)



In [16]:

```
sns.kdeplot(df['Age'])
```

Out[16]:

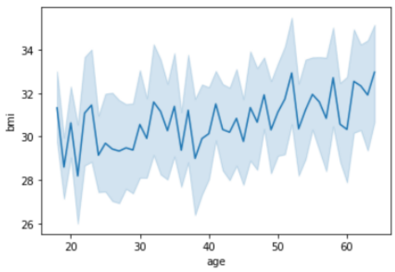


In [18]:

```
sns.lineplot(df.age,df.bmi)
```

```
C:\Users\Saumya\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variables as keyword args: 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[18]:



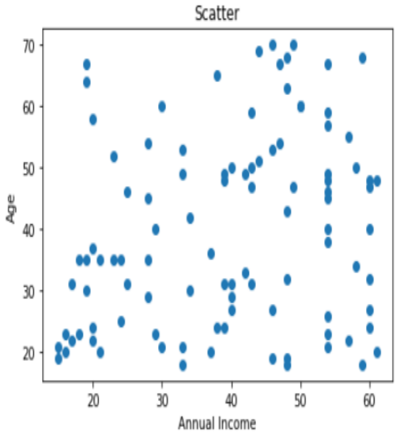
Bi - Variate Analysis

1. Scatterplots

In [17]:

```
plt.scatter(x=df["Annual Income (k$)"].head(100), y=df.Age.head(100))
plt.title('Scatter')
plt.xlabel('Annual Income')
plt.ylabel('Age')
```

Out[17]: Text(0, 0.5, 'Age')



2. Correlation Coefficients

In [19]:

```
df.corr()
```

Out[19]:

	CustomerID	Age	Annual Income (k$)	Spending Score (1-100)
CustomerID	1.000000	-0.026763	0.977548	0.013835
Age	-0.026763	1.000000	-0.012398	-0.327227
Annual Income (k$)	0.977548	-0.012398	1.000000	0.009903
Spending Score (1-100)	0.013835	-0.327227	0.009903	1.000000

In [19]:
```
y = df['Annual Income (k$)']
x = df['Spending Score (1-100)']
x = sm.add_constant(x)
model = sm.OLS(y,x).fit()
model.summary()
```

Out[19]: OLS Regression Results

Dep. Variable:	Annual Income (k$)	**R-squared:**	0.000
Model:	OLS	**Adj. R-squared:**	-0.005
Method:	Least Squares	**F-statistic:**	0.01942
Date:	Sat, 29 Oct 2022	**Prob (F-statistic):**	0.889
Time:	10:45:38	**Log-Likelihood:**	-936.92
No. Observations:	200	**AIC:**	1878.
Df Residuals:	198	**BIC:**	1884.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	60.0544	4.078	14.726	0.000	52.012	68.097
Spending Score (1-100)	0.0101	0.072	0.139	0.889	-0.132	0.153

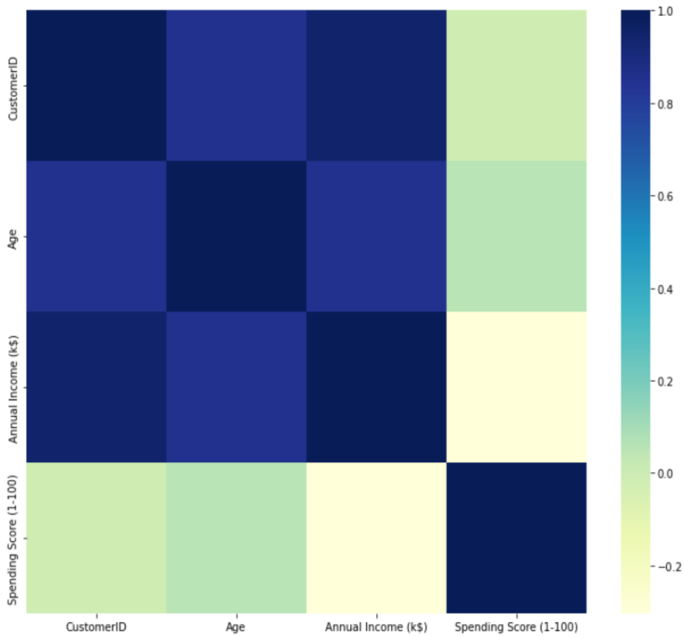
Omnibus:	3.510	**Durbin-Watson:**	0.005
Prob(Omnibus):	0.173	**Jarque-Bera (JB):**	3.531
Skew:	0.319	**Prob(JB):**	0.171
Kurtosis:	2.875	**Cond. No.**	124.

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

Multi - Variate Analysis

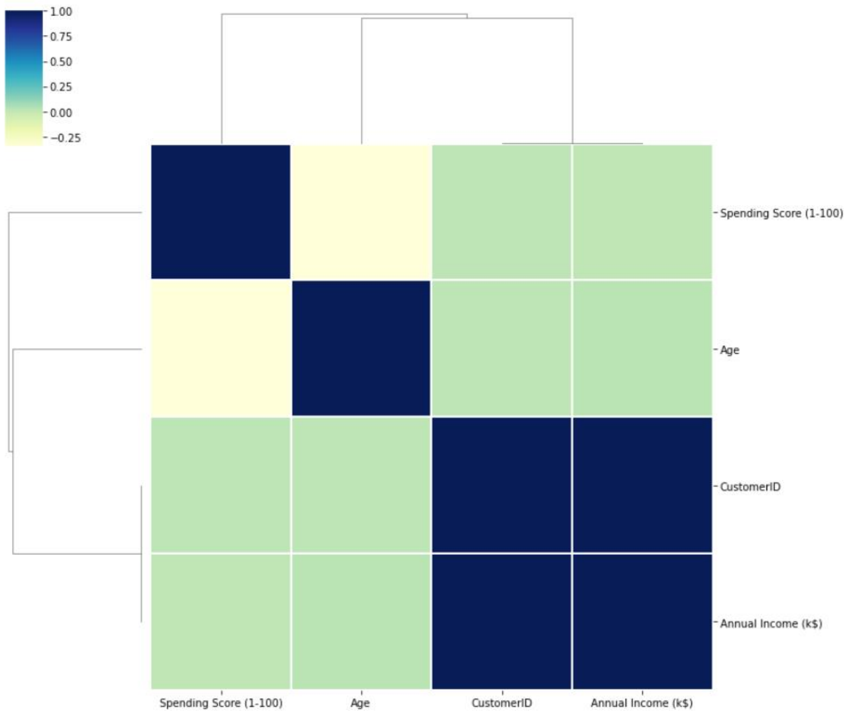
In [20]:
```
f = plt.subplots(figsize=(12,10))
sns.heatmap(df.head().corr(), cmap="YlGnBu")
```

Out[20]:



In [21]:
```
cormat = df.corr(method='spearman')
cg = sns.clustermap(cormat, cmap="YlGnBu", linewidths=0.1);
```

Out[21]:



4. Perform descriptive statistics on the dataset.

In [22]:

```
df.shape
```

Out[22]: (200, 5)

In [23]:

```
df.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
```

In [24]:

```
df.describe()
```

Out[24]:

	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

In [26]:
```
df.head()
```

Out[26]:

	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

In [27]:
```
df.tail()
```

Out[27]:

	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

In [28]:
```
df["Annual Income (k$)"].mean()
```

Out[28]: 60.56

In [29]:
```
df["Annual Income (k$)"].median()
```

Out[29]: 61.5

In [30]:
```
df["Annual Income (k$)"].mode()
```

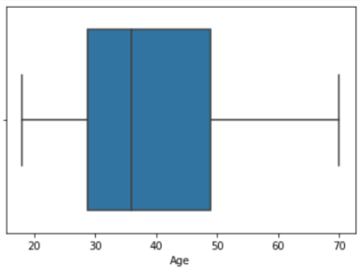
Out[30]:
```
0    54
1    78
Name: Annual Income (k$), dtype: int64
```

In [31]:
```
df["Annual Income (k$)"].var()
```

Out[31]: 689.8355778894478

In [32]:
```
sns.boxplot(df["Age"])
import warnings
warnings.filterwarnings('ignore')
```

```
C:\Users\sunda\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.1
2, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretat
ion.
  warnings.warn(
```



5. Handle the Missing values.

In [33]:
```
print(df.isnull())
```

```
     CustomerID  Gender    Age  Annual Income (k$)  Spending Score (1-100)
0         False   False  False               False                   False
1         False   False  False               False                   False
2         False   False  False               False                   False
3         False   False  False               False                   False
4         False   False  False               False                   False
..          ...     ...    ...                 ...                     ...
195       False   False  False               False                   False
196       False   False  False               False                   False
197       False   False  False               False                   False
198       False   False  False               False                   False
199       False   False  False               False                   False

[200 rows x 5 columns]
```

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

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

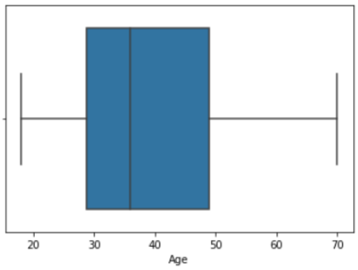
In [35]:
```
df.isna().any()
```

Out[35]:
```
CustomerID                False
Gender                    False
Age                       False
Annual Income (k$)        False
Spending Score (1-100)    False
dtype: bool
```

6. Find the outliers and replace the outliers

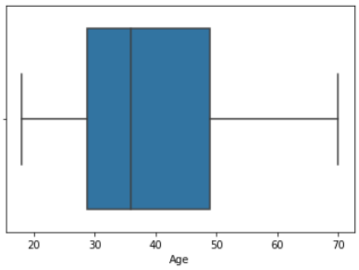
In [36]:
```
x = sns.boxplot(x=df["Age"])
x
```

Out[36]:



In [37]:
```
sns.boxplot(df['Age'])
```

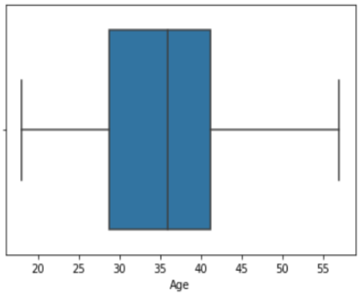
Out[37]:



In [38]:
```
df['Age']=np.where(df['Age']>57,39, df['Age'])
```

In [39]:
```
sns.boxplot(df['Age'])
```

Out[39]:



7.Check for Categorical columns and perform encoding

In [40]:
```
pd.Categorical(df["Annual Income (k$)"])
```

Out[40]:
```
[15, 15, 16, 16, 17, ..., 120, 126, 126, 137, 137]
Length: 200
Categories (64, int64): [15, 16, 17, 18, ..., 113, 120, 126, 137]
```

In [41]:
```
# One Hot Encoding

pd.get_dummies(df["Annual Income (k$)"]).head(10)
```

Out[41]:

	15	16	17	18	19	20	21	23	24	25	...	93	97	98	99	101	103	113	120	126	137
0	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	0	1	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	0	1	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	0	0	1	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
5	0	0	1	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
6	0	0	0	1	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
7	0	0	0	1	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	1	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	1	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

10 rows × 64 columns

In [42]:
```
pd.get_dummies(df).head(10)
```

Out[42]:

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

8. Scaling the data

In [43]:

```
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
```

In [44]:

```
label = LabelEncoder()
label = label.fit_transform(df['Gender'])
df["Gender"] = label
df['Gender'].value_counts()

X = df.drop("Age",axis=1)
Y = df['Age']
```

In [45]:

```
object1 = StandardScaler()
scale = object1.fit_transform(X)
scale
```

Out[45]:

```
array([[-1.7234121 ,  1.12815215, -1.73899919, -0.43480148],
       [-1.70609137,  1.12815215, -1.73899919,  1.19570407],
       [-1.68877065, -0.88640526, -1.70082976, -1.71591298],
       [-1.67144992, -0.88640526, -1.70082976,  1.04041783],
       [-1.6541292 , -0.88640526, -1.66266033, -0.39597992],
       [-1.63680847, -0.88640526, -1.66266033,  1.00159627],
       [-1.61948775, -0.88640526, -1.62449091, -1.71591298],
       [-1.60216702, -0.88640526, -1.62449091,  1.70038436],
       [-1.5848463 ,  1.12815215, -1.58632148, -1.83237767],
       [-1.56752558, -0.88640526, -1.58632148,  0.84631002],
       [-1.55020485,  1.12815215, -1.58632148, -1.4053405 ],
       [-1.53288413, -0.88640526, -1.58632148,  1.89449216],
       [-1.5155634 , -0.88640526, -1.54815205, -1.36651894],
       [-1.49824268, -0.88640526, -1.54815205,  1.04041783],
       [-1.48092195,  1.12815215, -1.54815205, -1.44416206],
       [-1.46360123,  1.12815215, -1.54815205,  1.11806095],
       [-1.4462805 , -0.88640526, -1.50998262, -0.59008772],
       [-1.42895978,  1.12815215, -1.50998262,  0.61338066],
       [-1.41163905,  1.12815215, -1.43364376, -0.82301709],
       [-1.39431833, -0.88640526, -1.43364376,  1.8556706 ],
       [-1.3769976 ,  1.12815215, -1.39547433, -0.59008772],
       [-1.35967688,  1.12815215, -1.39547433,  0.88513158],
       [-1.34235616, -0.88640526, -1.3573049 , -1.75473454],
       [-1.32503543,  1.12815215, -1.3573049 ,  0.88513158],
       [-1.30771471, -0.88640526, -1.24279661, -1.4053405 ],
```

In [46]:

```
X_scaled  = pd.DataFrame(scale, columns = X.columns)
X_scaled
```

Out[46]:

	CustomerID	Gender	Annual Income (k$)	Spending Score (1-100)
0	-1.723412	1.128152	-1.738999	-0.434801
1	-1.706091	1.128152	-1.738999	1.195704
2	-1.688771	-0.886405	-1.700830	-1.715913
3	-1.671450	-0.886405	-1.700830	1.040418
4	-1.654129	-0.886405	-1.662660	-0.395980
...	...	...	...	...
195	1.654129	-0.886405	2.268791	1.118061
196	1.671450	-0.886405	2.497807	-0.861839
197	1.688771	1.128152	2.497807	0.923953
198	1.706091	1.128152	2.917671	-1.250054
199	1.723412	1.128152	2.917671	1.273347

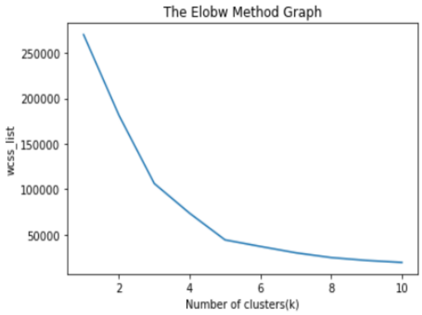
200 rows × 4 columns

9. Perform any of the clustering algorithms

In [48]:
```
from sklearn.cluster import KMeans
```

In [49]:
```
x = df.iloc[:, [3, 4]].values
```

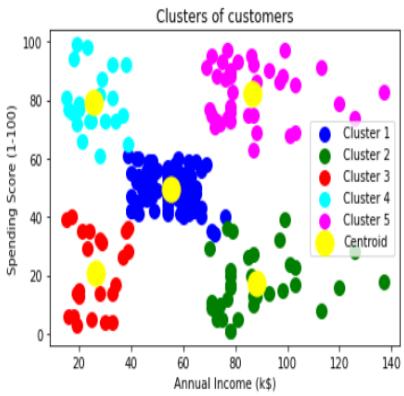
In [50]:
```
list= []

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



In [51]:
```
kmeans = KMeans(n_clusters=5, init='k-means++', random_state= 42)
y_predict= kmeans.fit_predict(x)
```

In [51]:
```
kmeans = KMeans(n_clusters=5, init='k-means++', random_state= 42)
y_predict= kmeans.fit_predict(x)
```


In [52]:
```
plt.scatter(x[y_predict == 0, 0], x[y_predict == 0, 1], s = 100, c = 'blue', label = 'Cluster 1') #for first cluster
plt.scatter(x[y_predict == 1, 0], x[y_predict == 1, 1], s = 100, c = 'green', label = 'Cluster 2') #for second cluster
plt.scatter(x[y_predict== 2, 0], x[y_predict == 2, 1], s = 100, c = 'red', label = 'Cluster 3') #for third cluster
plt.scatter(x[y_predict == 3, 0], x[y_predict == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4') #for fourth cluster
plt.scatter(x[y_predict == 4, 0], x[y_predict == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5') #for fifth cluster
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 300, c = 'yellow', label = 'Centroid')
plt.title('Clusters of customers')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.show()
```



10. Add the cluster data with the primary dataset

In [53]:
```
df['Cluster']=kmeans.labels_
df.head()
```

Out[53]:

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

In [54]:
```
df.tail()
```

Out[54]:

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)	Cluster
195	196	0	35	120	79	4
196	197	0	45	126	28	1
197	198	1	32	126	74	4
198	199	1	32	137	18	1
199	200	1	30	137	83	4

11. Split the data into dependent and independent variables.

In [55]:
```
X=df.drop('Cluster',axis=1)
Y=df['Cluster']
y=df['Cluster']
y
```

Out[55]:
```
0      2
1      3
2      2
3      3
4      2
      ..
195    4
196    1
197    4
198    1
199    4
Name: Cluster, Length: 200, dtype: int32
```

In [56]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,Y,test_size=0.2,random_state=42)
```

In [57]:
```
X_train.shape
```

Out[57]:
```
(160, 5)
```

In [58]:
```
y_train.shape
```

Out[58]:
```
(160,)
```

12. Split the data into training and testing

In [59]:
```
X_train
```

Out[59]:

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)
79	80	0	49	54	42
197	198	1	32	126	74
38	39	0	36	37	26
24	25	0	54	28	14
122	123	0	40	69	58
...	...	...	...	...	...
106	107	0	39	63	50
14	15	1	37	20	13
92	93	1	48	60	49
179	180	1	35	93	90
102	103	1	39	62	59

160 rows × 5 columns

In [60]:
```
X_test
```

Out[60]:

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)
95	96	1	24	60	52
15	16	1	22	20	79
30	31	1	39	30	4
158	159	1	34	78	1
128	129	1	39	71	11
115	116	0	19	65	50
69	70	0	32	48	47

In [61]:
```
y_train
```

Out[61]:
```
79     0
197    4
38     2
24     2
122    0
      ..
106    0
14     2
92     0
179    4
102    0
Name: Cluster, Length: 160, dtype: int32
```

In [62]:
```
y_test
```

Out[62]:
```
95     0
15     3
30     2
158    1
128    1
115    0
69     0
170    1
174    1
45     3
66     0
182    1
165    4
78     0
186    1
177    4
56     0
152    1
82     0
68     0
124    1
16     2
148    1
93     0
65     0
60     0
84     0
67     0
125    4
```

13. Build the Model

In [63]:
```
from sklearn.linear_model import LogisticRegression
model=LogisticRegression()
model.fit(X_train,y_train)
```

Out[63]:
```
LogisticRegression()
```

14. Train the Model

In [64]:
```
model.score(X_train,y_train)
```

Out[64]:
```
0.98125
```

15. Test the Model

In [65]:
```
model.score(X_test,y_test)
```

Out[65]:
```
0.95
```

16. Measure the performance using Evaluation Metrics.

In [66]:
```
from sklearn.metrics import confusion_matrix,classification_report
```

In [67]:
```
y_pred=model.predict(X_test)
confusion_matrix(y_test,y_pred)
```

Out[67]:
```
array([[16,  1,  0,  0,  1],
       [ 0, 11,  0,  0,  0],
       [ 0,  0,  3,  0,  0],
       [ 0,  0,  0,  3,  0],
       [ 0,  0,  0,  0,  5]], dtype=int64)
```

In [68]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.89      0.94        18
           1       0.92      1.00      0.96        11
           2       1.00      1.00      1.00         3
           3       1.00      1.00      1.00         3
           4       0.83      1.00      0.91         5

    accuracy                           0.95        40
   macro avg       0.95      0.98      0.96        40
weighted avg       0.96      0.95      0.95        40
```