Hyundai_Clustering

November 10, 2021

[117]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

[118]:
```
mall_customers = pd.read_csv("Mall_Customer.csv")
```

We are trying to form a cluster of customers based on Annual Income and Spending Score

[119]:
```
mall_customers.head()
```

[119]:

	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

[120]:
```
# missing values
mall_customers.isnull().sum()
```

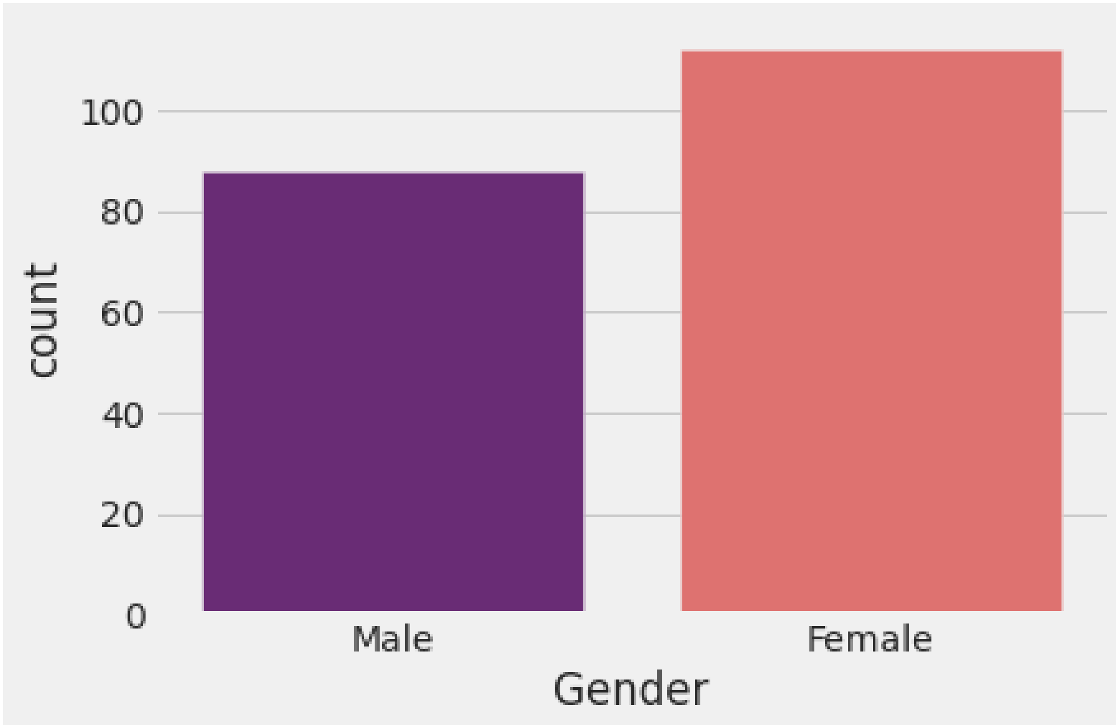
[120]:
```
CustomerID                0
Gender                    0
Age                       0
Annual Income (k$)        0
Spending Score (1-100)    0
dtype: int64
```

[121]:
```
mall_customers['Gender'].value_counts()
```

[121]:
```
Female    112
Male       88
Name: Gender, dtype: int64
```

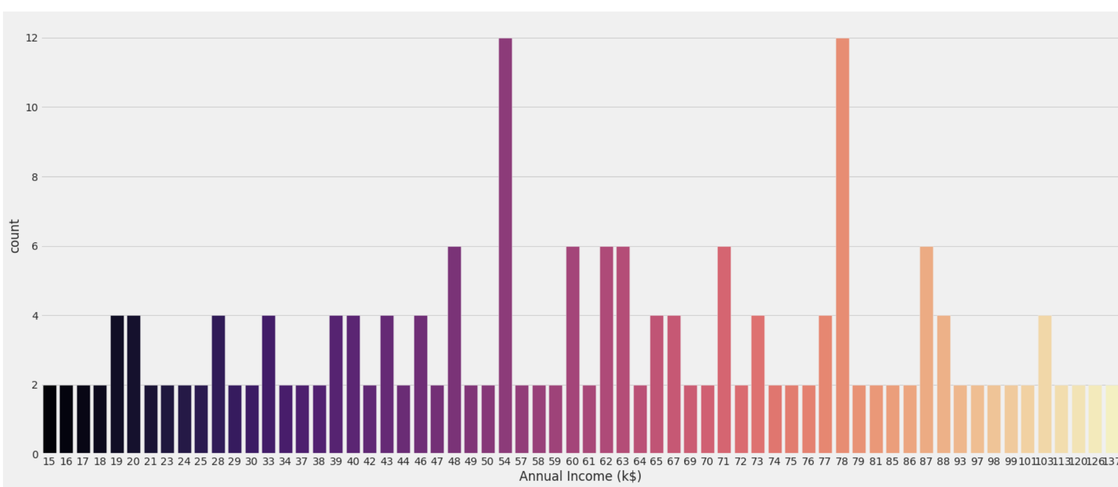
[122]:
```
sns.countplot(x = 'Gender', data = mall_customers, palette='magma')
```

[122]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0a0e2ceb50>
```




```
[123]: plt.figure(figsize=(23,10))
       sns.countplot(x = 'Annual Income (k$)', data = mall_customers, palette='magma')
```

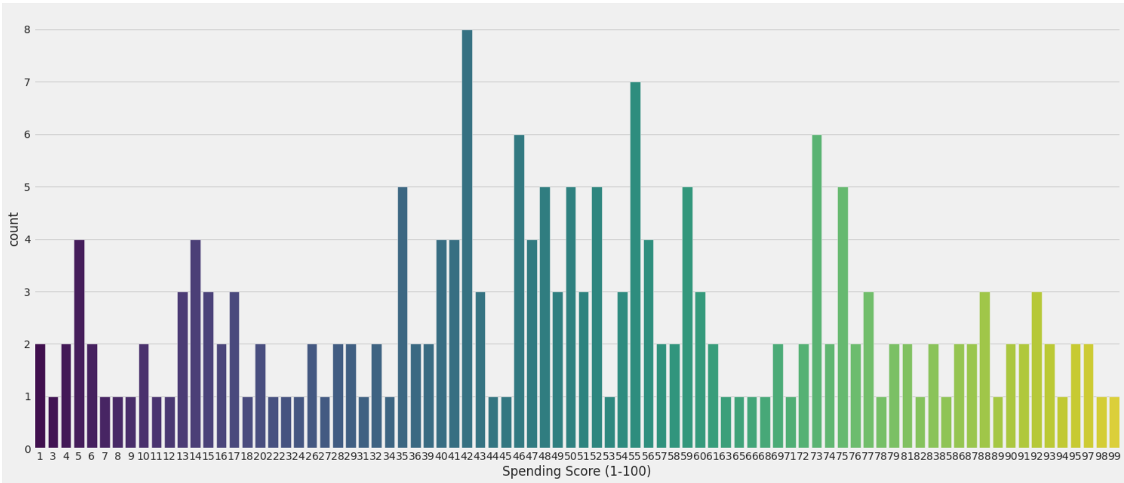
```
[123]: <matplotlib.axes._subplots.AxesSubplot at 0x7f0a0eb03fd0>
```



Top two Annual incomes are 54,78

[124]:
```
plt.figure(figsize=(23,10))
sns.countplot(x = 'Spending Score (1-100)', data = mall_customers,␣
 ↪palette='viridis')
```

[124]: <matplotlib.axes._subplots.AxesSubplot at 0x7f0a0e32f710>



Top two spending scores are 42, 55

Features

[125]:
```
X = mall_customers.iloc[:, [3,4]]
```

[126]:
```
X
```

[126]:
```
     Annual Income (k$)  Spending Score (1-100)
0                    15                      39
1                    15                      81
2                    16                       6
3                    16                      77
4                    17                      40
..                  ...                     ...
195                 120                      79
196                 126                      28
197                 126                      74
198                 137                      18
199                 137                      83

[200 rows x 2 columns]
```

[127]:
```
from sklearn.cluster import KMeans
```

Elbow Method

To find optimum number of clusters, k

- In the Elbow method, we are actually varying the number of clusters (K) from 1 – 15.
- For each value of K, we are calculating WCSS (Within-Cluster Sum of Square).
- WCSS is the sum of squared distance between each point and the centroid in a cluster.
- When we plot the WCSS with the K value, the plot looks like an Elbow.
- As the number of clusters increases, the WCSS value will start to decrease.
- When we analyze the graph we can see that the graph will rapidly change at a point and thus creating an elbow shape.
- From this point, the graph starts to move almost parallel to the X-axis.
- The K value corresponding to this point is the optimal K value or an optimal number of clusters.

[128]:
```
# Use elbow method to find optimum number of cluster i.e. value of 'K'
wcss = []
for i in range(1, 16):
    kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=10,␣
 ↪random_state=0)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)
```

kmeans++ - uses smarter selection of centroids

n_init - Number of time the k-means algorithm will be run with different centroid seeds. The final results will be the best output of n_init consecutive runs in terms of inertia.

max_iter - Maximum number of iterations of the k-means algorithm for a single run

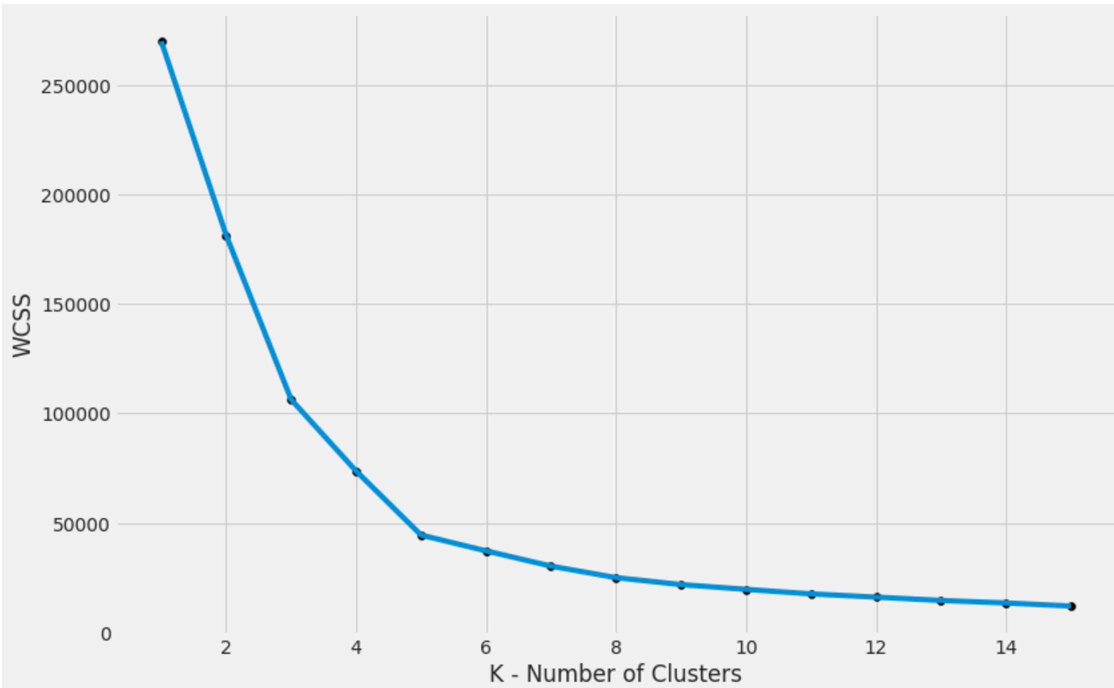
Inertia_ : Sum of squared distances of samples to their closest cluster center

[129]:
```
wcss
```

[129]:
```
[269981.28,
 181363.59595959596,
 106348.37306211118,
 73679.78903948834,
 44448.45544793371,
 37239.83554245604,
 30273.394312070042,
 25018.576334776335,
 21850.165282585633,
 19664.685196005543,
 17602.19046838677,
 16115.215606639838,
 14600.44364738564,
 13450.08023381847,
 12038.745689262341]
```

[130]:
```
# Plot visualization b/w WCSS and Number of Clusters(K)
plt.figure(figsize=(12,8))
```

```python
plt.plot(range(1,16), wcss)
# to view the points clearly we add scatter plot in addition to line
plt.scatter(range(1,16), wcss, c = 'black', marker='o')
plt.xlabel("K - Number of Clusters")
plt.ylabel("WCSS")
plt.show()
```



Optimal Number of Clusters K=5

K-Means with random centroid initialization

[131]:
```python
kmeans = KMeans(n_clusters = 5, init = 'random', max_iter= 300, n_init=10,␣
↪random_state=0)
```

[132]:
```python
kmeans.fit(X)
```

[132]:
```
KMeans(algorithm='auto', copy_x=True, init='random', max_iter=300, n_clusters=5,
       n_init=10, n_jobs=None, precompute_distances='auto', random_state=0,
       tol=0.0001, verbose=0)
```

[133]:
```python
y_clusters = kmeans.predict(X)
y_clusters
```

[133]:
```
array([2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4,
       2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 0,
       2, 4, 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, 3, 1, 3, 0, 3, 1, 3, 1, 3,
       0, 3, 1, 3, 1, 3, 1, 3, 1, 3, 0, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3,
       1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3,
       1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3,
       1, 3], dtype=int32)
```

Performance Evaluation using Silhouette Score

[134]:
```python
from sklearn.metrics import silhouette_score
silhouette_avg = silhouette_score(X, y_clusters)
silhouette_avg
```

[134]:
```
0.553931997444648
```

K-Means with k-means++ centroid initialization

[135]:
```python
kmeans = KMeans(n_clusters = 5, init = 'k-means++', max_iter= 300, n_init=10,␣
↪random_state=0)
```

[136]:
```python
kmeans.fit(X)
```

[136]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=5, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=0, tol=0.0001, verbose=0)
```

[137]:
```python
y_clusters = kmeans.predict(X)
y_clusters
```

[137]:
```
array([3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1,
       3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 0,
       3, 1, 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, 2, 4, 2, 0, 2, 4, 2, 4, 2,
       0, 2, 4, 2, 4, 2, 4, 2, 4, 2, 0, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2,
       4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2,
       4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2,
       4, 2], dtype=int32)
```

Performance Evaluation using Silhouette Score

[138]:
```python
from sklearn.metrics import silhouette_score
silhouette_avg = silhouette_score(X, y_clusters)
silhouette_avg
```

[138]:
```
0.553931997444648
```

[139]:
```python
# Centroids
kmeans.cluster_centers_
```

[139]:
```
array([[55.2962963 , 49.51851852],
       [25.72727273, 79.36363636],
       [86.53846154, 82.12820513],
       [26.30434783, 20.91304348],
       [88.2       , 17.11428571]])
```

Clustering Visualization

convert x to numpy arrays, y is already numpy array

[140]:
```python
y_clusters.astype
```

[140]:
```
<function ndarray.astype>
```

[141]:
```python
x_array = np.array(X)
x_array
```

[141]:
```
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]])
```

[142]:
```
x_array.astype
```

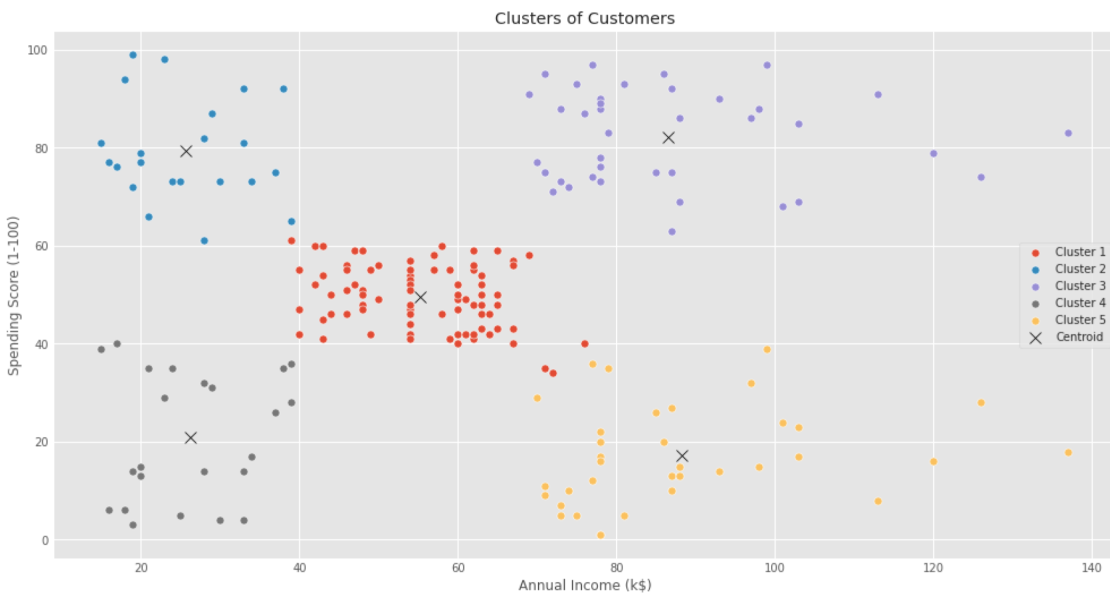
[142]:
```
<function ndarray.astype>
```

[143]:
```
# Visualization of Cluster
plt.style.use("ggplot")
plt.figure(figsize=(15,8))
plt.title('Clusters of Customers')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')

# x_array contain two values, Annual Income and Spending Score
# Index 0 - contains Annual Income and index 1 contains spending score
sns.scatterplot( x = x_array[y_clusters == 0, 0], y = x_array[y_clusters == 0,␣
↪1], label = "Cluster 1", s = 50)
```

```
sns.scatterplot( x = x_array[y_clusters == 1, 0], y = x_array[y_clusters == 1,␣
↪1], label = "Cluster 2", s = 50)
sns.scatterplot( x = x_array[y_clusters == 2, 0], y = x_array[y_clusters == 2,␣
↪1], label = "Cluster 3", s = 50)
sns.scatterplot( x = x_array[y_clusters == 3, 0], y = x_array[y_clusters == 3,␣
↪1], label = "Cluster 4", s = 50)
sns.scatterplot( x = x_array[y_clusters == 4, 0], y = x_array[y_clusters == 4,␣
↪1], label = "Cluster 5", s = 50)
#plot the centroid of the cluster inside the plot
sns.scatterplot(kmeans.cluster_centers_[:,0],kmeans.cluster_centers_[:
↪,1],marker='x',color='black',label='Centroid',s=100)
plt.show()
```

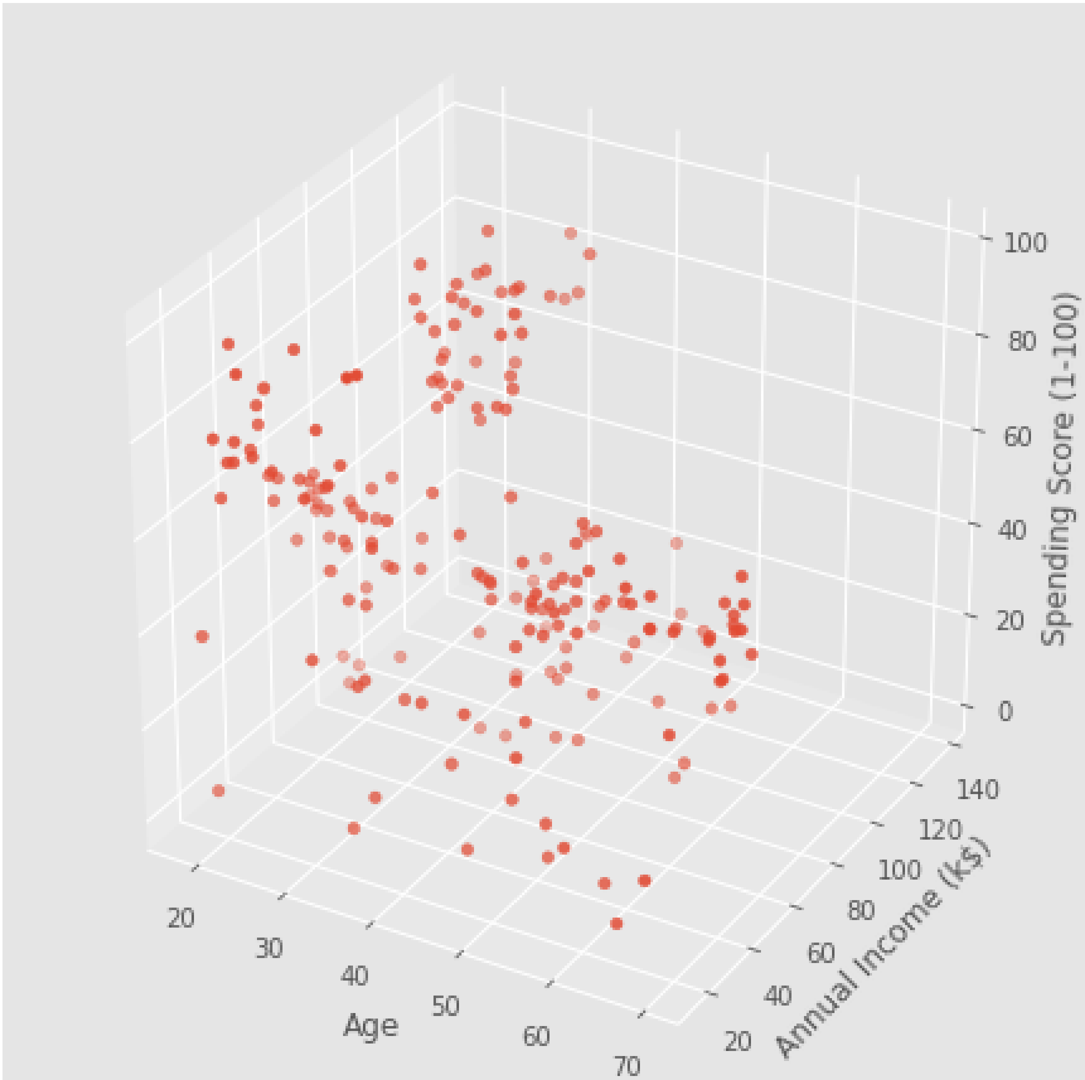
```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning:
Pass the following variables as keyword args: x, y. From version 0.12, the only
valid positional argument will be `data`, and passing other arguments without an
explicit keyword will result in an error or misinterpretation.
  FutureWarning
```



[144]:
```
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure(figsize=(6,6))
ax = Axes3D(fig)
ax.scatter(mall_customers['Age'], mall_customers['Annual Income (k$)'],␣
↪mall_customers['Spending Score (1-100)'])
#ax.scatter(dataframe['x2'], dataframe['x1'], predicted)
ax.set_xlabel('Age')
ax.set_ylabel('Annual Income (k$)')
```

```
ax.set_zlabel('Spending Score (1-100)')
```

[144]: Text(0.5, 0, 'Spending Score (1-100)')



[145]:
```
x = mall_customers.iloc[:, 2:].values

# let's check the shape of x
print(x.shape)
```

(200, 3)

[146]:
```
x
```

```
[146]: array([[ 19,  15,  39],
       [ 21,  15,  81],
       [ 20,  16,   6],
       [ 23,  16,  77],
       [ 31,  17,  40],
       [ 22,  17,  76],
       [ 35,  18,   6],
       [ 23,  18,  94],
       [ 64,  19,   3],
       [ 30,  19,  72],
       [ 67,  19,  14],
       [ 35,  19,  99],
       [ 58,  20,  15],
       [ 24,  20,  77],
       [ 37,  20,  13],
       [ 22,  20,  79],
       [ 35,  21,  35],
       [ 20,  21,  66],
       [ 52,  23,  29],
       [ 35,  23,  98],
       [ 35,  24,  35],
       [ 25,  24,  73],
       [ 46,  25,   5],
       [ 31,  25,  73],
       [ 54,  28,  14],
       [ 29,  28,  82],
       [ 45,  28,  32],
       [ 35,  28,  61],
       [ 40,  29,  31],
       [ 23,  29,  87],
       [ 60,  30,   4],
       [ 21,  30,  73],
       [ 53,  33,   4],
       [ 18,  33,  92],
       [ 49,  33,  14],
       [ 21,  33,  81],
       [ 42,  34,  17],
       [ 30,  34,  73],
       [ 36,  37,  26],
       [ 20,  37,  75],
       [ 65,  38,  35],
       [ 24,  38,  92],
       [ 48,  39,  36],
       [ 31,  39,  61],
       [ 49,  39,  28],
       [ 24,  39,  65],
       [ 50,  40,  55],
```

```
[ 27,  40,  47],
[ 29,  40,  42],
[ 31,  40,  42],
[ 49,  42,  52],
[ 33,  42,  60],
[ 31,  43,  54],
[ 59,  43,  60],
[ 50,  43,  45],
[ 47,  43,  41],
[ 51,  44,  50],
[ 69,  44,  46],
[ 27,  46,  51],
[ 53,  46,  46],
[ 70,  46,  56],
[ 19,  46,  55],
[ 67,  47,  52],
[ 54,  47,  59],
[ 63,  48,  51],
[ 18,  48,  59],
[ 43,  48,  50],
[ 68,  48,  48],
[ 19,  48,  59],
[ 32,  48,  47],
[ 70,  49,  55],
[ 47,  49,  42],
[ 60,  50,  49],
[ 60,  50,  56],
[ 59,  54,  47],
[ 26,  54,  54],
[ 45,  54,  53],
[ 40,  54,  48],
[ 23,  54,  52],
[ 49,  54,  42],
[ 57,  54,  51],
[ 38,  54,  55],
[ 67,  54,  41],
[ 46,  54,  44],
[ 21,  54,  57],
[ 48,  54,  46],
[ 55,  57,  58],
[ 22,  57,  55],
[ 34,  58,  60],
[ 50,  58,  46],
[ 68,  59,  55],
[ 18,  59,  41],
[ 48,  60,  49],
[ 40,  60,  40],
```

```
[ 32,  60,  42],
[ 24,  60,  52],
[ 47,  60,  47],
[ 27,  60,  50],
[ 48,  61,  42],
[ 20,  61,  49],
[ 23,  62,  41],
[ 49,  62,  48],
[ 67,  62,  59],
[ 26,  62,  55],
[ 49,  62,  56],
[ 21,  62,  42],
[ 66,  63,  50],
[ 54,  63,  46],
[ 68,  63,  43],
[ 66,  63,  48],
[ 65,  63,  52],
[ 19,  63,  54],
[ 38,  64,  42],
[ 19,  64,  46],
[ 18,  65,  48],
[ 19,  65,  50],
[ 63,  65,  43],
[ 49,  65,  59],
[ 51,  67,  43],
[ 50,  67,  57],
[ 27,  67,  56],
[ 38,  67,  40],
[ 40,  69,  58],
[ 39,  69,  91],
[ 23,  70,  29],
[ 31,  70,  77],
[ 43,  71,  35],
[ 40,  71,  95],
[ 59,  71,  11],
[ 38,  71,  75],
[ 47,  71,   9],
[ 39,  71,  75],
[ 25,  72,  34],
[ 31,  72,  71],
[ 20,  73,   5],
[ 29,  73,  88],
[ 44,  73,   7],
[ 32,  73,  73],
[ 19,  74,  10],
[ 35,  74,  72],
[ 57,  75,   5],
```

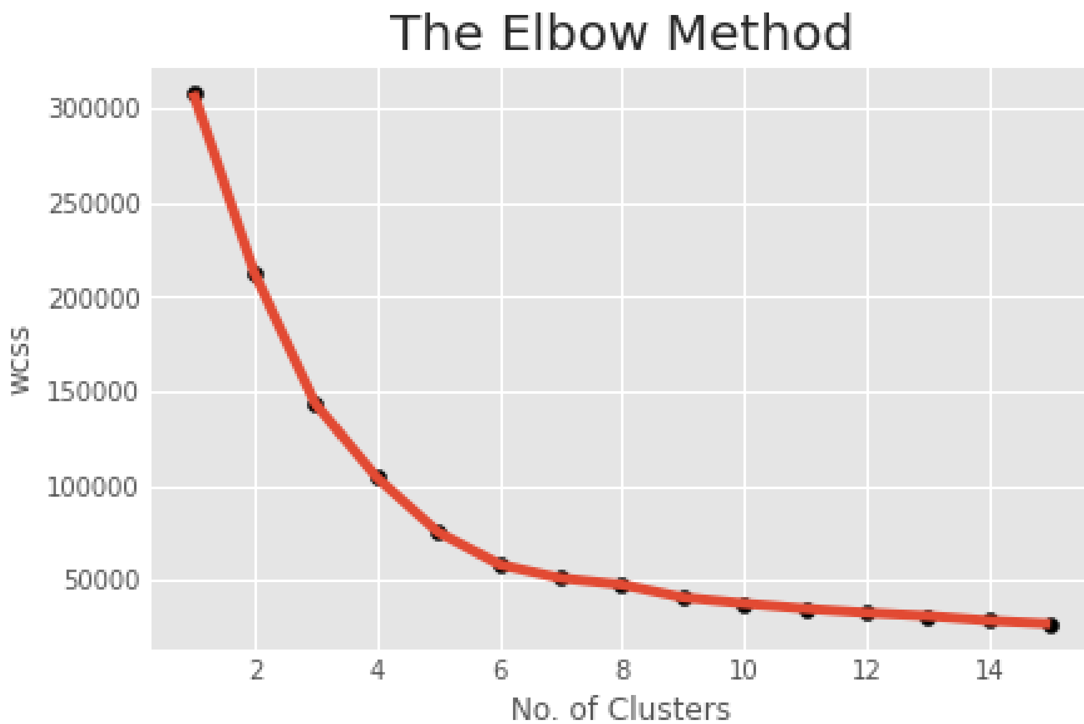
```
[ 32,  75,  93],
[ 28,  76,  40],
[ 32,  76,  87],
[ 25,  77,  12],
[ 28,  77,  97],
[ 48,  77,  36],
[ 32,  77,  74],
[ 34,  78,  22],
[ 34,  78,  90],
[ 43,  78,  17],
[ 39,  78,  88],
[ 44,  78,  20],
[ 38,  78,  76],
[ 47,  78,  16],
[ 27,  78,  89],
[ 37,  78,   1],
[ 30,  78,  78],
[ 34,  78,   1],
[ 30,  78,  73],
[ 56,  79,  35],
[ 29,  79,  83],
[ 19,  81,   5],
[ 31,  81,  93],
[ 50,  85,  26],
[ 36,  85,  75],
[ 42,  86,  20],
[ 33,  86,  95],
[ 36,  87,  27],
[ 32,  87,  63],
[ 40,  87,  13],
[ 28,  87,  75],
[ 36,  87,  10],
[ 36,  87,  92],
[ 52,  88,  13],
[ 30,  88,  86],
[ 58,  88,  15],
[ 27,  88,  69],
[ 59,  93,  14],
[ 35,  93,  90],
[ 37,  97,  32],
[ 32,  97,  86],
[ 46,  98,  15],
[ 29,  98,  88],
[ 41,  99,  39],
[ 30,  99,  97],
[ 54, 101,  24],
[ 28, 101,  68],
```

```
       [ 41, 103,  17],
       [ 36, 103,  85],
       [ 34, 103,  23],
       [ 32, 103,  69],
       [ 33, 113,   8],
       [ 38, 113,  91],
       [ 47, 120,  16],
       [ 35, 120,  79],
       [ 45, 126,  28],
       [ 32, 126,  74],
       [ 32, 137,  18],
       [ 30, 137,  83]])
```

[147]:
```
from sklearn.cluster import KMeans

wcss = []
for i in range(1, 16):
    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, 16), wcss)
# to view the points clearly we add scatter plot in addition to line
plt.scatter(range(1,16), wcss, c = 'black', marker='o')
plt.title('The Elbow Method', fontsize = 20)
plt.xlabel('No. of Clusters')
plt.ylabel('wcss')
plt.show()
```

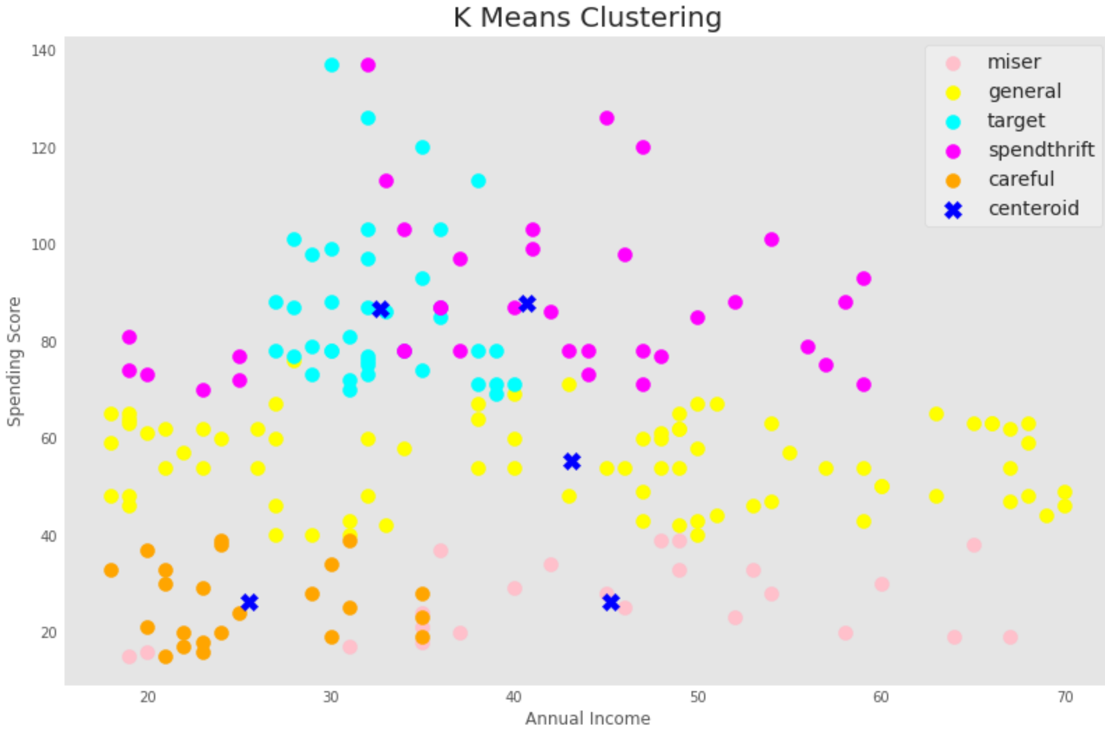


[148]:

```
km = KMeans(n_clusters = 5, init = 'k-means++', max_iter = 300, n_init = 10,␣
 ↪random_state = 0)
y_means = km.fit_predict(x)
plt.figure(figsize=(12,8))
plt.scatter(x[y_means == 0, 0], x[y_means == 0, 1], s = 100, c = 'pink', label␣
 ↪= 'miser')
plt.scatter(x[y_means == 1, 0], x[y_means == 1, 1], s = 100, c = 'yellow',␣
 ↪label = 'general')
plt.scatter(x[y_means == 2, 0], x[y_means == 2, 1], s = 100, c = 'cyan', label␣
 ↪= 'target')
plt.scatter(x[y_means == 3, 0], x[y_means == 3, 1], s = 100, c = 'magenta',␣
 ↪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 = 150, c =␣
 ↪'blue' , marker = 'X',label = 'centeroid')

plt.style.use('fivethirtyeight')
plt.title('K Means Clustering', fontsize = 20)
plt.xlabel('Annual Income')
plt.ylabel('Spending Score')
plt.legend()
plt.grid()
```

```
plt.show()
```



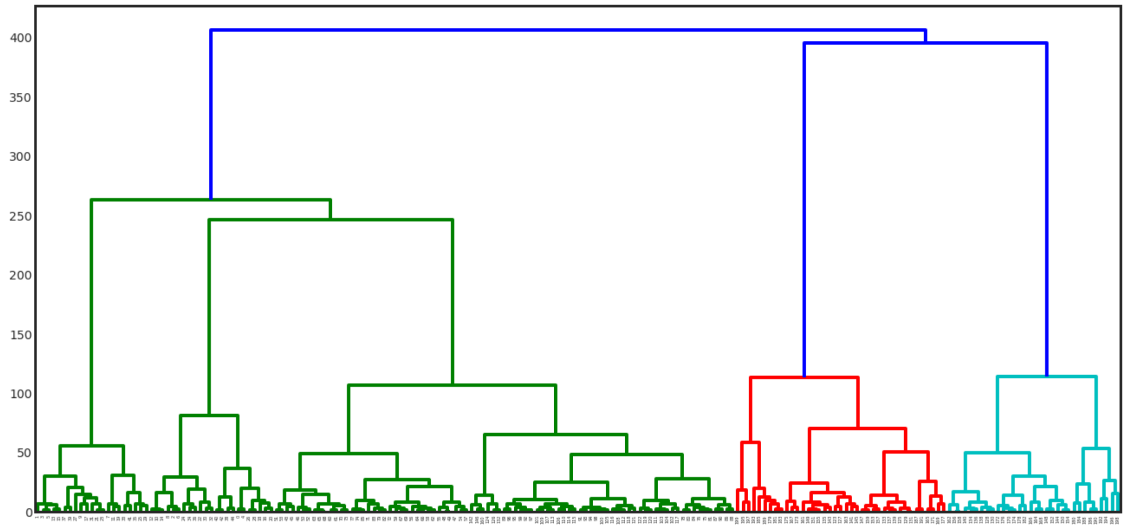
0.1 Hierarchical Clustering

Agglomerative Clustering - Bottom up approach

Dendogram

- It is a tree like diagram, it illustrates the arrangement of cluster produced by analysis of cluster
- Dendogram is used by hiererchical clustering to find number of clusters
- Dendogram is summary of distance matrix between two different clusters

[149]:
```
import scipy.cluster.hierarchy as sch
sns.set_style('white')
fig = plt.figure(figsize=(20,10))
dendrogram = sch.dendrogram(sch.linkage(X, method = 'ward'))
plt.show()
```



Number of clusters can be found by number of vertical lines in the top

[150]:
```
from sklearn.cluster import AgglomerativeClustering
hc = AgglomerativeClustering(n_clusters = 5, affinity = 'euclidean', linkage =
 ↪'ward')
```

[151]:
```
y_hc = hc.fit_predict(X)
```

[152]:
```
y_hc
```

[152]:
```
array([4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3,
       4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 1,
       4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 2, 1, 2, 0, 2, 0, 2,
       1, 2, 0, 2, 0, 2, 0, 2, 0, 2, 1, 2, 0, 2, 1, 2, 0, 2, 0, 2, 0, 2,
       0, 2, 0, 2, 0, 2, 1, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2,
       0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2,
       0, 2])
```

[152]:
```
```