# Hierarchical Clustering

Problem statement : Implement a hierarchical clustering method and check whether result differs from above one.

## Theory

1. Hierarchical clustering is another unsupervised machine learning algorithm, which is used to group the unlabeled datasets into a cluster and also known as hierarchical cluster analysis or HCA.

2. In this algorithm, we develop the hierarchy of clusters in the form of a tree, and this tree-shaped structure is known as the dendrogram.

3. Sometimes the results of K-means clustering and hierarchical clustering may look similar, but they both differ depending on how they work. As there is no requirement to predetermine the number of clusters as we did in the K-Means algorithm.

The hierarchical clustering technique has two approaches:

i. Agglomerative: Agglomerative is a bottom-up approach, in which the algorithm starts with taking all data points as single clusters and merging them until one cluster is left.

ii. Divisive: Divisive algorithm is the reverse of the agglomerative algorithm as it is a top-down approach.

## Implementation

The steps for implementation will be the same as the k-means clustering, except for some changes such as the method to find the number of clusters. Below are the steps:

1. Data Pre-processing
2. Finding the optimal number of clusters using the Dendrogram
3. Training the hierarchical clustering model
4. Visualizing the clusters

## Step 1 : Data Pre-processing Steps

Importing the libraries

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

df = pd.read_csv("Mall_Customers.csv")

df
```

```
     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]
```

```
df.shape
```

```
(200, 5)
```

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

```
False
```

Here we will extract only the matrix of features as we don't have any further information about the dependent variable.

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

```
x
```

```
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)
```

```
print(x.shape)
```

```
(200, 2)
```

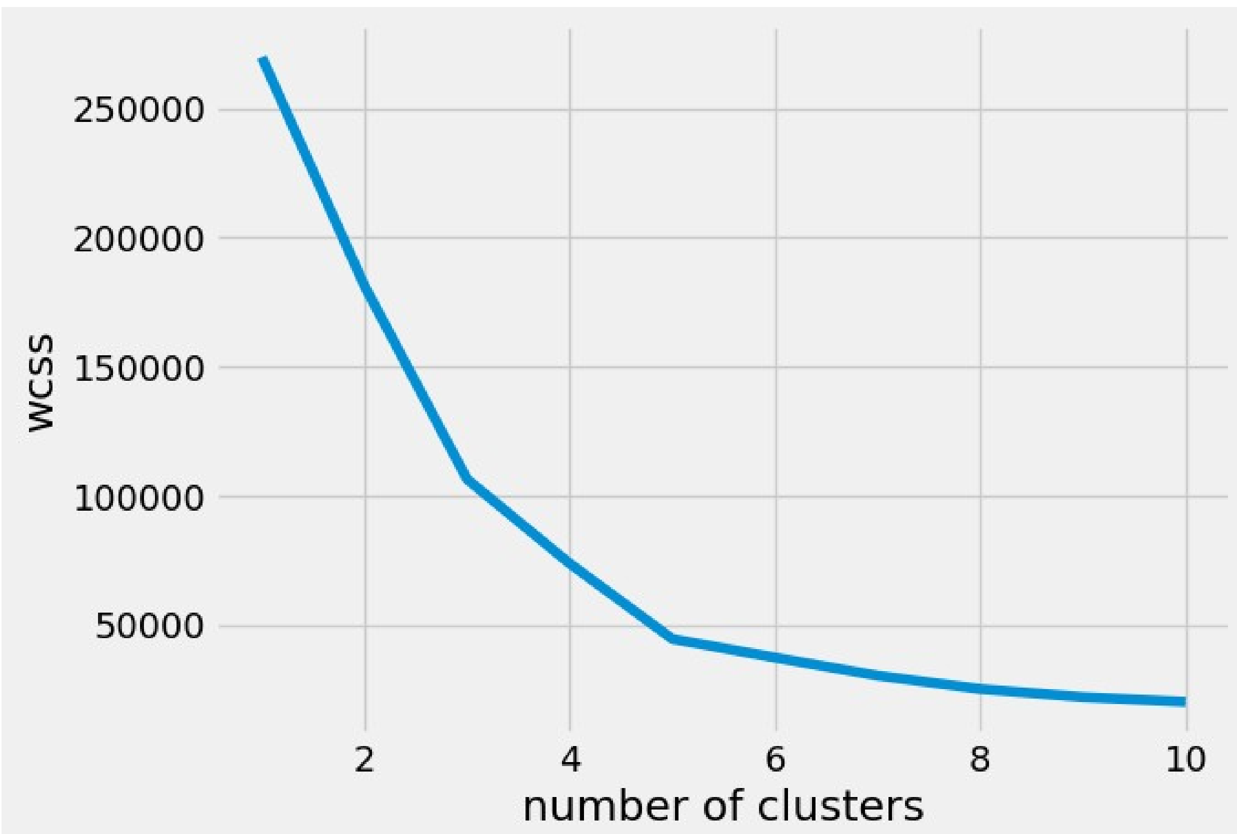
```python
#initializing wcss
from sklearn.cluster import KMeans
wcss = []

for i in range(1,11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 
42)
    kmeans.fit(x)
    wcss.append(kmeans.inertia_)
```

```
C:\Users\UMAP\AppData\Roaming\Python\Python39\site-packages\sklearn\
cluster\_kmeans.py:1412: FutureWarning: The default value of `n_init`
will change from 10 to 'auto' in 1.4. Set the value of `n_init`
explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\UMAP\AppData\Roaming\Python\Python39\site-packages\sklearn\
cluster\_kmeans.py:1412: FutureWarning: The default value of `n_init`
will change from 10 to 'auto' in 1.4. Set the value of `n_init`
explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\UMAP\AppData\Roaming\Python\Python39\site-packages\sklearn\
cluster\_kmeans.py:1412: FutureWarning: The default value of `n_init`
will change from 10 to 'auto' in 1.4. Set the value of `n_init`
explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```

```
C:\Users\UMAP\AppData\Roaming\Python\Python39\site-packages\sklearn\
cluster\_kmeans.py:1412: FutureWarning: The default value of `n_init`
will change from 10 to 'auto' in 1.4. Set the value of `n_init`
explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\UMAP\AppData\Roaming\Python\Python39\site-packages\sklearn\
cluster\_kmeans.py:1412: FutureWarning: The default value of `n_init`
will change from 10 to 'auto' in 1.4. Set the value of `n_init`
explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\UMAP\AppData\Roaming\Python\Python39\site-packages\sklearn\
cluster\_kmeans.py:1412: FutureWarning: The default value of `n_init`
will change from 10 to 'auto' in 1.4. Set the value of `n_init`
explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\UMAP\AppData\Roaming\Python\Python39\site-packages\sklearn\
cluster\_kmeans.py:1412: FutureWarning: The default value of `n_init`
will change from 10 to 'auto' in 1.4. Set the value of `n_init`
explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\UMAP\AppData\Roaming\Python\Python39\site-packages\sklearn\
cluster\_kmeans.py:1412: FutureWarning: The default value of `n_init`
will change from 10 to 'auto' in 1.4. Set the value of `n_init`
explicitly to suppress the warning
```

```
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\UMAP\AppData\Roaming\Python\Python39\site-packages\sklearn\
cluster\_kmeans.py:1412: FutureWarning: The default value of `n_init`
will change from 10 to 'auto' in 1.4. Set the value of `n_init`
explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\Users\UMAP\AppData\Roaming\Python\Python39\site-packages\sklearn\
cluster\_kmeans.py:1412: FutureWarning: The default value of `n_init`
will change from 10 to 'auto' in 1.4. Set the value of `n_init`
explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```

```
plt.plot(range(1,11), wcss)
plt.xlabel('number of clusters')
plt.ylabel('wcss')
plt.show()
```




```
kmeans = KMeans(n_clusters=5, init = "k-means++", random_state=42)
y_kmeans = kmeans.fit_predict(x)
```

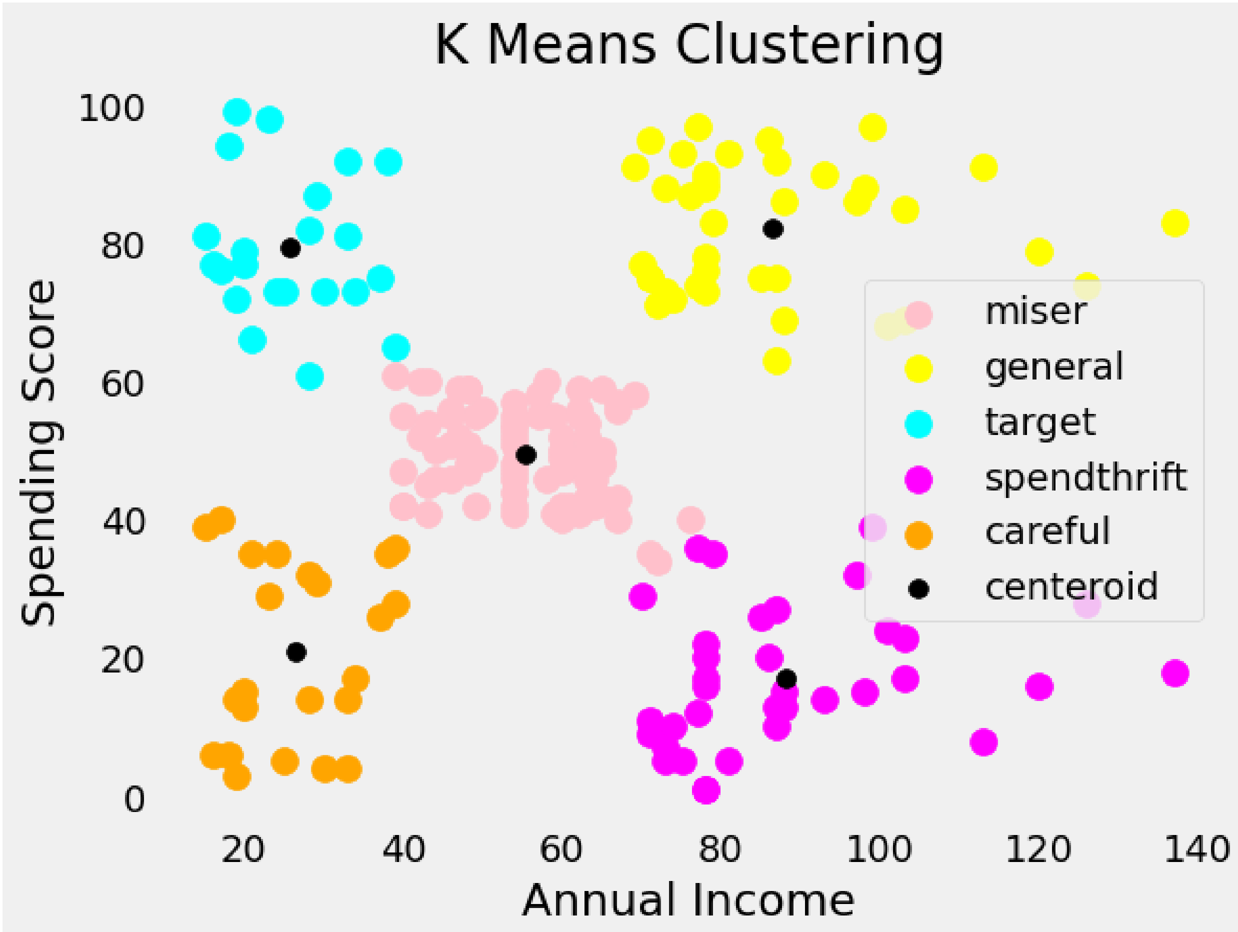
```
C:\Users\UMAP\AppData\Roaming\Python\Python39\site-packages\sklearn\
cluster\_kmeans.py:1412: FutureWarning: The default value of `n_init`
will change from 10 to 'auto' in 1.4. Set the value of `n_init`
```

```
explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```

```
y_kmeans
```

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

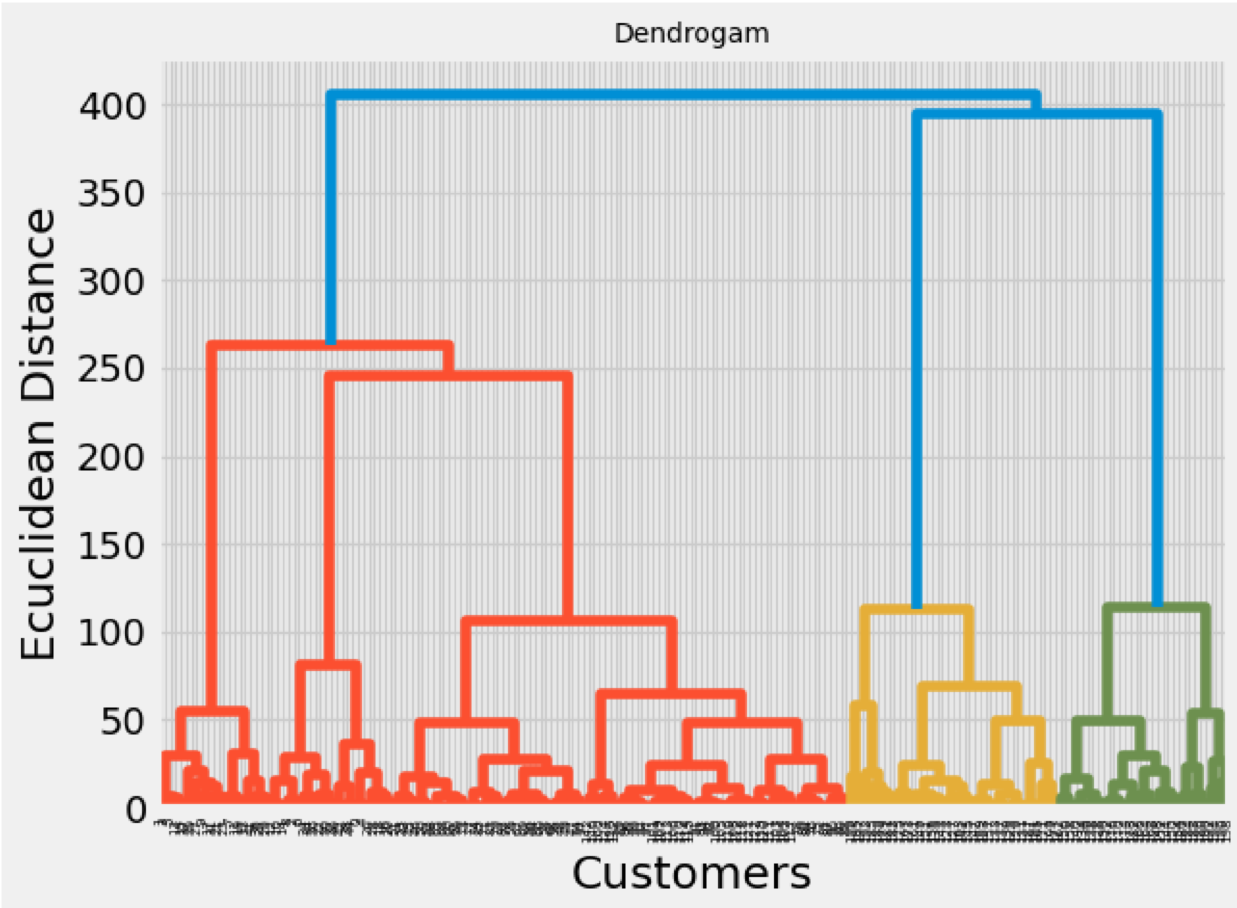
```
plt.scatter(x[y_kmeans == 0, 0], x[y_kmeans == 0, 1], s = 100, c =
'pink', label = 'miser')
plt.scatter(x[y_kmeans == 1, 0], x[y_kmeans == 1, 1], s = 100, c =
'yellow', label = 'general')
plt.scatter(x[y_kmeans == 2, 0], x[y_kmeans == 2, 1], s = 100, c =
'cyan', label = 'target')
plt.scatter(x[y_kmeans == 3, 0], x[y_kmeans == 3, 1], s = 100, c =
'magenta', label = 'spendthrift')
plt.scatter(x[y_kmeans == 4, 0], x[y_kmeans == 4, 1], s = 100, c =
'orange', label = 'careful')
plt.scatter(kmeans.cluster_centers_[:,0], kmeans.cluster_centers_[:,
1], s = 50, c = 'black' , 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()
```




Step-2: Finding the optimal number of clusters using the Dendrogram

Now we will find the optimal number of clusters using the Dendrogram for our model. For this, we are going to use scipy library as it provides a function that will directly return the dendrogram for our code.

```
import scipy.cluster.hierarchy as sch
dendrogram = sch.dendrogram(sch.linkage(x, method = 'ward'))
plt.title('Dendrogam', fontsize = 10)
plt.xlabel('Customers')
plt.ylabel('Ecuclidean Distance')
plt.show()
```




Step-3: Training the hierarchical clustering model

As we know the required optimal number of clusters, we can now train our model.

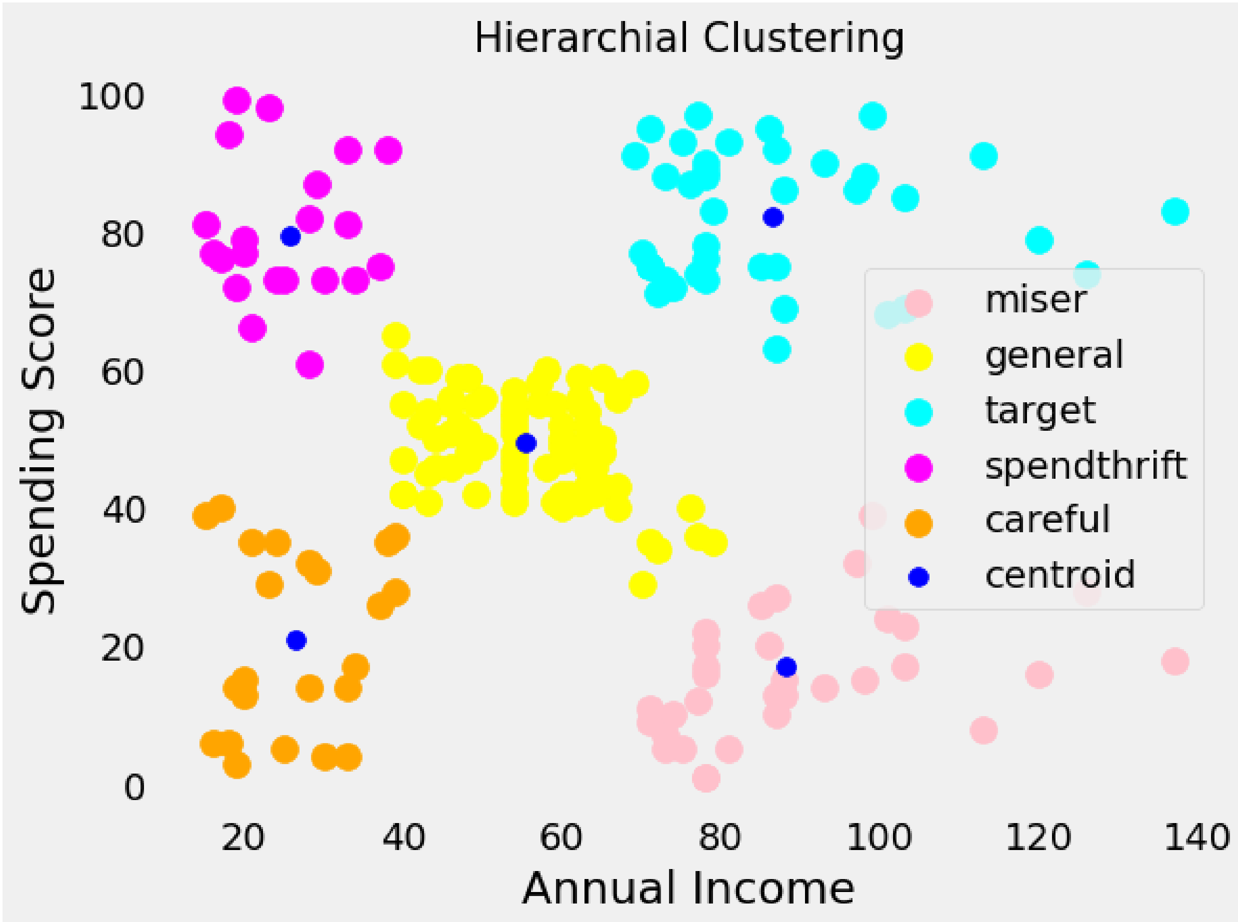
```
from sklearn.cluster import AgglomerativeClustering
hc = AgglomerativeClustering(n_clusters = 5, affinity = 'euclidean',
linkage = 'ward')
y_hc = hc.fit_predict(x)

C:\Users\UMAP\AppData\Roaming\Python\Python39\site-packages\sklearn\
cluster\_agglomerative.py:1005: FutureWarning: Attribute `affinity`
was deprecated in version 1.2 and will be removed in 1.4. Use `metric`
instead
  warnings.warn(

y_hc

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, 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], dtype=int64)
```

### Step-4: Visualizing the clusters

As we have trained our model successfully, now we can visualize the clusters corresponding to the dataset. Here we will use the same lines of code as we did in k-means clustering, except one change. Here we will not plot the centroid that we did in k-means, because here we have used dendrogram to determine the optimal number of clusters.

```
plt.scatter(x[y_hc == 0, 0], x[y_hc == 0, 1], s = 100, c = 'pink',
label = 'miser')
plt.scatter(x[y_hc == 1, 0], x[y_hc == 1, 1], s = 100, c = 'yellow',
label = 'general')
plt.scatter(x[y_hc == 2, 0], x[y_hc == 2, 1], s = 100, c = 'cyan',
label = 'target')
plt.scatter(x[y_hc == 3, 0], x[y_hc == 3, 1], s = 100, c = 'magenta',
label = 'spendthrift')
plt.scatter(x[y_hc == 4, 0], x[y_hc == 4, 1], s = 100, c = 'orange',
label = 'careful')
plt.scatter(kmeans.cluster_centers_[:,0], kmeans.cluster_centers_[:,
1], s = 50, c = 'blue' , label = 'centroid')
plt.style.use('fivethirtyeight')
plt.title('Hierarchial Clustering', fontsize = 15)
plt.xlabel('Annual Income')
plt.ylabel('Spending Score')
plt.legend()
plt.grid()
plt.show()
```




## Conclusion

Thus Hierarchical clustering algorithm was implemented to cluster customers based on their annual income and spending score. The optimal number of clusters was determined to be 5. The algorithm was trained on the dataset, and the clusters were visualized using a scatter plot.

The clusters were analyzed as follows:

- Cluster 1: Customers with average income and average spending, categorized as "general"
- Cluster 2: Customers with high income and high spending, categorized as "target" and considered highly profitable for the mall owner.
- Cluster 3: Customers with low income but very high spending, categorized as "spendthrift"
- Cluster 4: Customers with high income but low spending, categorized as "miser"
- Cluster 5: Customers with low income and low spending, categorized as "careful"

These findings provide valuable insights into different customer segments, allowing the mall owner to tailor their marketing strategies and services to better cater to each cluster's characteristics.