JUNAID GIRKAR
60004190057
TE Comps A4

EXPERIMENT 4

Aim

Implementation of K Means and Hierarchical Clustering algorithm

Theory

Clustering is the task of dividing the population or data points into a number of groups such that data points in the same groups are more similar to other data points in the same group and dissimilar to the data points in other groups. It is basically a collection of objects on the basis of similarity and dissimilarity between them.
Clustering is very much important as it determines the intrinsic grouping among the unlabelled data present. There are no criteria for good clustering. It depends on the user, what is the criteria they may use which satisfy their need. For instance, we could be interested in finding representatives for homogeneous groups (data reduction), in finding “natural clusters” and describe their unknown properties (“natural” data types), in finding useful and suitable groupings (“useful” data classes) or in finding unusual data objects (outlier detection). This algorithm must make some assumptions that constitute the similarity of points and each assumption make different and equally valid clusters.

Clustering Methods :

- **Density-Based Methods**: These methods consider the clusters as the dense region having some similarities and differences from the lower dense region of the space. These methods have good accuracy and the ability to merge two clusters. Example DBSCAN (Density-Based Spatial Clustering of Applications with Noise), OPTICS (Ordering Points to Identify Clustering Structure), etc.
- **Hierarchical Based Methods:** The clusters formed in this method form a tree-type structure based on the hierarchy. New clusters are formed using the previously formed one. It is divided into two category
    - Agglomerative (bottom-up approach)
    - Divisive (top-down approach)

- **Partitioning Methods:** These methods partition the objects into k clusters and each partition forms one cluster. This method is used to optimize an objective criterion similarity function such as when the distance is a major parameter example K-means, CLARANS (Clustering Large Applications based upon Randomized Search), etc.
- **Grid-based Method**s: In this method, the data space is formulated into a finite number of cells that form a grid-like structure. All the clustering operations done on these grids are fast and independent of the number of data objects.

K Means

K-Means Clustering is an Unsupervised Learning algorithm, which groups the unlabeled dataset into different clusters. Here K defines the number of pre-defined clusters that need to be created in the process, as if K=2, there will be two clusters, and for K=3, there will be three clusters, and so on.
there will be two clusters, and for K=3, there will be three clusters, and so on.

It is an iterative algorithm that divides the unlabeled dataset into k different clusters in such a way that each dataset belongs only one group that has similar properties.
It allows us to cluster the data into different groups and a convenient way to discover the categories of groups in the unlabeled dataset on its own without the need for any training.

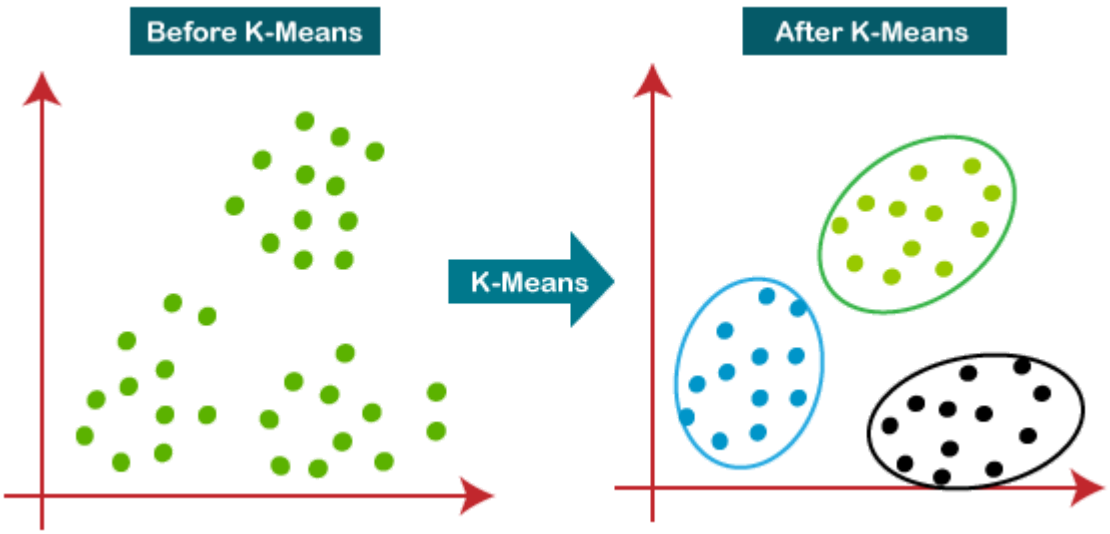
It is a centroid-based algorithm, where each cluster is associated with a centroid. The main aim of this algorithm is to minimize the sum of distances between the data point and their corresponding clusters.

The algorithm takes the unlabeled dataset as input, divides the dataset into k-number of clusters, and repeats the process until it does not find the best clusters. The value of k should be predetermined in this algorithm.

The k-means clustering algorithm mainly performs two tasks:

Determines the best value for K center points or centroids by an iterative process.
Assigns each data point to its closest k-center. Those data points which are near to the particular k-center, create a cluster.
Hence each cluster has datapoints with some commonalities, and it is away from other clusters.

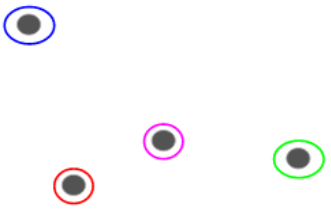
The below diagram explains the working of the K-means Clustering Algorithm:



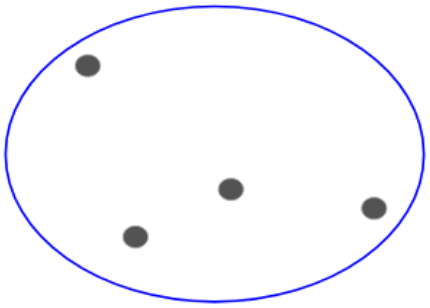
Hierarchical

Let's say we have the below points and we want to cluster them into groups:

We can assign each of these points to a separate cluster:



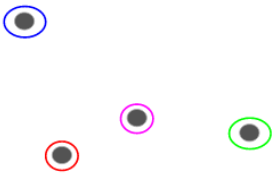
Now, based on the similarity of these clusters, we can combine the most similar clusters together and repeat this process until only a single cluster is left:



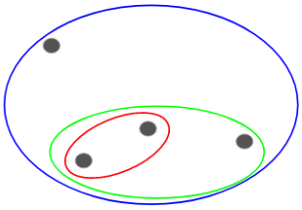
We are essentially building a hierarchy of clusters. That's why this algorithm is called hierarchical clustering. I will discuss how to decide the number of clusters in a later section.

Agglomerative Hierarchical Clustering

We assign each point to an individual cluster in this technique. Suppose there are 4 data points. We will assign each of these points to a cluster and hence will have 4 clusters in the beginning:



Then, at each iteration, we merge the closest pair of clusters and repeat this step until only a single cluster is left:

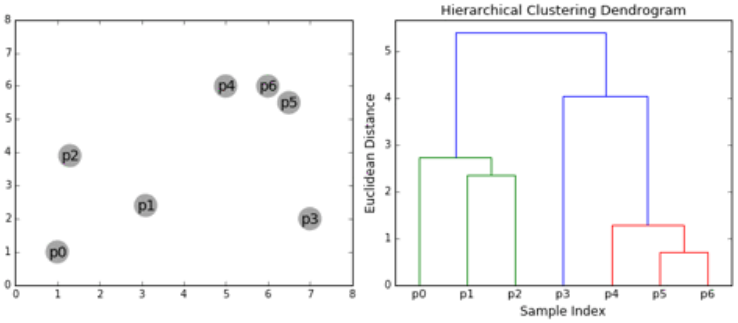


We are merging (or adding) the clusters at each step, right? Hence, this type of clustering is also known as additive hierarchical clustering.

DENDROGRAM

A Dendrogram is a type of tree diagram showing hierarchical relationships between different sets of data.

As already said, a Dendrogram contains the memory of a hierarchical clustering algorithm, so just by looking at the Dendrogram you can tell how the cluster is formed.



PART A (Using Inbuilt function)

K Means:

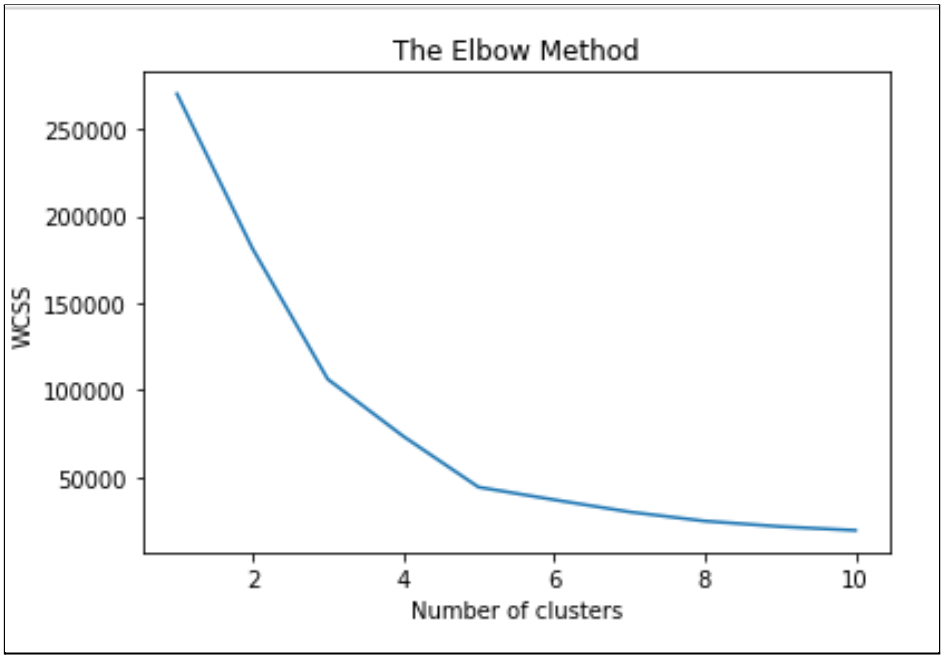
CODE:

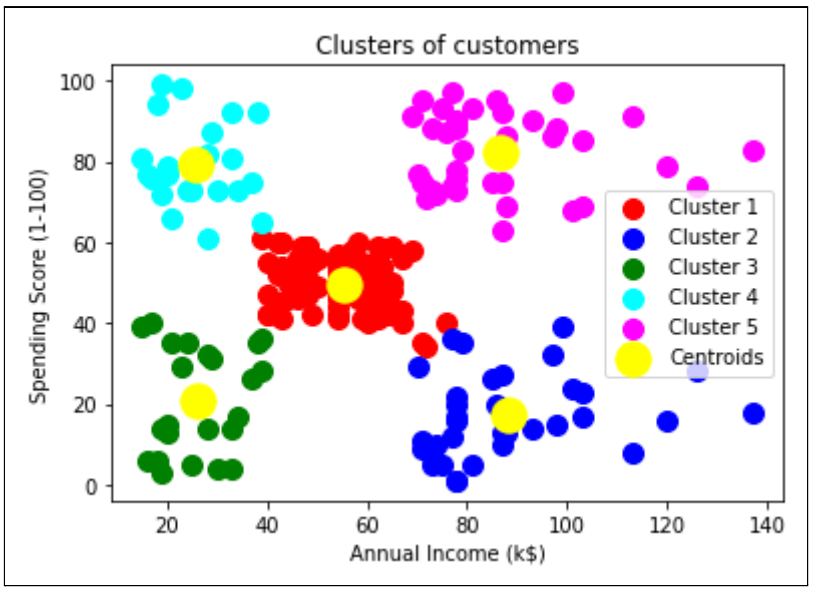
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

dataset = pd.read_csv('Mall_Customers.csv')
dataset.head()

X = dataset.iloc[:, [3, 4]].values

X

# Using the elbow method to find the optimal number of clusters
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_)
plt.plot(range(1, 11), wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()

# Training the K-Means model on the dataset
kmeans = KMeans(n_clusters = 5, init = 'k-means++', random_state =
42)
y_kmeans = kmeans.fit_predict(X)

print(y_kmeans)

# Visualising the clusters
plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], s = 100, c =
'red', label = 'Cluster 1')
plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], s = 100, c =
'blue', label = 'Cluster 2')
plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans == 2, 1], s = 100, c =
'green', label = 'Cluster 3')
plt.scatter(X[y_kmeans == 3, 0], X[y_kmeans == 3, 1], s = 100, c =
'cyan', label = 'Cluster 4')
```

```
plt.scatter(X[y_kmeans == 4, 0], X[y_kmeans == 4, 1], s = 100, c =
'magenta', label = 'Cluster 5')
plt.scatter(kmeans.cluster_centers_[:, 0],
kmeans.cluster_centers_[:, 1], s = 300, c = 'yellow', label =
'Centroids')
plt.title('Clusters of customers')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.show()
```

OUTPUT:

	CustomerID	Genre	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





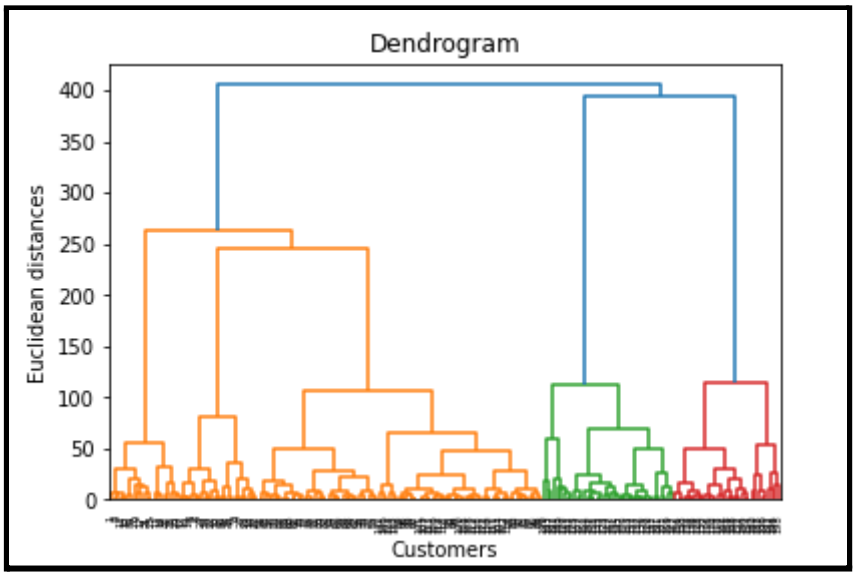
Hierarchical Clustering

CODE:

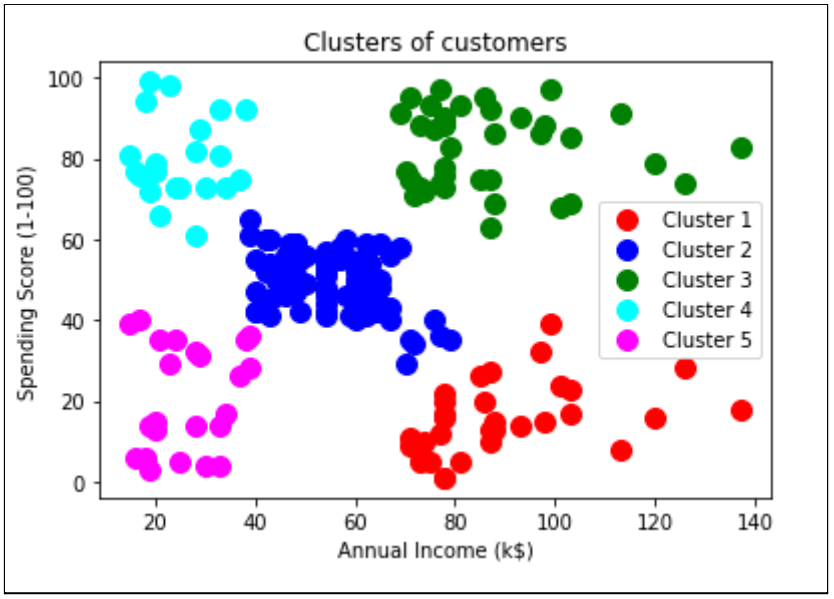
```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('Mall_Customers.csv')
X = dataset.iloc[:, [3, 4]].values
len(X)

# Using the dendrogram to find the optimal number of clusters
import scipy.cluster.hierarchy as sch
dendrogram = sch.dendrogram(sch.linkage(X, method = 'ward'))
plt.title('Dendrogram')
plt.xlabel('Customers')
plt.ylabel('Euclidean distances')
plt.show()

# Training the Hierarchical Clustering model on the dataset
from sklearn.cluster import AgglomerativeClustering
hc = AgglomerativeClustering(n_clusters = 5, affinity =
'euclidean', linkage = 'ward')
y_hc = hc.fit_predict(X)
```

```
print(y_hc)

# Visualising the clusters
plt.scatter(X[y_hc == 0, 0], X[y_hc == 0, 1], s = 100, c = 'red',
label = 'Cluster 1')
plt.scatter(X[y_hc == 1, 0], X[y_hc == 1, 1], s = 100, c = 'blue',
label = 'Cluster 2')
plt.scatter(X[y_hc == 2, 0], X[y_hc == 2, 1], s = 100, c =
'green', label = 'Cluster 3')
plt.scatter(X[y_hc == 3, 0], X[y_hc == 3, 1], s = 100, c = 'cyan',
label = 'Cluster 4')
plt.scatter(X[y_hc == 4, 0], X[y_hc == 4, 1], s = 100, c =
'magenta', label = 'Cluster 5')
plt.title('Clusters of customers')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.show()
```

OUTPUT:




```
[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 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]
```


Clusters of customers
Spending Score (1-100)
Annual Income (k$)
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Cluster 5
0
20
40
60
80
100
120
140

PART B

K Means
CODE:

```
import pandas as pd

data = pd.read_csv("driver-data.csv", index_col="id")
data.head()

from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=4)

kmeans.fit(data)

kmeans.cluster_centers_

kmeans.labels_

import numpy as np

unique, counts = np.unique(kmeans.labels_, return_counts=True)

dict_data = dict(zip(unique, counts))
dict_data

import seaborn as sns

data["cluster"] = kmeans.labels_

sns.pairplot(data)

kmeans.inertia_

kmeans.score

data


from sklearn import metrics
```

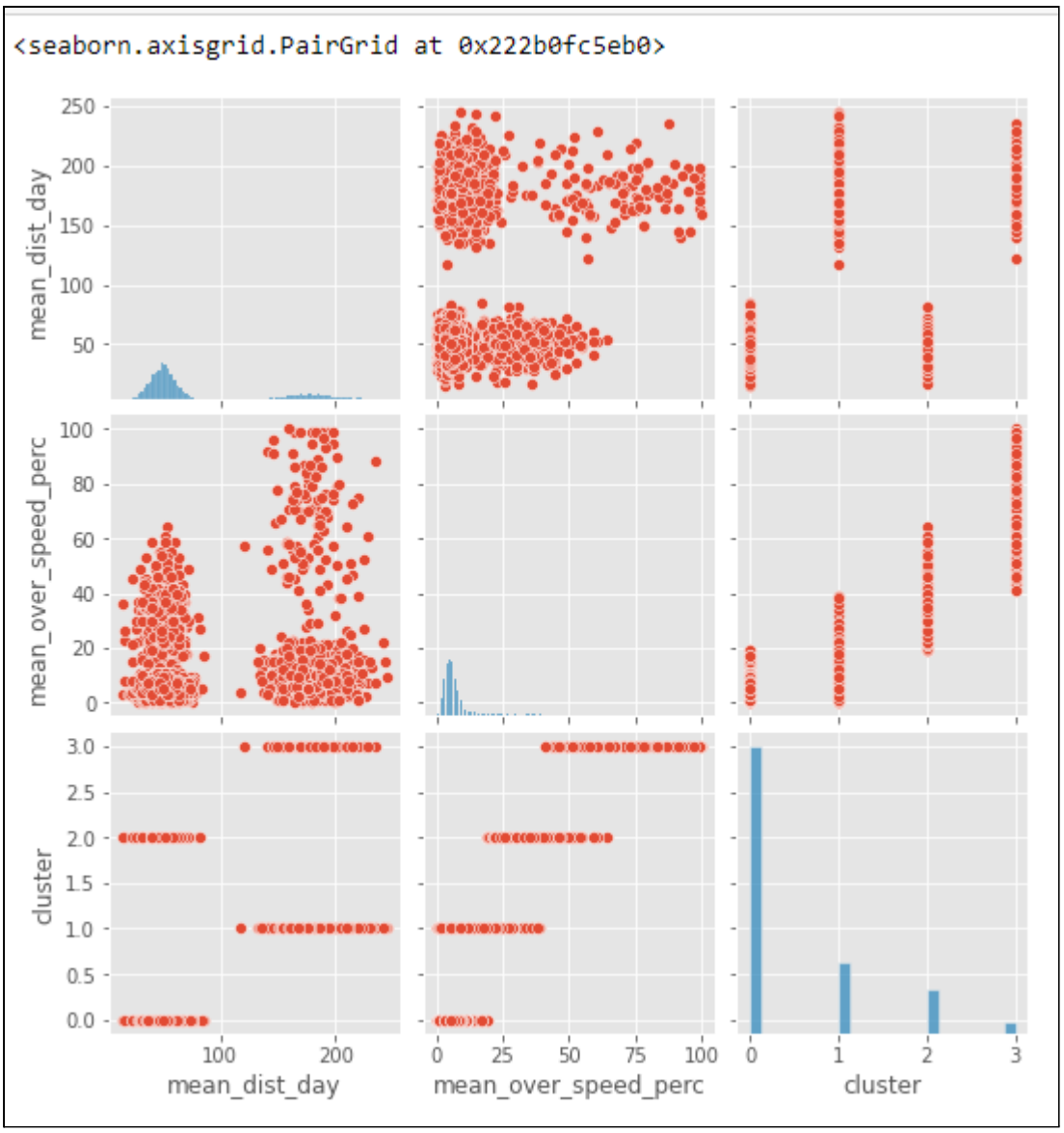
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import style
import pandas as pd

style.use('ggplot')

class K_Means:
     def __init__(self, k =3, tolerance = 0.0001, max_iterations =
500):
          self.k = k
          self.tolerance = tolerance
          self.max_iterations = max_iterations

     def fit(self, data):

          self.centroids = {}

          #initialize the centroids, the first 'k' elements in
the dataset will be our initial centroids
          for i in range(self.k):
               self.centroids[i] = data[i]

          #begin iterations
          for i in range(self.max_iterations):
               self.classes = {}
               for i in range(self.k):
                    self.classes[i] = []

               #find the distance between the point and cluster;
choose the nearest centroid
               for features in data:
                    distances = [np.linalg.norm(features -
self.centroids[centroid]) for centroid in self.centroids]
                    classification =
distances.index(min(distances))

self.classes[classification].append(features)

               previous = dict(self.centroids)
```

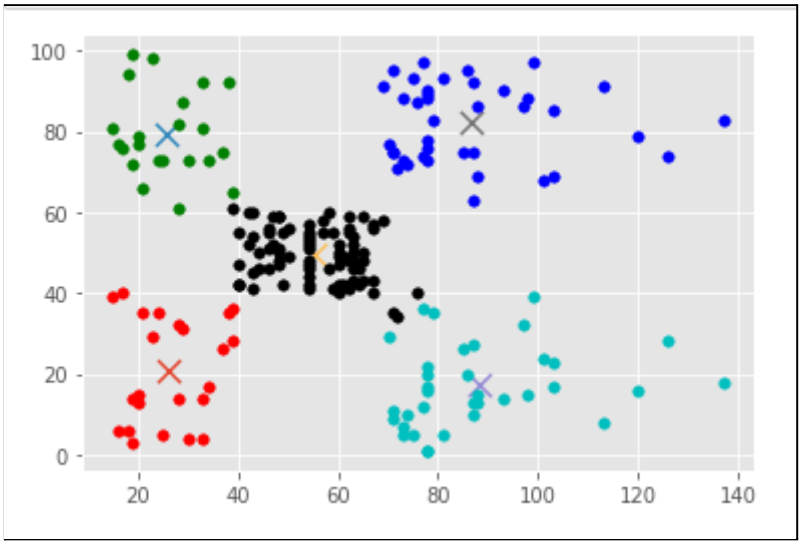
```
                #average the cluster datapoints to re-calculate
the centroids
                for classification in self.classes:
                    self.centroids[classification] =
np.average(self.classes[classification], axis = 0)

                isOptimal = True

                for centroid in self.centroids:

                    original_centroid = previous[centroid]
                    curr = self.centroids[centroid]

                    if np.sum((curr -
original_centroid)/original_centroid * 100.0) > self.tolerance:
                        isOptimal = False

                #break out of the main loop if the results are
optimal, ie. the centroids don't change their positions much(more
than our tolerance)
                if isOptimal:
                    break

    def pred(self, data):
        distances = [np.linalg.norm(data -
self.centroids[centroid]) for centroid in self.centroids]
        classification = distances.index(min(distances))
        return classification

def main():

    df = pd.read_csv("Mall_Customers.csv")
    df = X = df.iloc[:, [3, 4]]
    dataset = df.astype(float).values.tolist()

    X = df.values #returns a numpy array

    km = K_Means(5)
    km.fit(X)

    # Plotting starts here
    colors = 10*["r", "g", "c", "b", "k"]
```

```
      for centroid in km.centroids:
            plt.scatter(km.centroids[centroid][0],
km.centroids[centroid][1], s = 130, marker = "x")

      for classification in km.classes:
            color = colors[classification]
            for features in km.classes[classification]:
                  plt.scatter(features[0], features[1], color =
color,s = 30)

      plt.show()


if __name__ == "__main__":
      main()
```

OUTPUT:

id	mean_dist_day	mean_over_speed_perc
3423311935	71.24	28
3423313212	52.53	25
3423313724	64.54	27
3423311373	55.69	22
3423310999	54.58	25


<seaborn.axisgrid.PairGrid at 0x222b0fc5eb0>
mean_dist_day
mean_over_speed_perc
cluster
mean_dist_day
mean_over_speed_perc
cluster



Hierarchical Clustering

CODE:

```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns

# Importing the dataset
dataset = pd.read_csv('Mall_Customers.csv')
X = dataset.iloc[:, [3, 4]].values
X

new_data = dataset
new_data = new_data.drop('CustomerID', axis=1)
new_data

sns.pairplot(dataset)

from sklearn.preprocessing import LabelEncoder
new_data = new_data.apply(LabelEncoder().fit_transform)

X = new_data.to_numpy()

class Distance_computation_grid(object):
    '''
        class to enable the Computation of distance matrix
    '''
    def __init__(self):
        pass

    def compute_distance(self,samples):
        '''
            Creates a matrix of distances between individual
samples and clusters attained at a particular step
        '''
        Distance_mat = np.zeros((len(samples),len(samples)))
        for i in range(Distance_mat.shape[0]):
            for j in range(Distance_mat.shape[0]):
                if i!=j:
```

```
                    Distance_mat[i,j] =
float(self.distance_calculate(samples[i],samples[j]))
                else:
                    Distance_mat[i,j] = 10**4
        return Distance_mat


    def distance_calculate(self,sample1,sample2):
        '''
            Distance calulated between two samples. The two
samples can be both samples, both clusters or
            one cluster and one sample. If both of them are
samples/clusters, then simple norm is used. In other
            cases, we refer it as an exception case and pass the
samples as parameter to some function that
            calculates the necessary distance between cluster and
a sample
        '''
        dist = []
        for i in range(len(sample1)):
            for j in range(len(sample2)):
                try:

dist.append(np.linalg.norm(np.array(sample1[i])-np.array(sample2[j
]])))
                except:

dist.append(self.intersampledist(sample1[i],sample2[j]))
        return min(dist)


    def intersampledist(self,s1,s2):
        '''
            To be used in case we have one sample and one cluster
. It takes the help of one
            method 'interclusterdist' to compute the distances
between elements of a cluster(which are
            samples) and the actual sample given.
        '''
        if str(type(s2[0]))!='<class \'list\'>':
            s2=[s2]
        if str(type(s1[0]))!='<class \'list\'>':
```

```
            s1=[s1]
        m = len(s1)
        n = len(s2)
        dist = []
        if n>=m:
            for i in range(n):
                for j in range(m):
                    if (len(s2[i])>=len(s1[j])) and
str(type(s2[i][0])!='<class \'list\'>'):

dist.append(self.interclusterdist(s2[i],s1[j]))
                    else:

dist.append(np.linalg.norm(np.array(s2[i])-np.array(s1[j])))
        else:
            for i in range(m):
                for j in range(n):
                    if (len(s1[i])>=len(s2[j])) and
str(type(s1[i][0])!='<class \'list\'>'):

dist.append(self.interclusterdist(s1[i],s2[j]))
                    else:

dist.append(np.linalg.norm(np.array(s1[i])-np.array(s2[j])))
        return min(dist)

    def interclusterdist(self,cl,sample):
        if sample[0]!='<class \'list\'>':
            sample = [sample]
        dist   = []
        for i in range(len(cl)):
            for j in range(len(sample)):

dist.append(np.linalg.norm(np.array(cl[i])-np.array(sample[j])))
        return min(dist)

progression = [[i] for i in range(X.shape[0])]
samples     = [[list(X[i])] for i in range(X.shape[0])][:10]
m = len(samples)
distcal  = Distance_computation_grid()

while m>2:
```

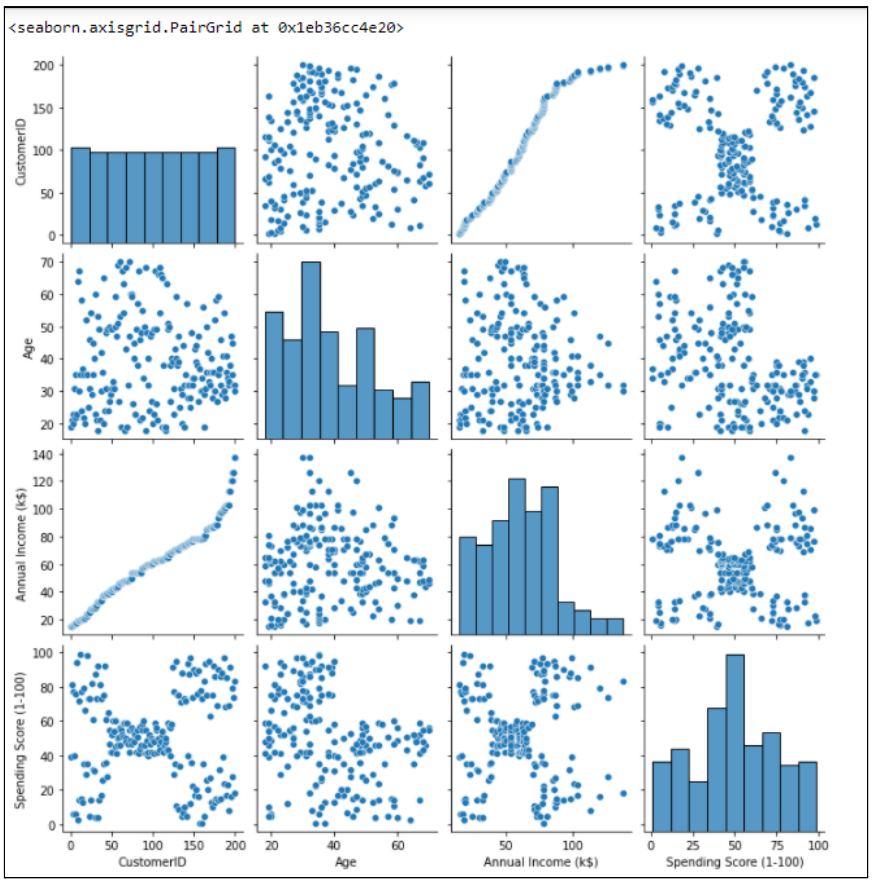
```
    print('Sample size before clustering    :- ',m)
    Distance_mat      = distcal.compute_distance(samples)
    sample_ind_needed =
np.where(Distance_mat==Distance_mat.min())[0]
    value_to_add      = samples.pop(sample_ind_needed[1])
    samples[sample_ind_needed[0]].append(value_to_add)

    print('Cluster Node 1
:-',progression[sample_ind_needed[0]])
    print('Cluster Node 2
:-',progression[sample_ind_needed[1]])


progression[sample_ind_needed[0]].append(progression[sample_ind_ne
eded[1]])
    progression[sample_ind_needed[0]] =
[progression[sample_ind_needed[0]]]
    v = progression.pop(sample_ind_needed[1])
    m = len(samples)

    print('Progression(Current Sample)      :-',progression)
    print('Cluster attained
:-',progression[sample_ind_needed[0]])
    print('Sample size after clustering     :-',m)
    print('\n')

from scipy.cluster.hierarchy import dendrogram, linkage
from matplotlib import pyplot as plt
Z = linkage(X, 'single')
fig = plt.figure(figsize=(8, 8))
plt.title('Dendrogram')

dn = dendrogram(Z)

plt.scatter(X[:,2], X[:,3], cmap="rainbow")

from sklearn.cluster import AgglomerativeClustering
aggclus = AgglomerativeClustering().fit(X)
aggclus.labels_
```

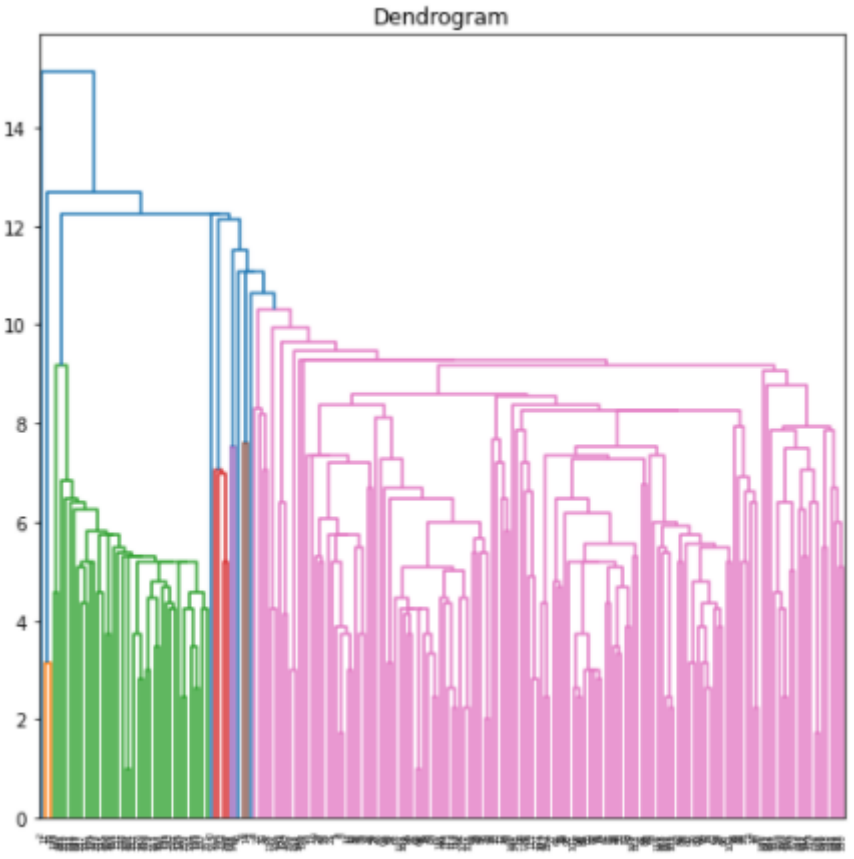
OUTPUT:

```
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],
```

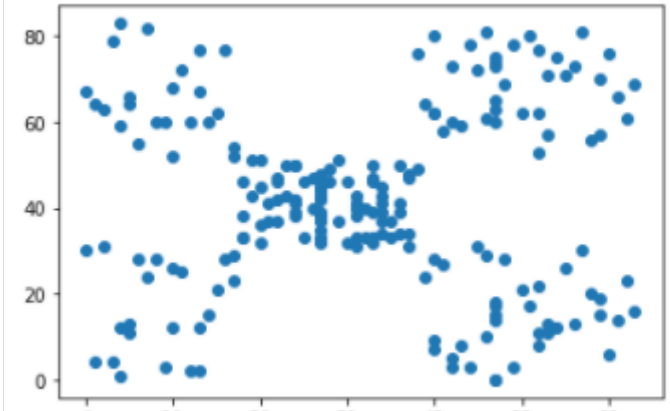
	Genre	Age	Annual Income (k$)	Spending Score (1-100)
0	Male	19	15	39
1	Male	21	15	81
2	Female	20	16	6
3	Female	23	16	77
4	Female	31	17	40
...	...	...	...	...
195	Female	35	120	79
196	Female	45	126	28
197	Male	32	126	74
198	Male	32	137	18
199	Male	30	137	83

200 rows × 4 columns


<seaborn.axisgrid.PairGrid at 0x1eb36cc4e20>
CustomerID
Age
Annual Income (k$)
Spending Score (1-100)


Dendrogram

```
<matplotlib.collections.PathCollection at 0x1eb365528e0>
```



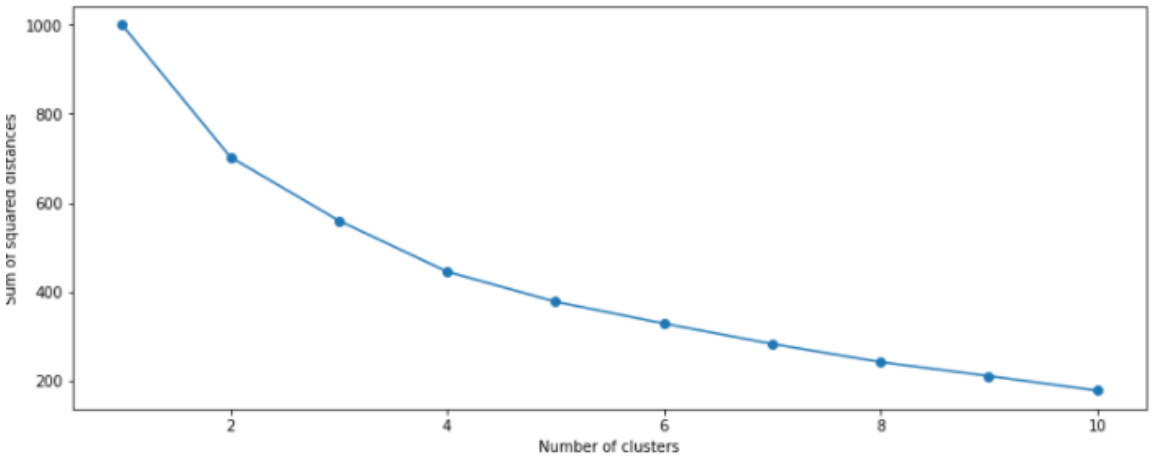

```
array([0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1,
       0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1,
       0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1,
       0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1,
       0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0,
       0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1,
       1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1,
       0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1,
       0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1,
       0, 1], dtype=int64)
```

PART C

Code:

```
from sklearn import datasets, preprocessing
from sklearn.preprocessing import LabelEncoder
from sklearn.cluster import KMeans

df=pd.read_csv('Mall_Customers.csv')
df = df.apply(LabelEncoder().fit_transform)

scaler = preprocessing.StandardScaler()
scaled_df = scaler.fit_transform(df)
pd.DataFrame(scaled_df).describe()
clusters = range(1, 11)
sse=[]
for cluster in clusters:
    model = KMeans(n_clusters=cluster, init='k-means++',
max_iter=300, tol=0.0001, verbose=0,random_state=0)
    model.fit(scaled_df)
    sse.append(model.inertia_)
sse_df = pd.DataFrame(np.column_stack((clusters, sse)),
columns=['cluster', 'SSE'])
fig, ax = plt.subplots(figsize=(13, 5))
ax.plot(sse_df['cluster'], sse_df['SSE'], marker='o')
ax.set_xlabel('Number of clusters')
```

Output:



Conclusion:

Clustering is a method of partitioning a set of data or objects into a set of significant subclasses called clusters. Elbow graph is used to find the optimal value of k, no of clusters.

Kmeans clustering is one of the most popular clustering algorithms and usually the first thing practitioners apply when solving clustering tasks to get an idea of the structure of the dataset. The goal of k means is to group data points into distinct non-overlapping subgroups. It doesn’t learn the number of clusters from the data and requires it to be pre-defined.

Hierarchical clustering is a powerful technique that allows you to build tree structures from data similarities. We can see how different sub-clusters relate to each other, and how far apart data points are. The advantage of not having to pre-define the number of clusters gives it quite an edge over kMeans. However, it doesn't work well when we have a huge amount of data.