COSC 3337 : Data Science I

N. Rizk

College of Natural and Applied Sciences

Department of Computer Science

University of Houston

K-means Clustering

- Partitioning Clustering Approach
  - a typical clustering analysis approach via iteratively partitioning training data set to learn a partition of the given data space
  - learning a partition on a data set to produce several non-empty clusters (usually, the number of clusters given in advance)
  - in principle, optimal partition achieved via minimising the sum of squared distance to its "representative object" in each cluster

$$E = \sum_{k=1}^{K} \sum_{\mathbf{x} \in C_k} d^2(\mathbf{x}, \mathbf{m}_k)$$

e.g., Euclidean distance $d^2(\mathbf{x}, \mathbf{m}_k) = \sum_{n=1}^{N} (x_n - m_{kn})^2$

Introduction

- Given a *K*, find a partition of *K clusters* to optimise the chosen partitioning criterion (cost function)
  - global optimum: exhaustively search all partitions
- The *K-means* algorithm: a heuristic method
  - K-means algorithm (MacQueen'67): each cluster is represented by the centre of the cluster and the algorithm converges to stable centriods of clusters.
  - K-means algorithm is the simplest partitioning method for clustering analysis and widely used in data mining applications.

K-means Algorithm

- Given the cluster number *K*, the *K-means* algorithm is carried out in three steps after initialisation:

Initialisation: set seed points (randomly)

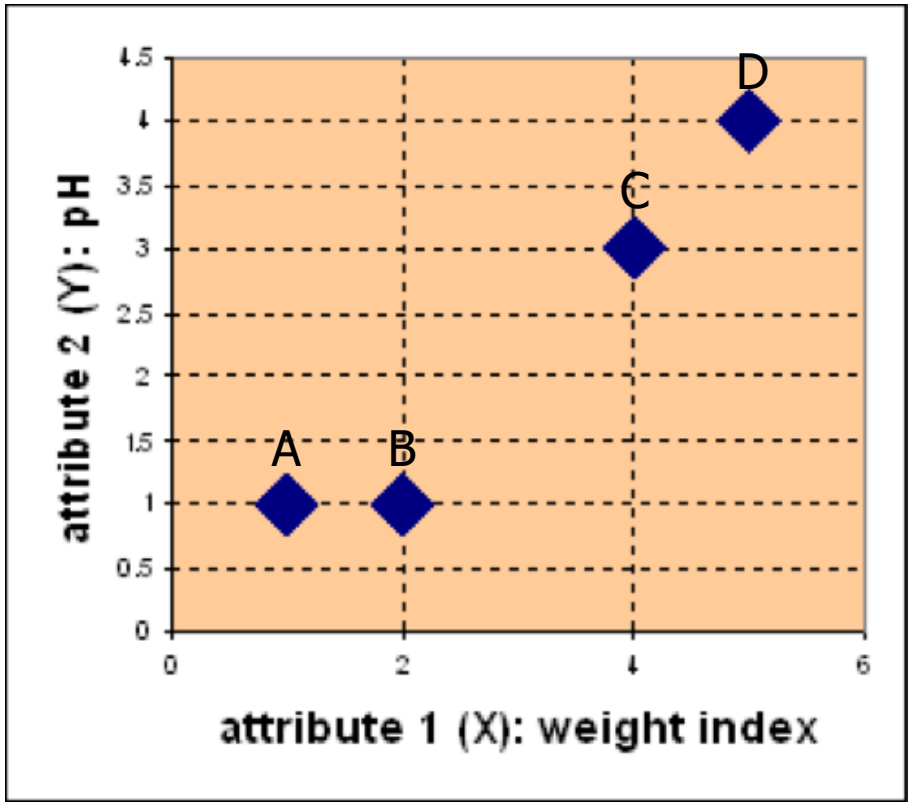
1) Assign each object to the cluster of the nearest seed point measured with a specific distance metric
2) Compute new seed points as the centroids of the clusters of the current partition (the centroid is the centre, i.e., *mean point*, of the cluster)
3) Go back to Step 1), stop when no more new assignment (i.e., membership in each cluster no longer changes)

Example

- Problem

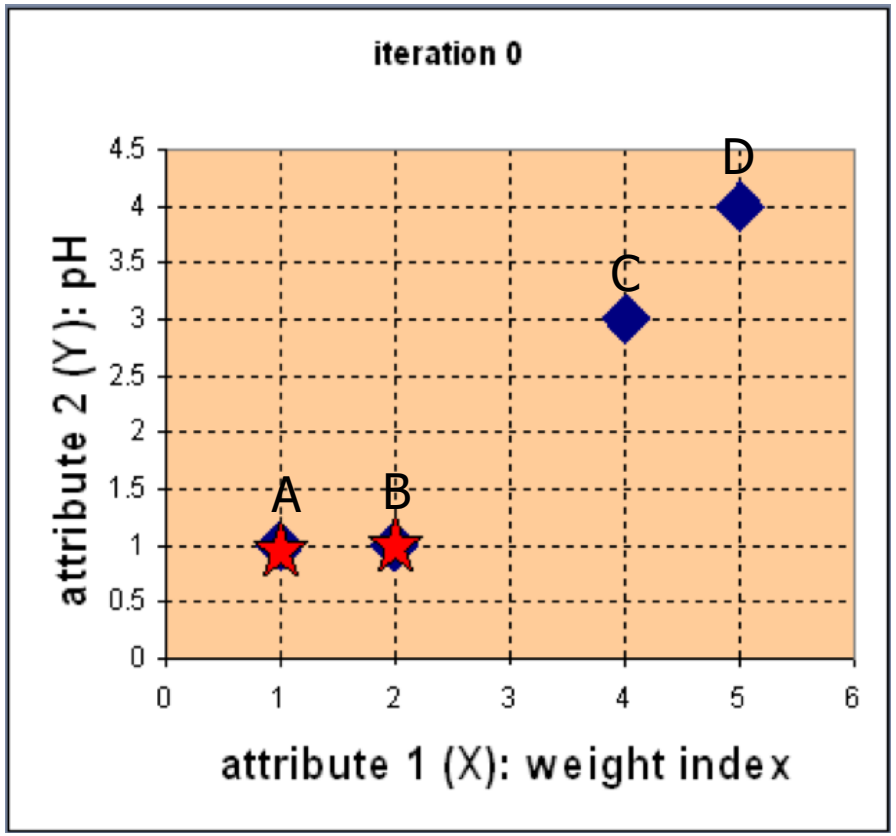
  Suppose we have 4 types of medicines and each has two attributes (pH and weight index). Our goal is to group these objects into *K=2* group of medicine.

Medicine	Weight	pH-Index
A	1	1
B	2	1
C	4	3
D	5	4



Example

- Step 1: Use initial seed points for partitioning



$$c_1 = A, c_2 = B$$

$$D^0 = \begin{bmatrix} 0 & 1 & 3.61 & 5 \\ 1 & 0 & 2.83 & 4.24 \end{bmatrix} \quad \begin{matrix} c_1 = (1,1) & group-1 \\ c_2 = (2,1) & group-2 \end{matrix}$$

$$\begin{matrix} A & B & C & D \end{matrix}$$

$$\begin{bmatrix} 1 & 2 & 4 & 5 \\ 1 & 1 & 3 & 4 \end{bmatrix} \quad \begin{matrix} X \\ Y \end{matrix}$$

Euclidean distance

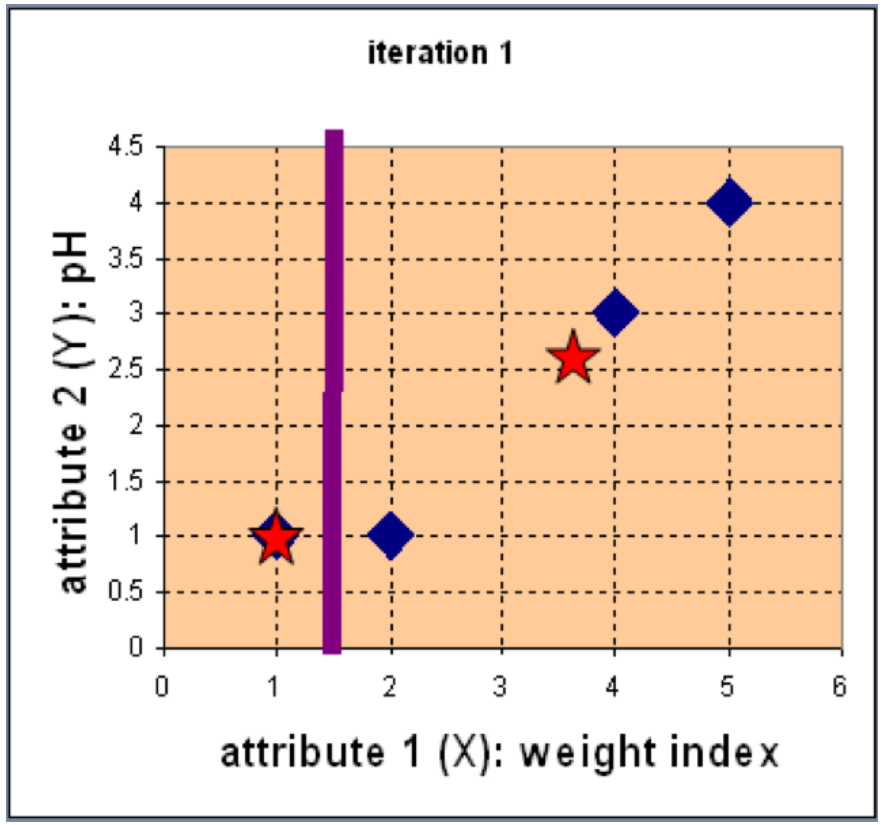
$$d(D,c_1) = \sqrt{(5-1)^2 + (4-1)^2} = 5$$

$$d(D,c_2) = \sqrt{(5-2)^2 + (4-1)^2} = 4.24$$

Assign each object to the cluster with the nearest seed point

Example

- Step 2: Compute new centroids of the current partition



Knowing the members of each cluster, now we compute the new centroid of each group based on these new memberships.

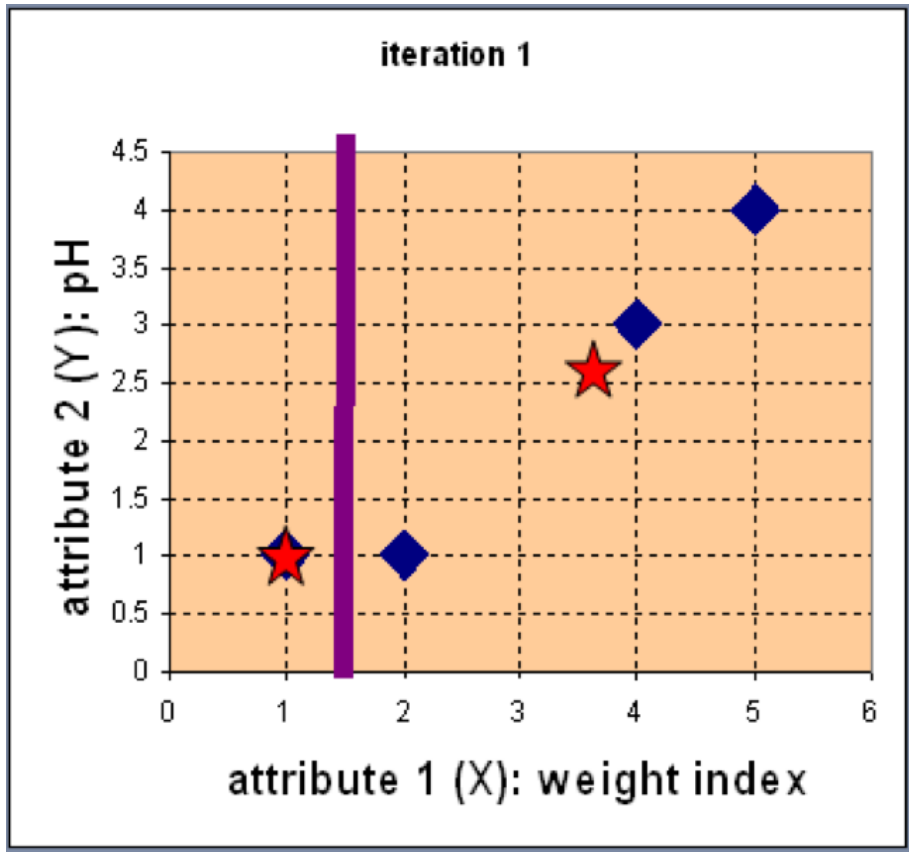
$$c_1 = (1,\ 1)$$

$$c_2 = \left(\frac{2+4+5}{3},\ \frac{1+3+4}{3}\right)$$

$$= \left(\frac{11}{3},\ \frac{8}{3}\right)$$

Example

- Step 2: Renew membership based on new centroids



Compute the distance of all objects to the new centroids

$$D^1 = \begin{bmatrix} 0 & 1 & 3.61 & 5 \\ 3.14 & 2.36 & 0.47 & 1.89 \end{bmatrix} \quad \begin{matrix} c_1 = (1,1) & group-1 \\ c_2 = (\frac{11}{3}, \frac{8}{3}) & group-2 \end{matrix}$$

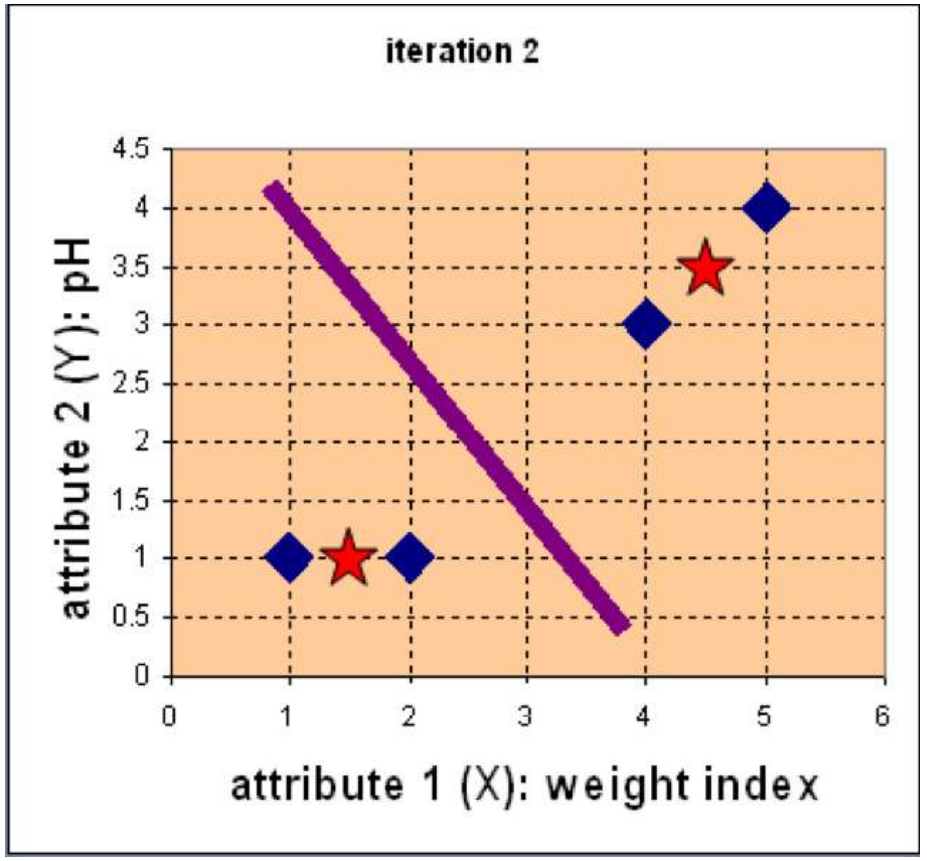
$$\begin{matrix} A & B & C & D \end{matrix}$$

$$\begin{bmatrix} 1 & 2 & 4 & 5 \\ 1 & 1 & 3 & 4 \end{bmatrix} \begin{matrix} X \\ Y \end{matrix}$$

Assign the membership to objects

Example

- Step 3: Repeat the first two steps until its convergence



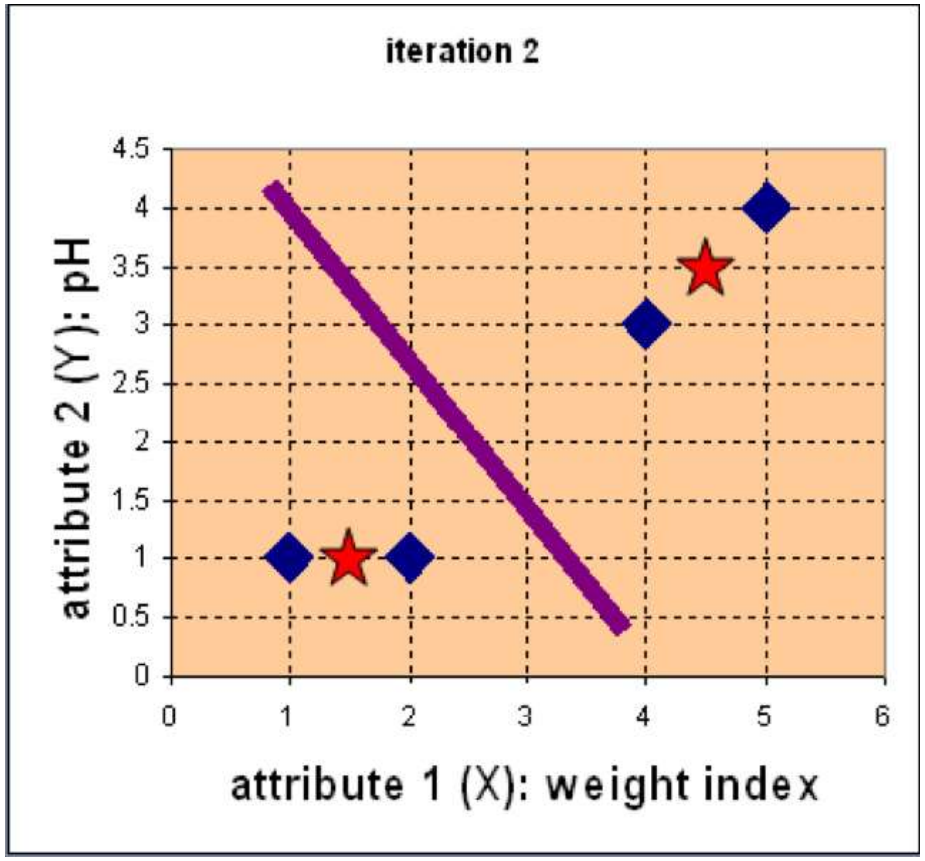
Knowing the members of each cluster, now we compute the new centroid of each group based on these new memberships.

$$c_1 = \left(\frac{1+2}{2}, \frac{1+1}{2}\right) = (1\frac{1}{2}, 1)$$

$$c_2 = \left(\frac{4+5}{2}, \frac{3+4}{2}\right) = (4\frac{1}{2}, 3\frac{1}{2})$$

Example

- Step 3: Repeat the first two steps until its convergence



Compute the distance of all objects to the new centroids

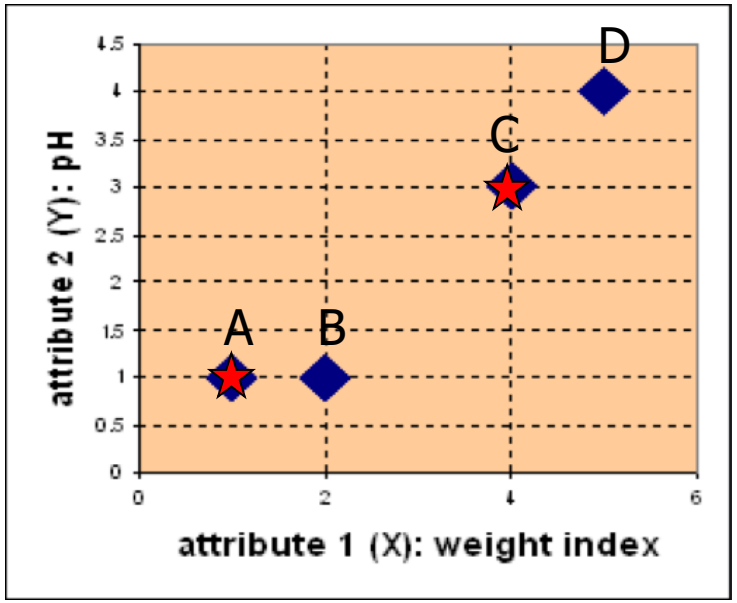
Stop due to no new assignment
Membership in each cluster no longer change

Exercise

For the medicine data set, use K-means with the Manhattan distance metric for clustering analysis by setting $K=2$ and initialising seeds as C_1 = A and C_2 = C. Answer three questions as follows:

1. How many steps are required for convergence?
2. What are memberships of two clusters after convergence?
3. What are centroids of two clusters after convergence?

Medicine	Weight	pH-Index
A	1	1
B	2	1
C	4	3
D	5	4



Detailed Example

Sample Data set

Objects	X	Y	Z
OB-1	1	4	1
OB-2	1	2	2
OB-3	1	4	2
OB-4	2	1	2
OB-5	1	1	1
OB-6	2	4	2
OB-7	1	1	2
OB-8	2	1	1

Task is to cluster these objects into two clusters

Algorithm steps

- Taking any two centroids or data points (as you took 2 as K hence the number of centroids also 2) in its account initially.
- After choosing the centroids, (say C1 and C2) the data points (coordinates here) are assigned to any of the Clusters (let's take centroids = clusters for the time being) depending upon the distance between them and the centroids.
- Assume that the algorithm chose OB-2 (1,2,2) and OB-6 (2,4,2) as centroids and cluster 1 and cluster 2 as well.
- measuring the distances

$$d=|x2-x1|+|y2-y1|+|z2-z1| \text{ (Manhathan Distance)}$$

Step 1:calculation of distances between the objects and centroids (OB-2 and OB-6):

Objects	X	Y	Z	Distance from C1(1,2,2)	Distance from C2(2,4,2)
OB-1	1	4	1	3	2
OB-2	1	2	2	0	3
OB-3	1	4	2	2	1
OB-4	2	1	2	2	3
OB-5	1	1	1	2	5
OB-6	2	4	2	3	0
OB-7	1	1	2	1	4
OB-8	2	1	1	3	4

Step 2: Cluster Formation

Objects	X	Y	Z	Distance from C1(1,2,2)	Distance from C2(2,4,2)
OB-1	1	4	1	3	2
OB-2	1	2	2	0	3
OB-3	1	4	2	2	1
OB-4	2	1	2	2	3
OB-5	1	1	1	2	5
OB-6	2	4	2	3	0
OB-7	1	1	2	1	4
OB-8	2	1	1	3	4

Cluster 1
OB-2
OB-4
OB-5
OB-7
OB-8

Cluster 2
OB-1
OB-3
OB-6

An object which has a shorter distance between a centroid (say C1) than the other centroid (say C2) will fall into the cluster of C1.

updating cluster centroids

$$\frac{\sum_{i=1}^{n} x_i}{n}, \frac{\sum_{i=1}^{n} y_i}{n}, \frac{\sum_{i=1}^{n} z_i}{n}$$

New C1

the updated cluster 1 will be ((1+2+1+1+2)/5, (2+1+1+1+1)/5,(2+2+1+2+1)/5) = (1.4,1.2,1.6).

New C2

And for cluster 2 it will be ((1+1+2)/3, (4+4+4)/3, (1+2+2)/3) = (1.33, 4, 1.66).

Objects	X	Y	Z	Distance from C1(1.4,1.2, 1.6)	Distance from C2(1.33, 4, 1.66)
OB-1	1	4	1	3.8	1
OB-2	1	2	2	1.6	2.66
OB-3	1	4	2	3.6	0.66
OB-4	2	1	2	1.2	4
OB-5	1	1	1	1.2	4
OB-6	2	4	2	3.8	1
OB-7	1	1	2	1	3.66
OB-8	2	1	1	1.4	4.33

Objects	X	Y	Z	Distance from C1(1.4,1.2,1.6)	Distance from C2(1.33, 4, 1.66)
OB-1	1	4	1	3.8	1
OB-2	1	2	2	1.6	2.66
OB-3	1	4	2	3.6	0.66
OB-4	2	1	2	1.2	4
OB-5	1	1	1	1.2	4
OB-6	2	4	2	3.8	1
OB-7	1	1	2	1	3.66
OB-8	2	1	1	1.4	4.33

Cluster 1
OB-2
OB-4
OB-5
OB-7
OB-8

Cluster 2
OB-1
OB-3
OB-6

This is where the algorithm no longer updates the centroids. Because there is no change in the current cluster formation, it is the same as the previous formation.

Step 3: Apply test sets

Test Sets

Objects	X	Y	Z	Distance from C1(1.4,1.2, 1.6)	Distance from C2(1.33, 4, 1.66)
OB-1	2	4	1	4	0.73
OB-2	2	2	2	2.4	3.01
OB-3	1	2	1	1.6	2.99
OB-4	2	2	1	1.4	3.33

Cluster 1
OB-2
OB-3
OB-4

d=|x2–x1|+|y2–y1|+|z2–z1| (Manhathan Distance)

Cluster 2
OB-1

Step 4: Measuring performance

- Adjusted rand index
- Mutual information based scoring
- Homogeneity, completeness and v-measure

Kmeans Disadvatnages

- Requires to pre-specify the number of clusters (k)➔Hierarchical clustering is an alternative approach that does not require a particular choice of clusters.
- is sensitive to outliers and different results can occur if you change the ordering of the data.

- K-Means is a lazy learner where generalization of the training data is delayed until a query is made to the system.
- Learning methods can construct a different approximation or result to the target function for each encountered query.

- It is a good method for online learning, but it requires a possibly large amount of memory to store the data, and each request involves starting the identification of a local model from scratch.

Variants of *K*-mean

- There are several variants of *K*-means to overcome its weaknesses
  - *K*-Medoids: resistance to noise and/or outliers
  - *K*-Modes: extension to categorical data clustering analysis
  - CLARA: extension to deal with large data sets
  - Mixture models (EM algorithm): handling uncertainty of clusters

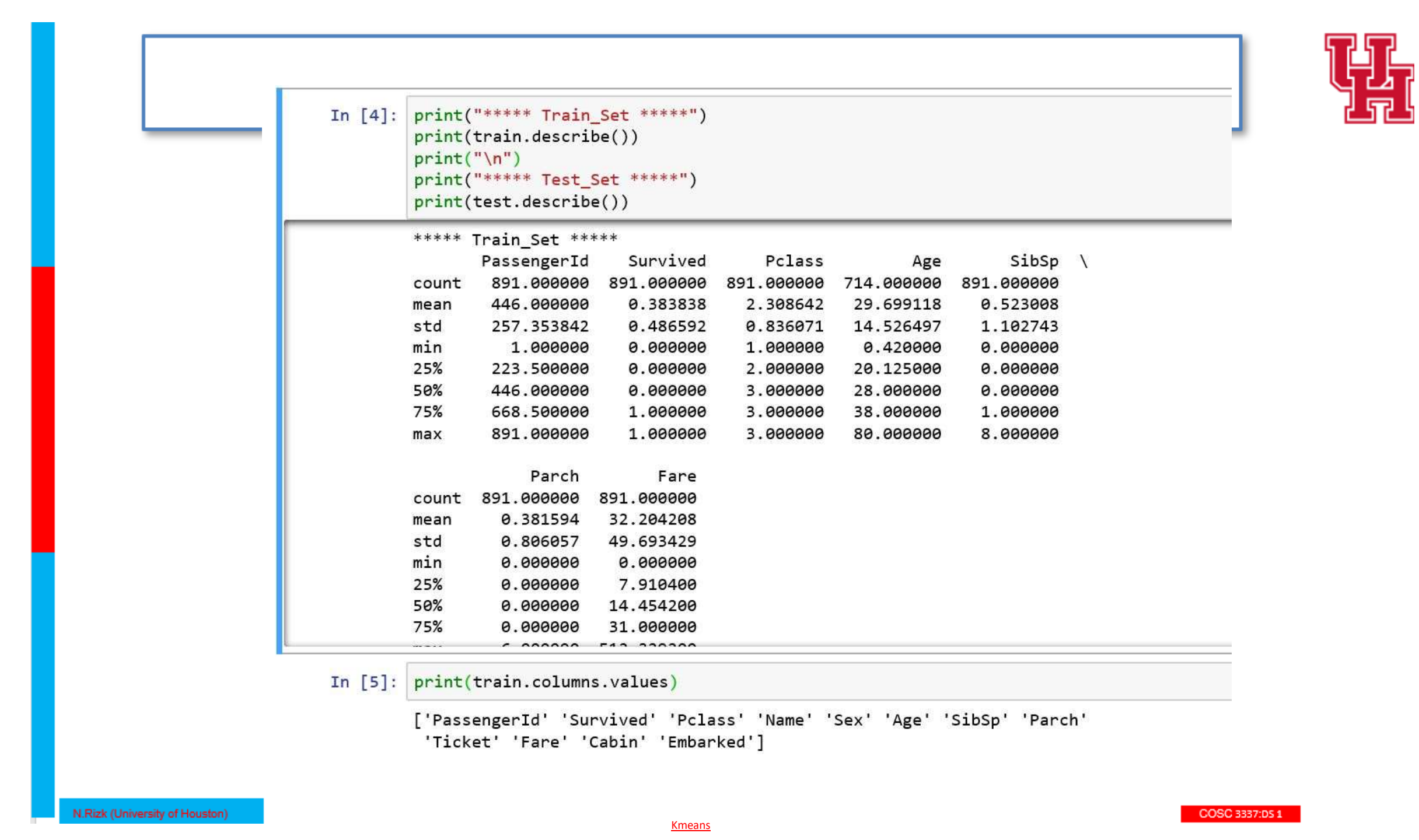
K-Means Clustering in Python with scikit-learn

In [1]:
```
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:
```
# Load the train and test datasets to create two DataFrames

train_url = "http://s3.amazonaws.com/assets.datacamp.com/course/Kaggle/train.csv"
train = pd.read_csv(train_url)
test_url = "http://s3.amazonaws.com/assets.datacamp.com/course/Kaggle/test.csv"
test = pd.read_csv(test_url)
```

In [3]:
```
print("***** Train_Set *****")
print(train.head())
print("\n")
print("***** Test_Set *****")
print(test.head())
```

```
***** Train_Set *****
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3
```



In [4]:
```
print("***** Train_Set *****")
print(train.describe())
print("\n")
print("***** Test_Set *****")
print(test.describe())
```

```
***** Train_Set *****
       PassengerId    Survived      Pclass         Age       SibSp  \
count   891.000000  891.000000  891.000000  714.000000  891.000000
mean    446.000000    0.383838    2.308642   29.699118    0.523008
std     257.353842    0.486592    0.836071   14.526497    1.102743
min       1.000000    0.000000    1.000000    0.420000    0.000000
25%     223.500000    0.000000    2.000000   20.125000    0.000000
50%     446.000000    0.000000    3.000000   28.000000    0.000000
75%     668.500000    1.000000    3.000000   38.000000    1.000000
max     891.000000    1.000000    3.000000   80.000000    8.000000

            Parch        Fare
count  891.000000  891.000000
mean     0.381594   32.204208
std      0.806057   49.693429
min      0.000000    0.000000
25%      0.000000    7.910400
50%      0.000000   14.454200
75%      0.000000   31.000000
```

In [5]:
```
print(train.columns.values)
```

```
['PassengerId' 'Survived' 'Pclass' 'Name' 'Sex' 'Age' 'SibSp' 'Parch'
 'Ticket' 'Fare' 'Cabin' 'Embarked']
```

```
# missing values For the train set
train.isna().head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	False	False	False	False	False	False	False	False	False	False	True	False
1	False	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	True	False
3	False	False	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	True	False

```
# For the test set
test.isna().head()
```

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	False	False	False	False	False	False	False	False	False	True	False
1	False	False	False	False	False	False	False	False	False	True	False
2	False	False	False	False	False	False	False	False	False	True	False
3	False	False	False	False	False	False	False	False	False	True	False
4	False	False	False	False	False	False	False	False	False	True	False

```
#total number of missing values
print("*****In the train set*****")
print(train.isna().sum())
print("\n")
print("*****In the test set*****")
print(test.isna().sum())
```

```
*****In the train set*****
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64


*****In the test set*****
PassengerId      0
Pclass           0
Name             0
Sex              0
Age             86
SibSp            0
Parch            0
Ticket           0
Fare             1
Cabin          327
Embarked         0
dtype: int64
```

```
# Fill missing values with mean column values in the train set
train.fillna(train.mean(), inplace=True)
```

```
# Fill missing values with mean column values in the test set
test.fillna(test.mean(), inplace=True)
```

```
print(train.isna().sum())
```

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age              0
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

```
print(test.isna().sum())
```

```
PassengerId      0
Pclass           0
Name             0
Sex              0
Age              0
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          327
Embarked         0
dtype: int64
```

```
#Categorical: Survived, Sex, and Embarked. Ordinal: Pclass.
#Continuous: Age, Fare. Discrete: SibSp, Parch.
#Ticket and Cabin. Ticket is a mix of numeric and alphanumeric data types. Cabin is alphanumeric
```

```
train['Ticket'].head()
```

```
0           A/5 21171
1            PC 17599
2    STON/O2. 3101282
3              113803
4              373450
Name: Ticket, dtype: object
```

```
train['Cabin'].head()
```

```
0     NaN
1     C85
2     NaN
3    C123
4     NaN
Name: Cabin, dtype: object
```

```
#Survival count with respect to Pclass:
train[['Pclass', 'Survived']].groupby(['Pclass'], as_index=False).mean().sort_values(by='Survived', ascending=False)
```

	Pclass	Survived
0	1	0.629630
1	2	0.472826
2	3	0.242363

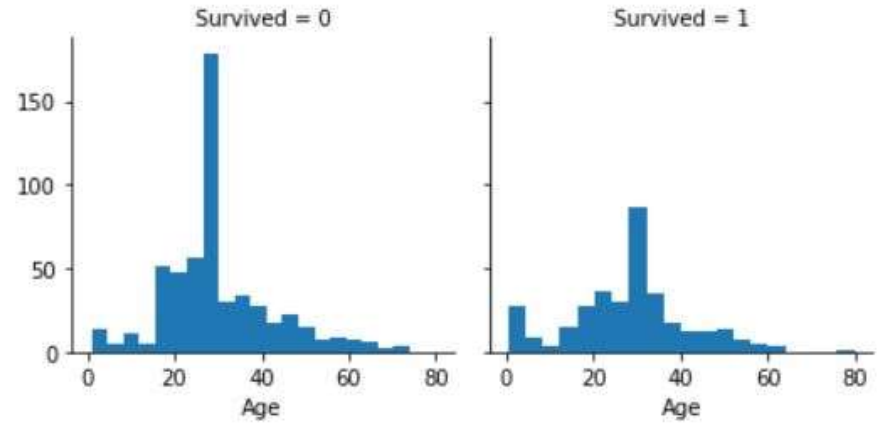
```
#Survival count with respect to Sex:
train[["Sex", "Survived"]].groupby(['Sex'], as_index=False).mean().sort_values(by='Survived', ascending=False)
```

	Sex	Survived
0	female	0.742038
1	male	0.188908

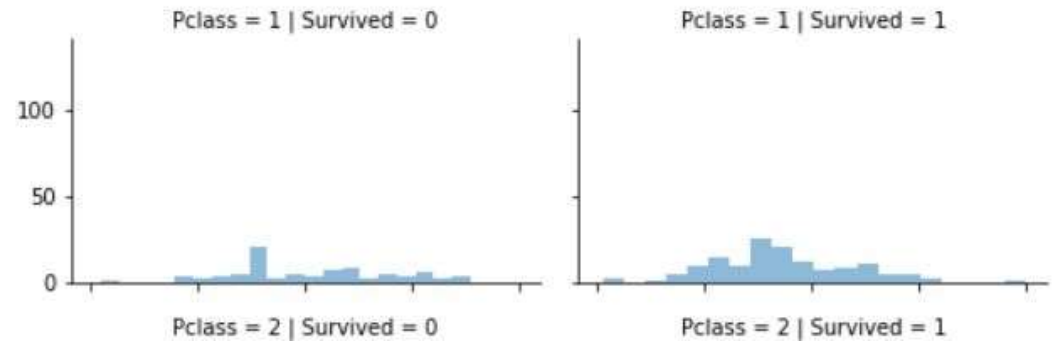
```
#Survival count with respect to SibSp:
train[["SibSp", "Survived"]].groupby(['SibSp'], as_index=False).mean().sort_values(by='Survived', ascending=False)
```

```
g = sns.FacetGrid(train, col='Survived')
g.map(plt.hist, 'Age', bins=20)
```

```
<seaborn.axisgrid.FacetGrid at 0x1eb47de8d68>
```




```
grid = sns.FacetGrid(train, col='Survived', row='Pclass', size=2.2, aspect=1.6)
grid.map(plt.hist, 'Age', alpha=.5, bins=20)
grid.add_legend();
```



```
# build a K-Mean
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
PassengerId    891 non-null int64
Survived       891 non-null int64
Pclass         891 non-null int64
Name           891 non-null object
Sex            891 non-null object
Age            891 non-null float64
SibSp          891 non-null int64
Parch          891 non-null int64
Ticket         891 non-null object
Fare           891 non-null float64
Cabin          204 non-null object
Embarked       889 non-null object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.6+ KB
```

```
#feature engineering, i.e. features like Name, Ticket, Cabin and
# Embarked do not have any impact on the survival status of the passengers.
train = train.drop(['Name','Ticket', 'Cabin','Embarked'], axis=1)
test = test.drop(['Name','Ticket', 'Cabin','Embarked'], axis=1)
```

```
# the only none numeric value is sex let's convert it
labelEncoder = LabelEncoder()
labelEncoder.fit(train['Sex'])
labelEncoder.fit(test['Sex'])
train['Sex'] = labelEncoder.transform(train['Sex'])
test['Sex'] = labelEncoder.transform(test['Sex'])
```

```
# Let's investigate if you have non-numeric data left

train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 8 columns):
PassengerId    891 non-null int64
Survived       891 non-null int64
Pclass         891 non-null int64
Sex            891 non-null int64
Age            891 non-null float64
SibSp          891 non-null int64
Parch          891 non-null int64
Fare           891 non-null float64
dtypes: float64(2), int64(6)
memory usage: 55.8 KB
```

```
#the label survived should be dropped , the test set does not have one anyway
test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 7 columns):
PassengerId    418 non-null int64
Pclass         418 non-null int64
Sex            418 non-null int64
Age            418 non-null float64
SibSp          418 non-null int64
Parch          418 non-null int64
Fare           418 non-null float64
dtypes: float64(2), int64(5)
memory usage: 22.9 KB
```

```
#drop Survival column from train
X = np.array(train.drop(['Survived'], axis=1))
```

```
y = np.array(train['Survived'])
```

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 8 columns):
PassengerId    891 non-null int64
Survived       891 non-null int64
Pclass         891 non-null int64
Sex            891 non-null int64
Age            891 non-null float64
SibSp          891 non-null int64
Parch          891 non-null int64
Fare           891 non-null float64
dtypes: float64(2), int64(6)
memory usage: 55.8 KB
```

```
kmeans = KMeans(n_clusters=2) # You want cluster the passenger records into 2: Survived or Not survived
kmeans.fit(X)
```

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=2, n_init=10, n_jobs=1, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=0)
```

```
# how many are predicted correctly based on Y
correct = 0
for i in range(len(X)):
    predict_me = np.array(X[i].astype(float))
    predict_me = predict_me.reshape(-1, len(predict_me))
    prediction = kmeans.predict(predict_me)
    if prediction[0] == y[i]:
        correct += 1
print(correct/len(X))
```

```
0.5084175084175084
```

```
#model was able to cluster correctly with a 50%  accuracy
#tweak some parameters of the model itself to enhance the model accuracy such as algorithm, max_ter, n-jobs
kmeans = kmeans = KMeans(n_clusters=2, max_iter=600, algorithm = 'auto')
kmeans.fit(X)
```

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=600,
    n_clusters=2, n_init=10, n_jobs=1, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=0)
```

```
correct = 0
for i in range(len(X)):
    predict_me = np.array(X[i].astype(float))
    predict_me = predict_me.reshape(-1, len(predict_me))
    prediction = kmeans.predict(predict_me)
    if prediction[0] == y[i]:
        correct += 1

print(correct/len(X))
```

```
0.5084175084175084
```

```
#a slight change because we have not scaled the values of the different features that we are feeding to the model
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)
```

```
kmeans.fit(X_scaled)
```

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=600,
    n_clusters=2, n_init=10, n_jobs=1, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=0)
```

```
#+12 % after scaling
correct = 0
for i in range(len(X)):
    predict_me = np.array(X[i].astype(float))
    predict_me = predict_me.reshape(-1, len(predict_me))
    prediction = kmeans.predict(predict_me)
    if prediction[0] == y[i]:
        correct += 1

print(correct/len(X))
```

```
0.6262626262626263
```

Determine optimal k

Centroid, Radius and Diameter of a Cluster

(for numerical data sets)

- Centroid: the "middle" of a cluster

$$C_m = \frac{\sum_{i=1}^{N}(t_{ip})}{N}$$

- Radius: square root of average distance from any point of the cluster to its centroid

$$R_m = \sqrt{\frac{\sum_{i=1}^{N}(t_{ip} - c_m)^2}{N}}$$

- Diameter: square root of average mean squared distance between all pairs of points in the cluster

$$D_m = \sqrt{\frac{\sum_{i=1}^{N}\sum_{i=1}^{N}(t_{ip} - t_{iq})^2}{N(N-1)}}$$

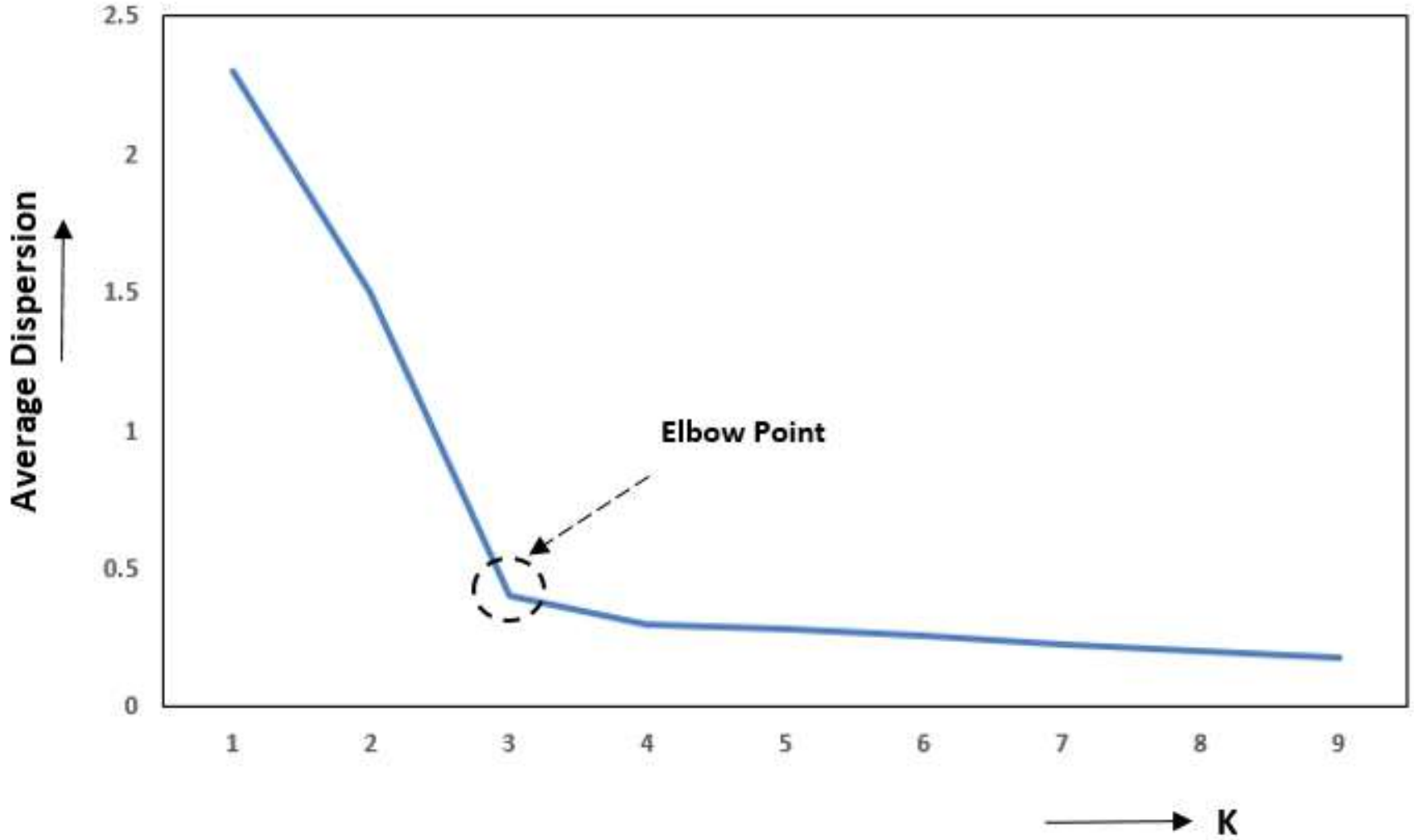
The elbow method is used to determine the optimal number of clusters in k-means clustering.

The elbow method plots the value of the cost function produced by different values of *k*.

If *k* increases, average distortion will decrease, each cluster will have fewer constituent instances, and the instances will be closer to their respective centroids.

However, the improvements in average distortion will decline as *k* increases. The value of *k* at which improvement in distortion declines the most is called the elbow, at which we should stop dividing the data into further clusters.

Elbow Method for selection of optimal "K" clusters



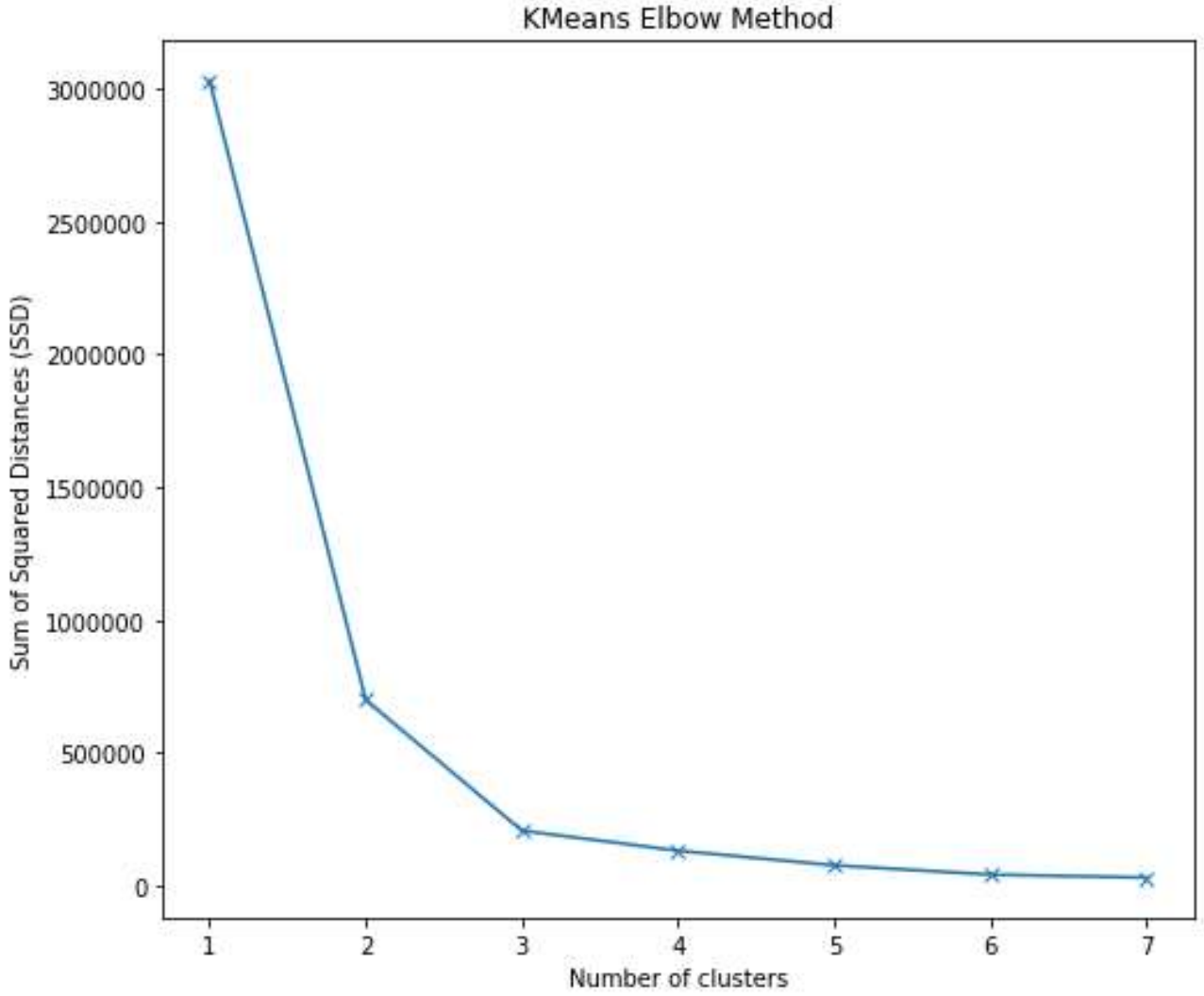
WSS:Cost Function/Objective

1.Compute clustering algorithm (e.g., k-means clustering) for different values of k. For instance, by varying k from 1 to 10 clusters.

2.For each k, calculate the **total within-cluster sum of square (wss)**.

3.Plot the curve of wss according to the number of clusters k.

4.The location of a bend (knee) in the plot is generally considered as an indicator of the appropriate number of clusters.



$$J(c_k) = \sum_{x_i \in c_k} ||x_i - \mu_k||^2$$

K-means optimization objective

→ $c^{(i)}$ = index of cluster (1,2,…,K) to which example $x^{(i)}$ is currently assigned

→ μ_k = cluster centroid k ($\mu_k \in \mathbb{R}^n$)

$\mu_{c^{(i)}}$ = cluster centroid of cluster to which example $x^{(i)}$ has been assigned

K $\quad k \in \{1,2,\ldots,K\}$

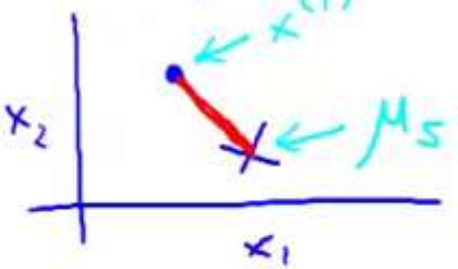
$x^{(i)} \rightarrow 5$ $\quad c^{(i)} = 5$ $\quad \mu_{c^{(i)}} = \mu_5$

Optimization objective:

→ $$J(c^{(1)},\ldots,c^{(m)},\mu_1,\ldots,\mu_K) = \frac{1}{m}\sum_{i=1}^{m} ||x^{(i)} - \mu_{c^{(i)}}||^2$$

→ $$\min_{\substack{c^{(1)},\ldots,c^{(m)},\\ \mu_1,\ldots,\mu_K}} J(c^{(1)},\ldots,c^{(m)},\mu_1,\ldots,\mu_K)$$

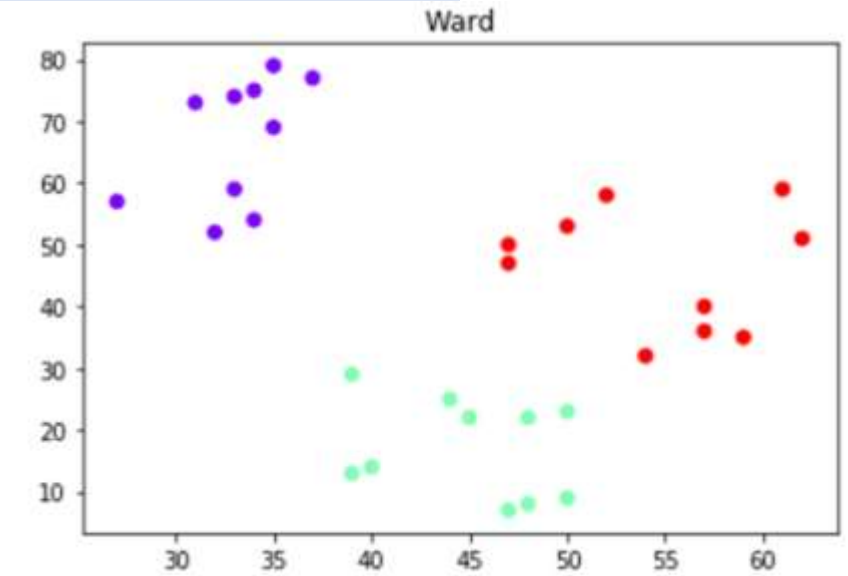
Distortion



```
from sklearn.cluster import KMeans
from sklearn import metrics
from scipy.spatial.distance import cdist
import numpy as np
import matplotlib.pyplot as plt

x1 = np.array([3, 1, 1, 2, 1, 6, 6, 6, 5, 6, 7, 8, 9, 8, 9, 9, 8])
x2 = np.array([5, 4, 5, 6, 5, 8, 6, 7, 6, 7, 1, 2, 1, 2, 3, 2, 3])

plt.plot()
plt.xlim([0, 10])
plt.ylim([0, 10])
plt.title('Dataset')
plt.scatter(x1, x2)
plt.show()

# create new plot and data
plt.plot()
X = np.array(list(zip(x1, x2))).reshape(len(x1), 2)
colors = ['b', 'g', 'r']
markers = ['o', 'v', 's']
```

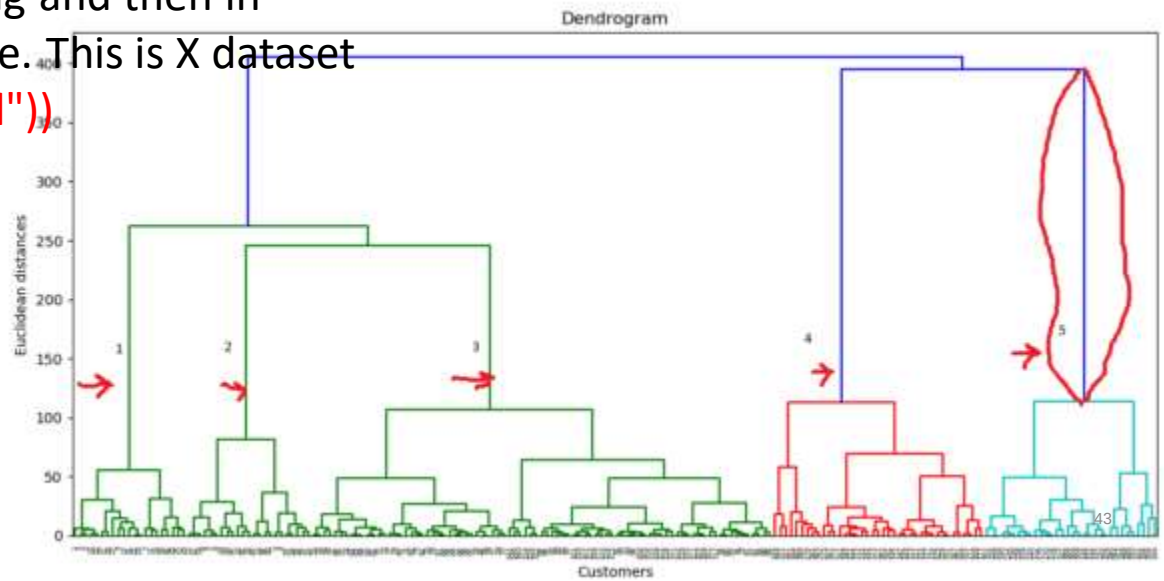
```
# k means determine k
distortions = []
K = range(1,10)
for k in K:
kmeanModel = KMeans(n_clusters=k).fit(X)
kmeanModel.fit(X)
distortions.append(np.sum(np.min(cdist(X, kmeanModel.cluster_centers_, 'euclidean'),
axis=1)) / X.shape[0])

# Plot the elbow
plt.plot(K, distortions, 'bx-')
plt.xlabel('k')
plt.ylabel('Distortion')
plt.title('The Elbow Method showing the optimal k')
plt.show()
```

```
cluster = AgglomerativeClustering(n_clusters=3, affinity='euclidean', linkage='ward
```



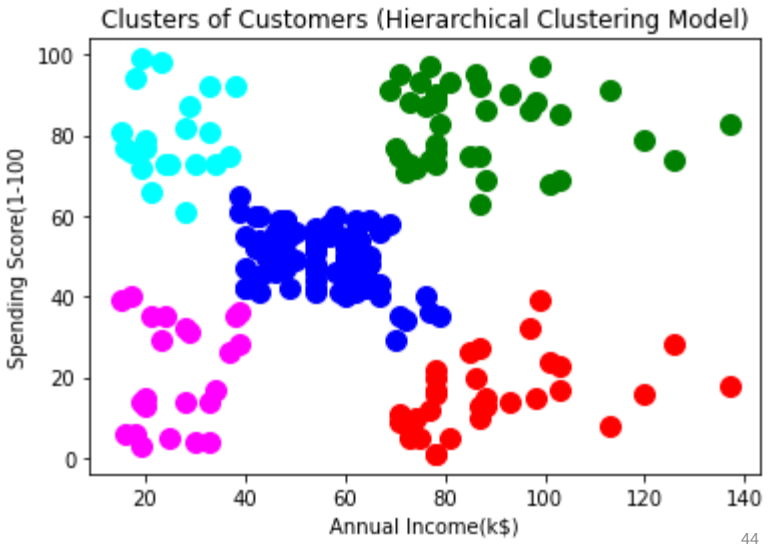
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

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

  #3 Using the dendrogram to find the optimal numbers of clusters.
  # First thing we're going to do is to import scipy library. scipy is an open source
  # Python library that contains tools to do hierarchical clustering and building dendrograms.
  # Only import the needed tool.
  import scipy.cluster.hierarchy as sch
  #create a dendrogram variable
  # linkage is actually the algorithm itself of hierarchical clustering and then in
  #linkage we have to specify on which data we apply and engage. This is X dataset
  dendrogram = sch.dendrogram(sch.linkage(X, method  = "ward"))
  plt.title('Dendrogram')
  plt.xlabel('Customers')
  plt.ylabel('Euclidean distances')
  plt.show()
```

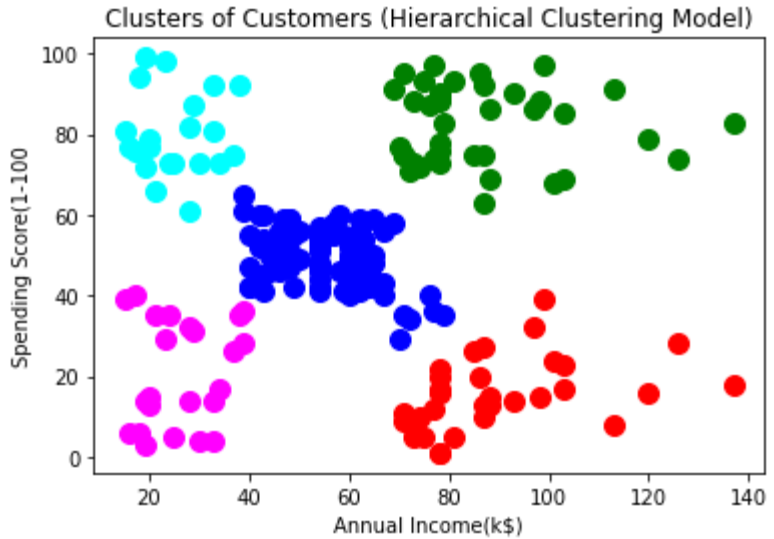


```
#4 Fitting hierarchical clustering to the Mall_Customes dataset
# There are two algorithms for hierarchical clustering: Agglomerative Hierarchical Clustering and
# Divisive Hierarchical Clustering. We choose Euclidean distance and ward method for our
# algorithm class
from sklearn.cluster import AgglomerativeClustering
hc = AgglomerativeClustering(n_clusters = 5, affinity = 'euclidean', linkage ='ward')
# Lets try to fit the hierarchical clustering algorithm  to dataset X while creating the
# clusters vector that tells for each customer which cluster the customer belongs to.
y_hc=hc.fit_predict(X)
```

```
#5 Visualizing the clusters. This code is similar to k-means visualization code.
#We only replace the y_kmeans vector name to y_hc for the hierarchical clustering
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 (Hierarchical Clustering Model)')
plt.xlabel('Annual Income(k$)')
plt.ylabel('Spending Score(1-100')
plt.show()
```

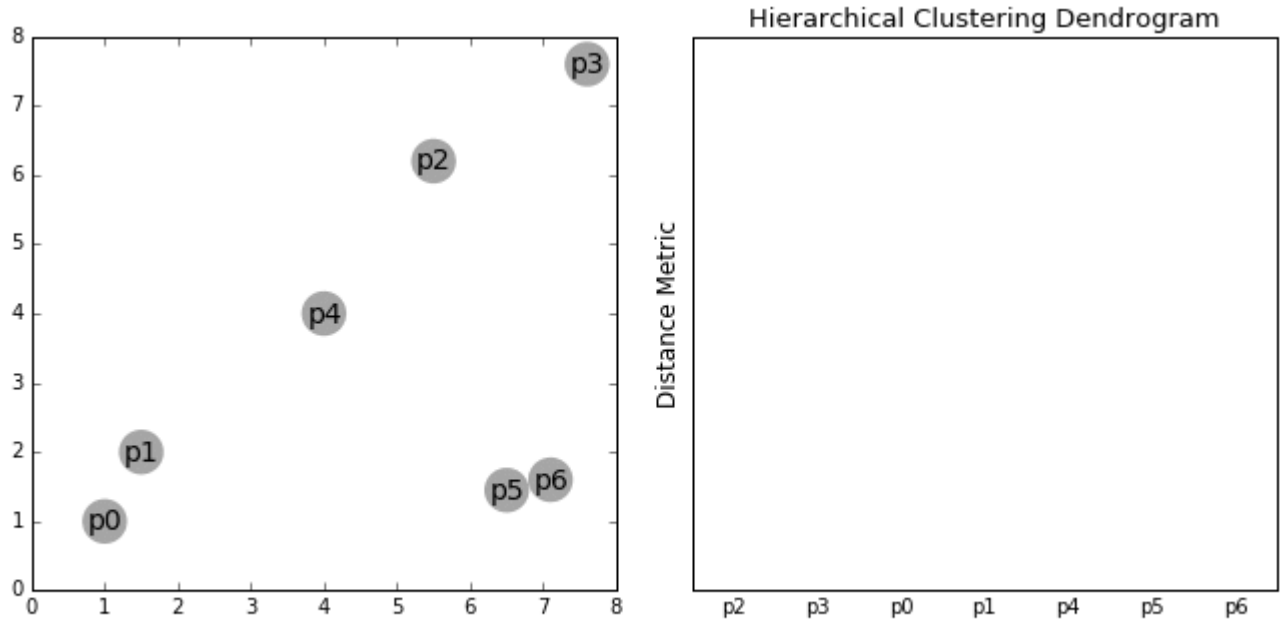


Cluster #	Annual Salary vs Spending Score	Customer Segment
Cluster1(Red)	High Income vs Low Spending Score	Carefull
Cluster2 (Blue)	Normal Income vs Normal Spending Score	Standard
Cluster3(Green)	High Income vs High Spending Score	Target
Cluster4(Cyan)	Low Income vs High Spending Score	Careless
Cluster5 (Magenta)	Low Income vs Low Spending Score	Sensible



Kmeans

Impact of different distance metrics on hierarchical clustering,



k-means Clustering

k-means, using a pre-specified number of clusters, the method assigns records to each cluster to find the mutually exclusive cluster of spherical shape based on distance.

K Means clustering needed advance knowledge of K i.e. no. of clusters one want to divide your data.

One can use median or mean as a cluster centre to represent each cluster.

Methods used are normally less computationally intensive and are suited with very large datasets.

In K Means clustering, since one start with random choice of clusters, the results produced by running the algorithm many times may differ.

K- means clustering a simply a division of the set of data objects into non-overlapping subsets (clusters) such that each data object is in exactly one subset).

K Means clustering is found to work well when the structure of the clusters is hyper spherical (like circle in 2D, sphere in 3D).

Advantages: 1. Convergence is guaranteed. 2. Specialized to clusters of different sizes and shapes.

Disadvantages: 1. K-Value is difficult to predict 2. Didn't work well with global cluster.

Hierarchical Clustering

Hierarchical methods can be either divisive or agglomerative.

In hierarchical clustering one can stop at any number of clusters, one find appropriate by interpreting the dendrogram.

Agglomerative methods begin with 'n' clusters and sequentially combine similar clusters until only one cluster is obtained.

Divisive methods work in the opposite direction, beginning with one cluster that includes all the records and Hierarchical methods are especially useful when the target is to arrange the clusters into a natural hierarchy.

In Hierarchical Clustering, results are reproducible in Hierarchical clustering

A hierarchical clustering is a set of nested clusters that are arranged as a tree.

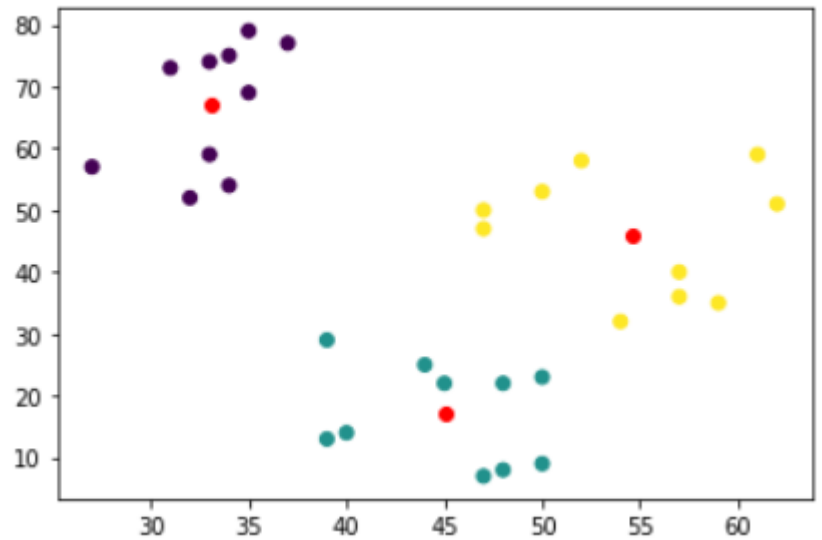
Hierarchical clustering don't work as well as, k means when the shape of the clusters is hyper spherical.

Advantages: 1 .Ease of handling of any forms of similarity or distance. 2. Consequently, applicability to any attributes types.

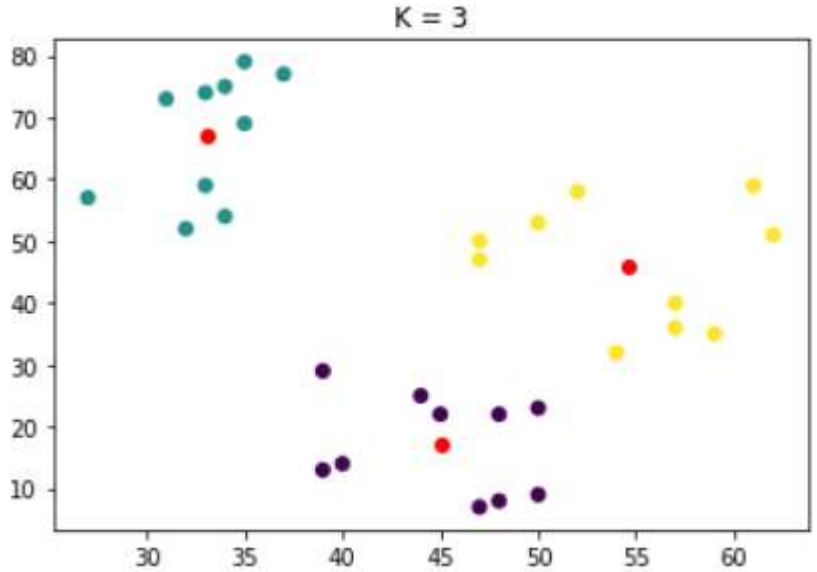
Disadvantage: 1. Hierarchical clustering requires the computation and storage of an n×n distance matrix. For very large datasets, this can be

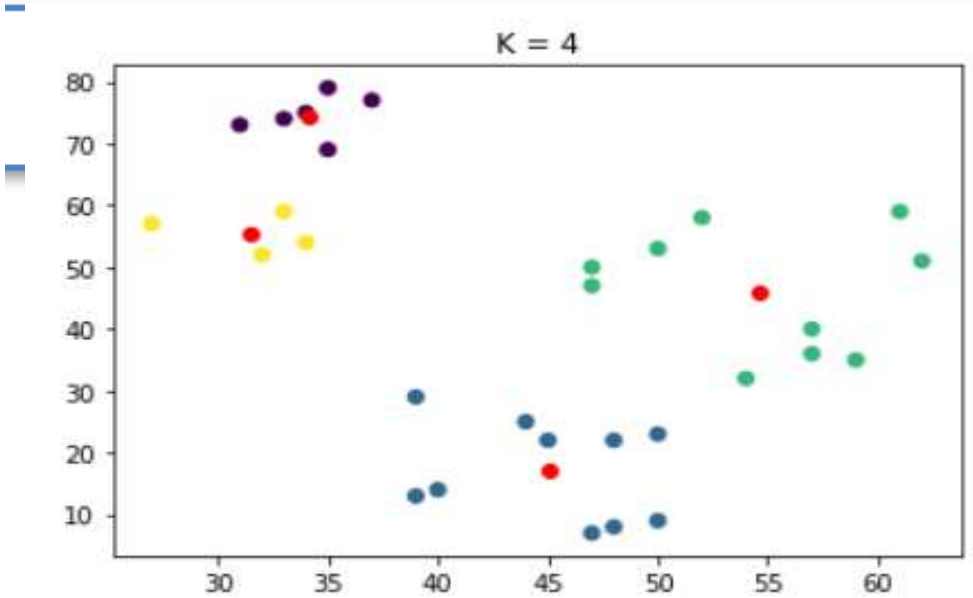
In [1]:

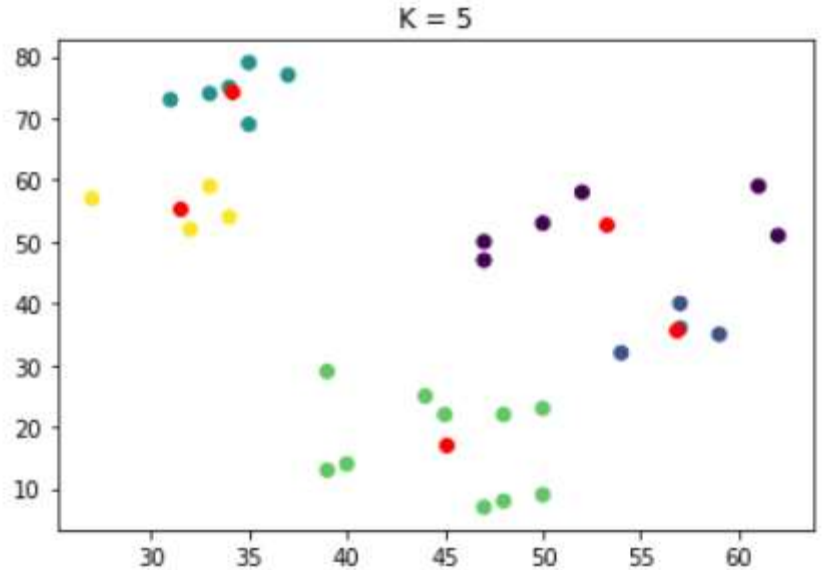
```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from pandas import DataFrame

Data = {
    'x': [35,34,32,37,33,33,31,27,35,34,62,54,57,47,50,57,59,52,61,47,50,48,39,40,45,47,39,44,50,48],
     'y': [79,54,52,77,59,74,73,57,69,75,51,32,40,47,53,36,35,58,59,50,23,22,13,14,22,7,29,25,9,8]
    }

df = DataFrame(Data,columns=['x','y'])

kmeans = KMeans(n_clusters=3).fit(df)
centroids = kmeans.cluster_centers_

plt.scatter(df['x'], df['y'], c=kmeans.labels_.astype(float))
plt.scatter(centroids[:, 0], centroids[:, 1], c='red')
plt.show()
```



```python
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from pandas import DataFrame

Data = {
        'x': [35,34,32,37,33,33,31,27,35,34,62,54,57,47,50,57,59,52,61,47,50,48,39,40,45,47,39,44,50,48],
        'y': [79,54,52,77,59,74,73,57,69,75,51,32,40,47,53,36,35,58,59,50,23,22,13,14,22,7,29,25,9,8]
       }

df = DataFrame(Data,columns=['x','y'])

kmeans = KMeans(n_clusters=3).fit(df)
centroids = kmeans.cluster_centers_

plt.scatter(df['x'], df['y'], c=kmeans.labels_.astype(float))
plt.scatter(centroids[:, 0], centroids[:, 1], c='red')
plt.title('K = 3')
plt.show()

kmeans = KMeans(n_clusters=4).fit(df)
centroids = kmeans.cluster_centers_

plt.scatter(df['x'], df['y'], c=kmeans.labels_.astype(float))
plt.scatter(centroids[:, 0], centroids[:, 1], c='red')
plt.title('K = 4')
plt.show()

kmeans = KMeans(n_clusters=5).fit(df)
centroids = kmeans.cluster_centers_

plt.scatter(df['x'], df['y'], c=kmeans.labels_.astype(float))
plt.scatter(centroids[:, 0], centroids[:, 1], c='red')
plt.title('K = 5')
plt.show()
```


K = 3
80
70
60
50
40
30
20
10
30
35
40
45
50
55
60


K = 4
80
70
60
50
40
30
20
10
30
35
40
45
50
55
60


K = 5
80
70
60
50
40
30
20
10
30
35
40
45
50
55
60

```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from pandas import DataFrame
Data = {
        'x': [35,34,32,37,33,33,31,27,35,34,62,54,57,47,50,57,59,52,61,47,50,48,39,40,45,47,39,44,50,48],
        'y': [79,54,52,77,59,74,73,57,69,75,51,32,40,47,53,36,35,58,59,50,23,22,13,14,22,7,29,25,9,8]
       }

df = DataFrame(Data,columns=['x','y'])
distances = []
K = range(1,10)
for k in K:
    ClusterInfo = kmeanModel = KMeans(n_clusters=k).fit(df)
    distances.append(ClusterInfo.inertia_)

plt.plot(K, distances, 'bo-')
plt.xlabel('K-Clusters')
plt.ylabel('Distance')
plt.title('Cluster Values and Distances')
plt.show()
```