K-Means Clustering

Presented by ~

Abhisek Sarkar

(as20ms091@iiserkol.ac.in)

What is Clustering?

Clustering is an unsupervised learning technique that groups data points based on their inherient similarities, aiming to discover underlying patterns and structures within the data.

But What is Unsupervised learning?

Types of Machine Learning :- 3 main types :-

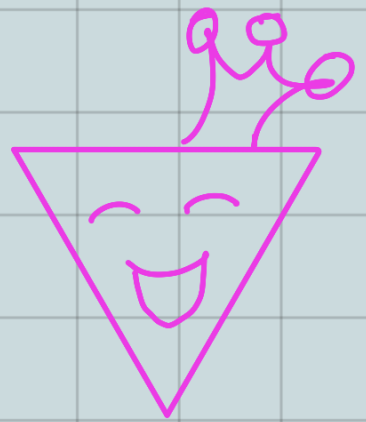
i) Supervised Learning :- model is trained on labeled data, meaning each training example is paired with an output label. The model learns to map input to the correct outputs. Ex :- Classification

ii) Unsupervised Learning :- model deals with unlabeled data. Model tries to identify patterns or groupings within the data without any explicit labels. Ex :- Clustering

iii) Reinforcement Learning :- Agent learns to make decisions by taking actions in an environment to maximize a cumulative reward. The agent receives feedback in the form of rewards or penalties based on its actions. Ex :- Navigate a maze

What is natural grouping among them?

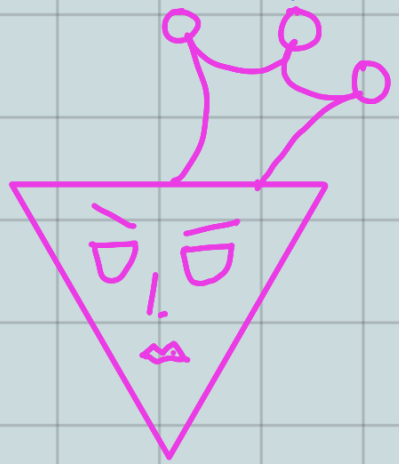
King



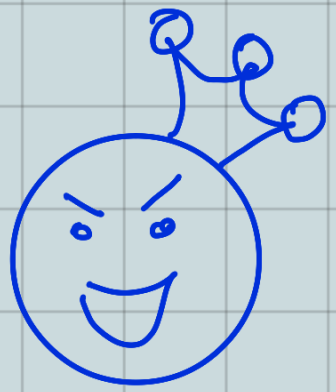
Princen

Man

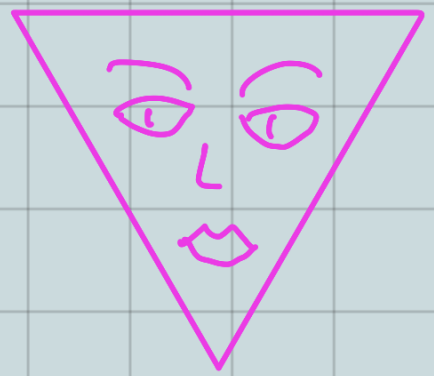
Girl



Queen



Prince

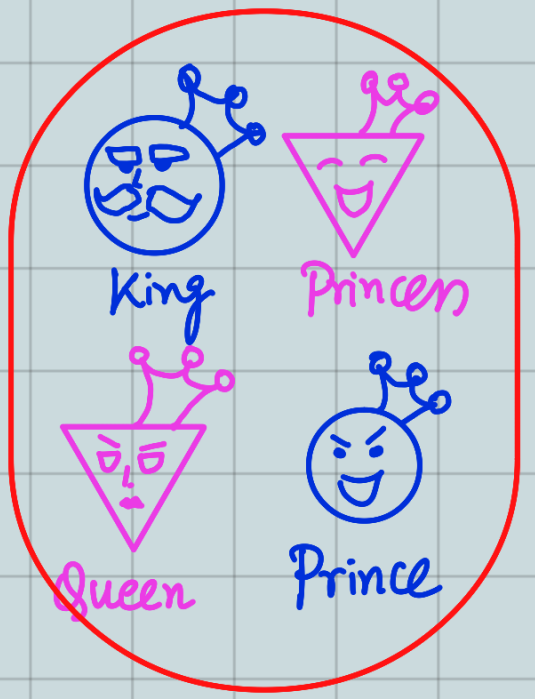


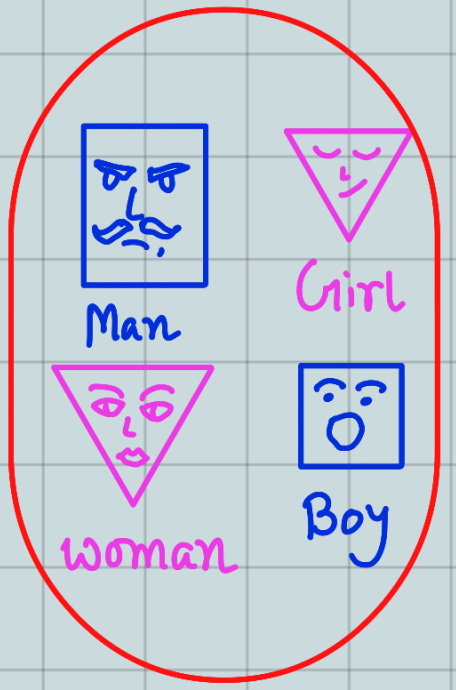
woman

Boy

Clustering is subjective!

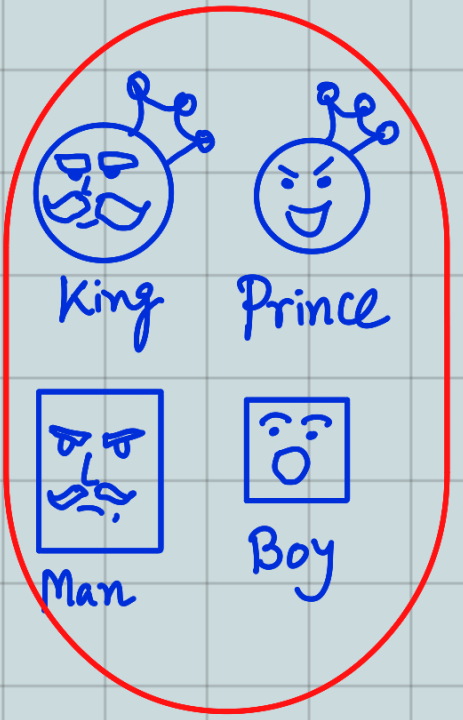
Royality

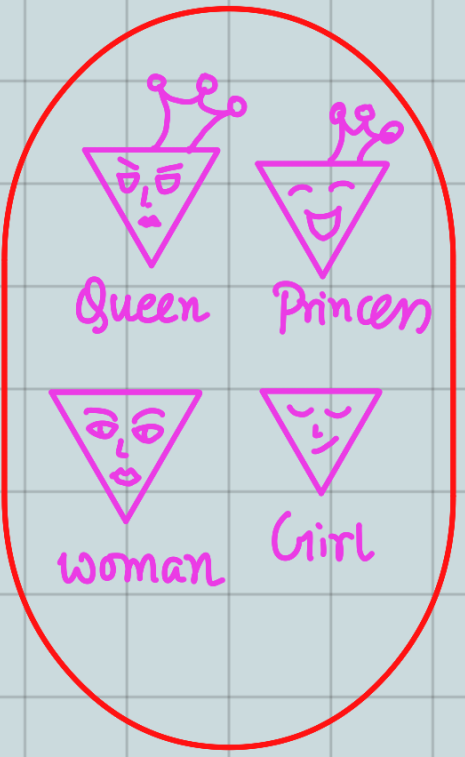




Based on Royality

Grender





Based on Grender

Aim of Clustering!

- high intra-class similarity
- low inter-class similarity

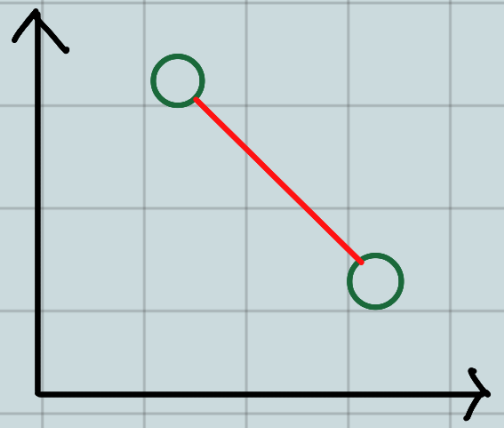
But what does determine the similarity?

The answer is Distance Measure.

Distance measure will determine how the similarity of two elements is calculated and it will influence the shape of the clusters.

Now we will discuss some distance measure.

1. The Euclidian distance :- (also called 2-norm distance)



measures the shortest distance between two real valued vectors.

$$d = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

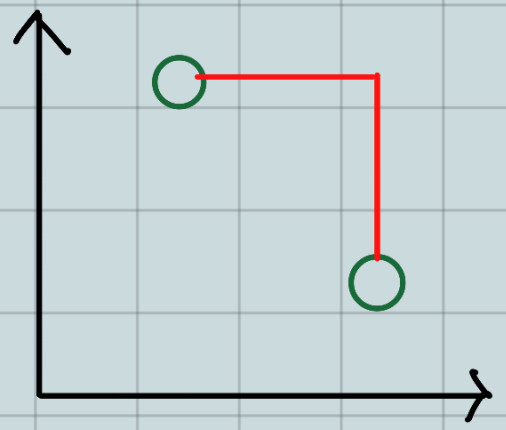
Code :-

```
from Scipy.Spatial import distance
distance.euclidean (vector_1, vector_2)
```

- Does not work well on higher dimensional data than 2D & 3D

- If we don't normalize the features, the distance might be skewed due to different units.

2. Manhattan distance :- (also called taxicab norm or 1-norm)



The distance betⁿ two real valued vectors is calculated as if one could only move at right angles.

$$d = \sum_{i=1}^{n} |x_i - y_i|$$

Code :- from scipy.spatial import distance
distance.cityblock (vector_1, vector_2)

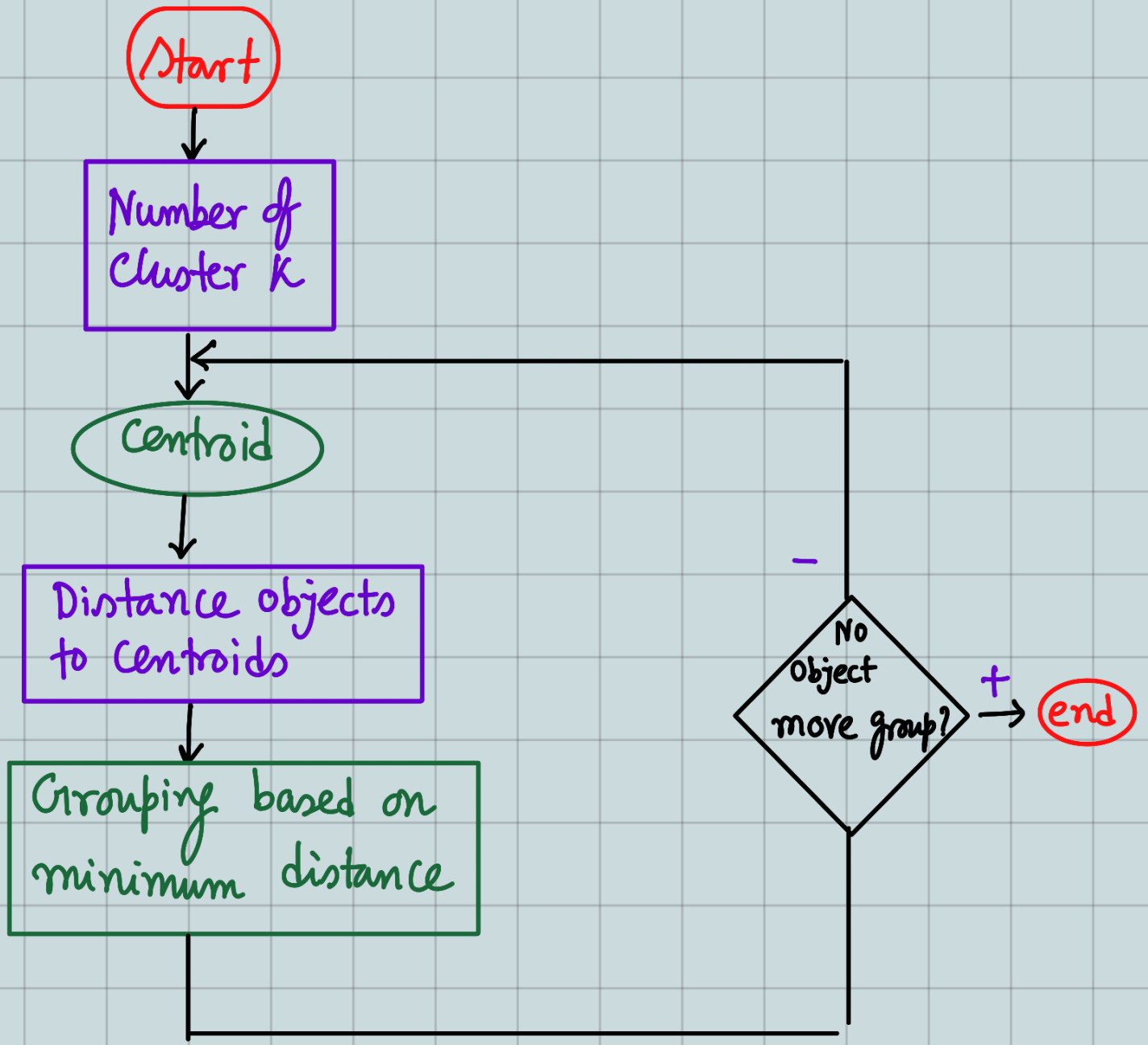
- It does not show the shortest path possible.
- Less intuitive than Eucledian distance in high dimensional space.

K-means Clustering:- The K-means clustering is an algorithm to cluster n objects based on attributes into k partitions where $k < n$

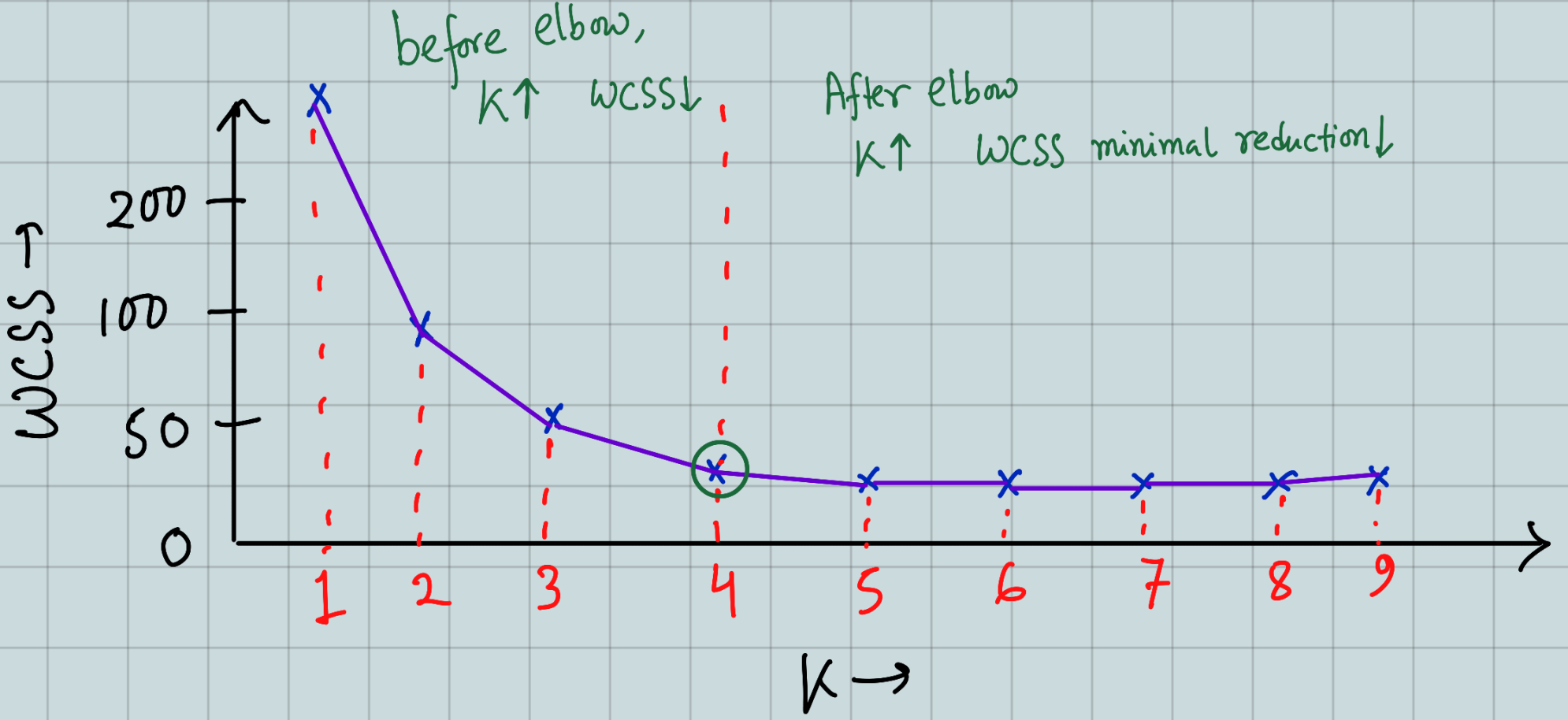
- k is a positive integer number

- The grouping is done by minimizing the sum of squares of distances betⁿ data and the corresponding Cluster Centroid.

But how does it work?

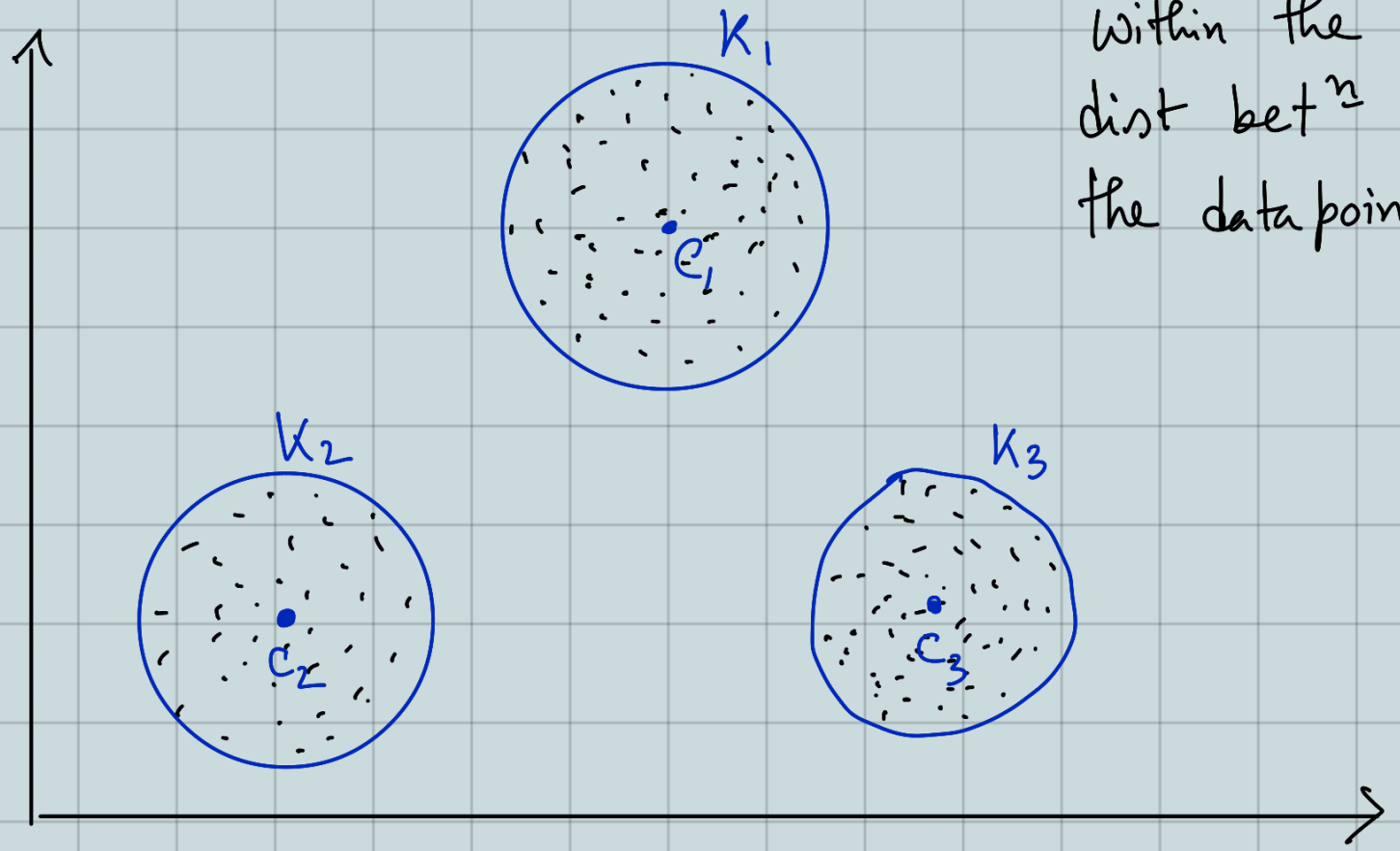

Start
Number of Cluster K
Centroid
Distance objects to centroids
Grouping based on minimum distance
No Object move group?
-
+
end

Elbow Method:- Choosing optimal number of clusters is a crucial step. Since we don't have predefined cluster counts in unsupervised learning, we need a symantic approach to determine the K- value.

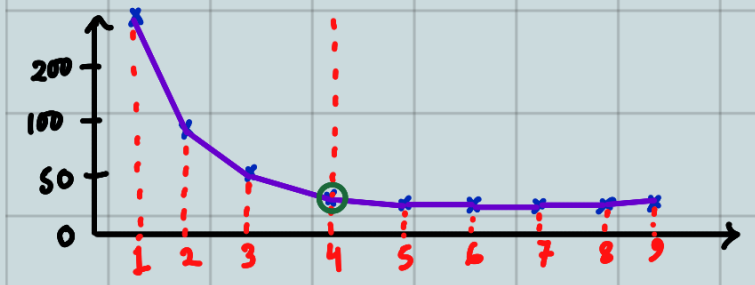


WCSS (Within Cluster Sum of Squares)

$$WCSS = \sum_{C_k}^{C_n} \left(\sum_{d_i \text{ in } C_i}^{d_m} \text{distance}(d_i, C_k)^2 \right)$$



within the cluster the dist betⁿ centeroids and the datapoints.



- We Calculate a distance measure called WCSS (within-Cluster-Sum of Squares). This tells us how spread out the data points are.
- We try different K-values and run KMeans and Calculate the WCSS.
- We plot a graph with K- on x axis and WCSS on y axis.
- We increase K, the WCSS typically decreases because we are creating more clusters, which tend to capture more data variations. However, there comes a point where adding more clusters results in only a marginal decrease in WCSS. This is where we observe "Elbow" shape in the graph.

kmeans

April 1, 2025

1 K- means clustering Code

1.1 CS2201 course

1.1.1 1. Importing Necessary Libraries

[1]:
```
import pandas as pd # Used for handling tabular data
import numpy as np # Provides support for numerical operations and arrays
import matplotlib.pyplot as plt # Used for data visualization
from sklearn.cluster import KMeans #  Implements the K-Means Clustering␣
 ↪Algorithm
```

1.1.2 2. Loading Data from a CSV File

[2]:
```
# Data loading from a file

file_path = '/home/abhisek/CS2201_Spring_2025/eighth_week/data3.csv'  # stores␣
 ↪the filename data3.csv, which contains the dataset
data = pd.read_csv(file_path) # reads the CSV file and stores it in a DataFrame␣
 ↪(data).

print(data) # displays the dataset
```

```
   Avg   SR
0   44   87
1   40   70
2   41   79
3   45  109
4   41   75
5   60  102
6   55  110
7   30  104
8   25  109
9   50  100
```

3. Feature Selection

[3]:
```
feature1 = 'Avg'  # Change to your first feature column name
feature2 = 'SR'  # Change to your second feature column name
```

feature1 and feature2 are set to column names 'Avg' and 'SR'.

These columns are selected as features for clustering

[4]:
```
X = data[[feature1, feature2]].values # stores the selected features as a NumPy␣
 ↪array (.values extracts values)
print(X) # isplays the extracted feature values
```

```
[[ 44  87]
 [ 40  70]
 [ 41  79]
 [ 45 109]
 [ 41  75]
 [ 60 102]
 [ 55 110]
 [ 30 104]
 [ 25 109]
 [ 50 100]]
```

4. Finding the Optimal Number of Clusters using the Elbow Method

a. Setting Up Parameters

[5]:
```
# Setting Up Parameters
cluster_range = range(1, 11) # # Trying clusters from 1 to 10
wcss = []  # List to store Within-Cluster Sum of Squares (WCSS)
```

The range of possible number of clusters (k) is set from 1 to 10.

wcss (Within-Cluster Sum of Squares) is an empty list to store WCSS values.

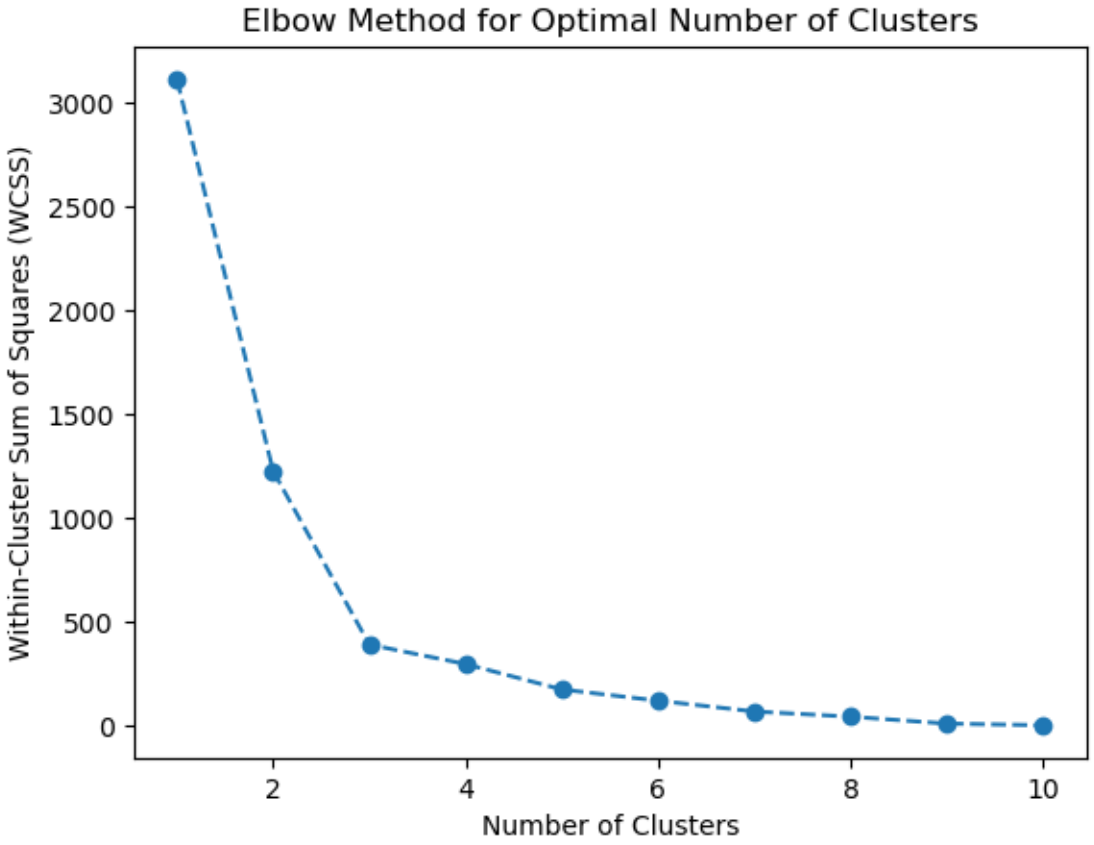
b. Loop to Compute WCSS for Different Values of k

[6]:
```
for k in cluster_range:
    kmeans = KMeans(n_clusters=k, init='k-means++', random_state=42)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)  # Inertia (WCSS) is calculated and stored
```

- For each **k** (number of clusters from 1 to 10):
  - **KMeans(n_clusters=k, init='k-means++', random_state=42)** initializes the K-Means model:
    - n_clusters=k: Number of clusters.
    - init='k-means++': Smart centroid initialization to avoid random initialization trap.
    - random_state=42: Ensures reproducibility.
  - **kmeans.fit(X)** applies clustering to the feature data.
  - **kmeans.inertia_** stores the sum of squared distances between points and their assigned cluster center (WCSS).
  - **wcss.append(kmeans.inertia_)** adds this value to the wcss list.

1.1.3 5. Visualizing the Elbow Method Plot

[7]:
```
plt.plot(cluster_range, wcss, marker='o', linestyle='--')
plt.title('Elbow Method for Optimal Number of Clusters')
plt.xlabel('Number of Clusters')
plt.ylabel('Within-Cluster Sum of Squares (WCSS)')
plt.show()
```



- Plots **number of clusters (x-axis) vs. WCSS (y-axis)**.
- **Elbow Point**: The point where WCSS decreases at a slower rate, indicating the best number of clusters.

1.1.4 6. User Input for Optimal k

[8]:
```
optimal_k = int(input("Based on the elbow plot, enter the optimal number of␣
↪clusters: "))
```

- The user manually enters the **best number of clusters** based on the elbow plot.

1.1.5 7. Applying K-Means Clustering

[9]:
```
kmeans = KMeans(n_clusters=optimal_k, init='k-means++', random_state=42)
clusters = kmeans.fit_predict(X)
```

- Initializes **K-Means** with the selected `optimal_k`.
- **`kmeans.fit_predict(X)`** assigns each data point to a cluster.
- **`clusters`** stores cluster labels (0, 1, 2, …).

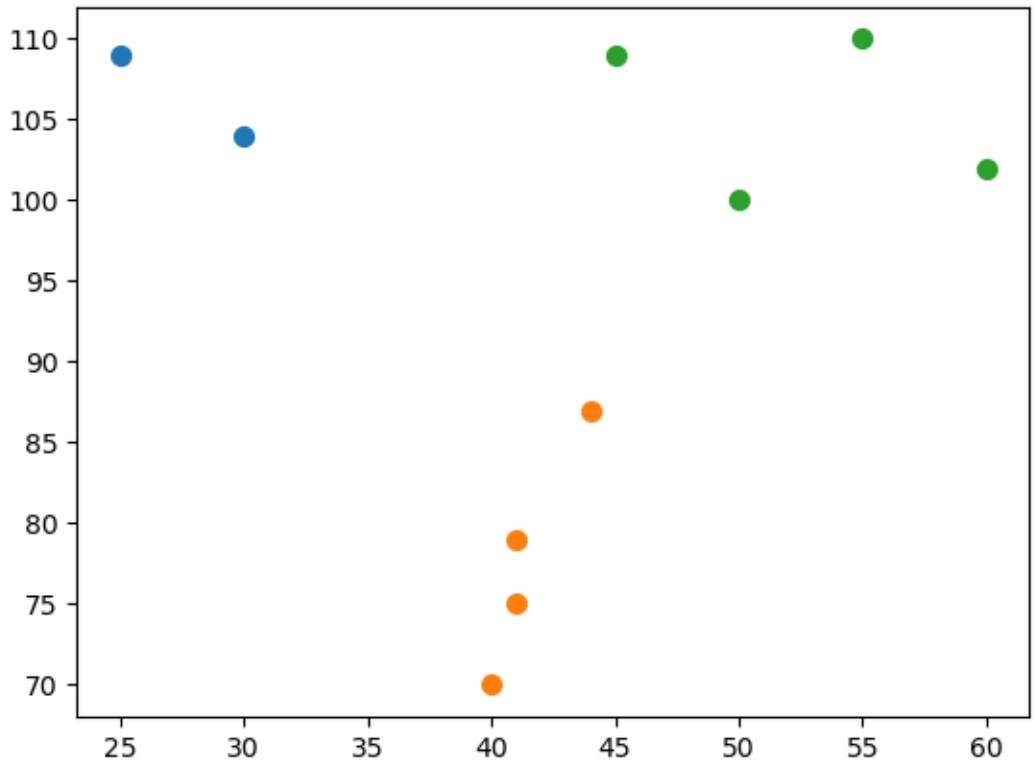
1.1.6 8. Adding Cluster Labels to Data

[10]:
```
data['Cluster'] = clusters
```

- A new column **"Cluster"** is added to the dataset to store each point's assigned cluster.

1.1.7 9. Plotting the Clusters

[11]:
```
for cluster in range(optimal_k):
    plt.scatter(X[clusters == cluster, 0], X[clusters == cluster, 1], s=50,␣
 ↪label=f'Cluster {cluster}')
```



- Loops through each **cluster (0 to optimal_k - 1)**:
  - Uses `plt.scatter()` to plot all data points in that cluster.
  - Uses different colors for different clusters.

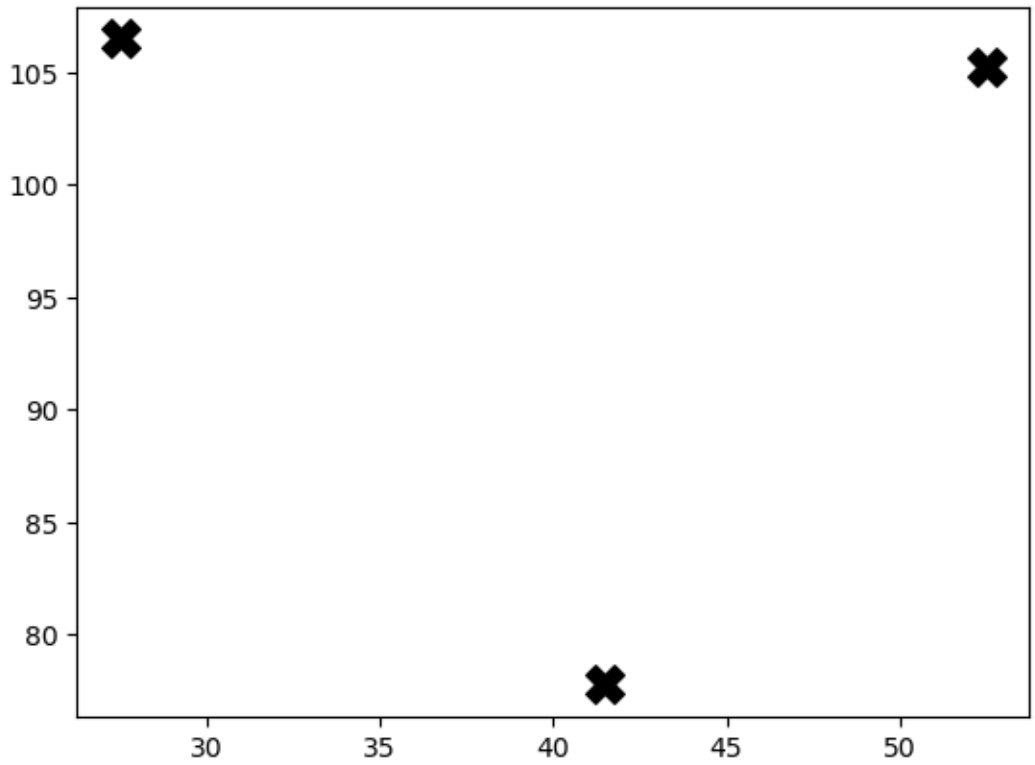
1.1.8 10. Plotting the Cluster Centroids

[16]:
```
print(kmeans.cluster_centers_)
```

```
[[ 27.5  106.5 ]
 [ 41.5   77.75]
 [ 52.5  105.25]]
```

[17]:
```
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],␣
↪s=200, c='black', marker='X', label='Centroids')
```

[17]: `<matplotlib.collections.PathCollection at 0x7bab3c108bf0>`

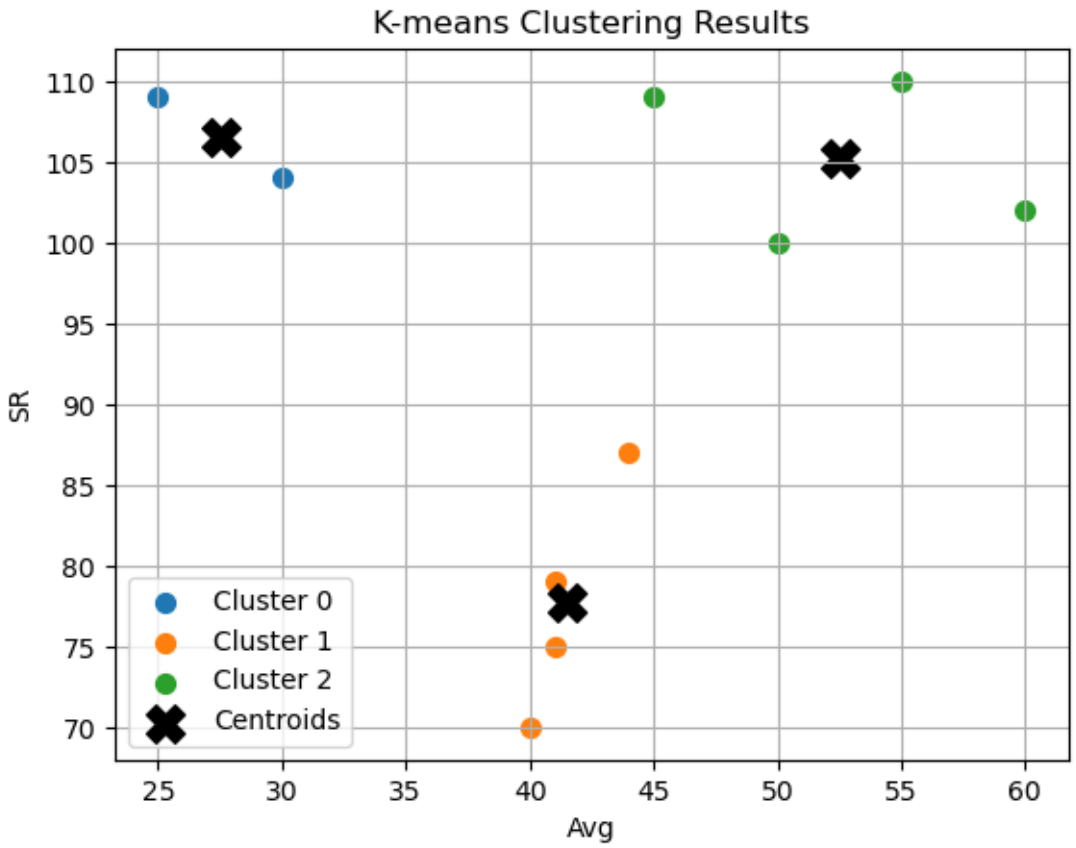


- **`kmeans.cluster_centers_`** contains the coordinates of the cluster centers.
- Plots the centroids as **black 'X' markers** with larger size (`s=200`).

1.1.9 11. Finalizing and Displaying the Cluster Plot

[22]:
```
# K-means

kmeans = KMeans(n_clusters=optimal_k, init='k-means++', random_state=42)
clusters = kmeans.fit_predict(X)

# Add cluster labels to original data
data['Cluster'] = clusters

# Plotting the clusters

print(X[clusters == 0])

#plt.figure(figsize=(10, 6))
for cluster in range(optimal_k):
    plt.scatter(X[clusters == cluster, 0], X[clusters == cluster, 1], s=50,␣
 ↪label=f'Cluster {cluster}')

# Plot the centroids

print(kmeans.cluster_centers_)

plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],␣
 ↪s=200, c='black', marker='X', label='Centroids')

plt.title('K-means Clustering Results')
plt.xlabel(feature1)
plt.ylabel(feature2)
plt.legend()
plt.grid()
plt.show()
```

```
[[ 30 104]
 [ 25 109]]
[[ 27.5  106.5 ]
 [ 41.5   77.75]
 [ 52.5  105.25]]
```



- Adds title, axis labels, legend, and grid.
- `plt.show()` displays the final **K-means Clustering Plot**.

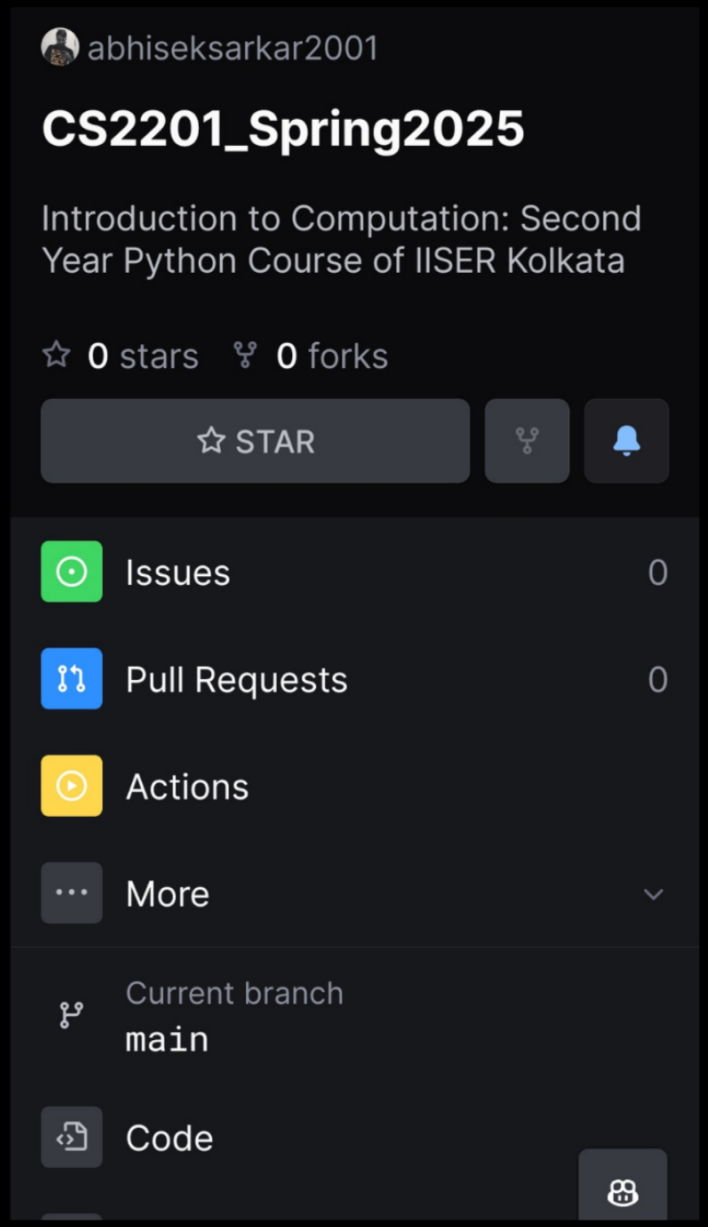
1.1.10 12. Displaying the Clustered Data

[14]:
```
print("Data with cluster assignments:")
print(data)
```

```
Data with cluster assignments:
   Avg   SR  Cluster
0   44   87        1
1   40   70        1
2   41   79        1
3   45  109        2
4   41   75        1
5   60  102        2
6   55  110        2
7   30  104        0
8   25  109        0
```

```
9    50   100          2
```

- Prints the updated dataset, now with **cluster assignments**.



Visit The GitHub Repo for all the Slides, Notes and Codes for this Course

https://github.com/abhiseksarkar2001/CS2201_Spring2025