Clustering → Grouping similar items together.

Why clustering → To group similar items together & take actions as a group.

ex →

	Industry	Action
1)	Retail	- Group similar customers. - Initiate different campaigns for different groups of customers.
2)	Banking	- Group similar transactions together. - Easier to identify fraudulent transactions.
3)	Healthcare	- Group similar patients together. - Prescribe medicine based on the group a patient is part of.

K-means → From K groups, to allocate each data point to one of the K groups.

Algorithm →

1) Choose a K value.
2) Assign K points randomly as cluster centers.
3) Assign each data point to one of the groups.
4) Update the centroid for groups by calculating the average point of all the points in the group.
5) Repeat step (3-4) until clusters stop changing.

Similarity → Euclidian distance b/w two points.

$$\text{Distance} \propto \frac{1}{\text{Similarity}}$$

Centroid → Center of a cluster. (similar to center of gravity).
- May or may not be an actual data point.
- Representative of a group.

Goal of K means → 1) Minimize intra cluster distances →
Data points inside a cluster are as close to the centroid

as possible.

2) Maximize inter-cluster distance → Make clusters as far apart from eachother as possible.

→ Metrics in K means →

1) WCSS → within cluster sum of squares.

- measure how close the points are to the centroid.
- not a good metric → A large cluster can have large wcss value even if it is tightly packed.

$$wcss \propto \frac{1}{compactness\ (closeness\ of\ points)}$$

2) BCSS → Between cluster sum of square.

- how far apart the centers of each cluster are from a true center point.

3) TSS → Total sum of squares.

- Measures the total variability in the dataset.

$$TSS = wcss + bcss.$$

→ K-means works both on small & large datasets.

1) Calculate distance b/w points & centroids.
& Assign each point to the nearest centroid.

2) Recompute the centroid for each group of points.

def k-mean(points, k, initialCentroids, iterations)

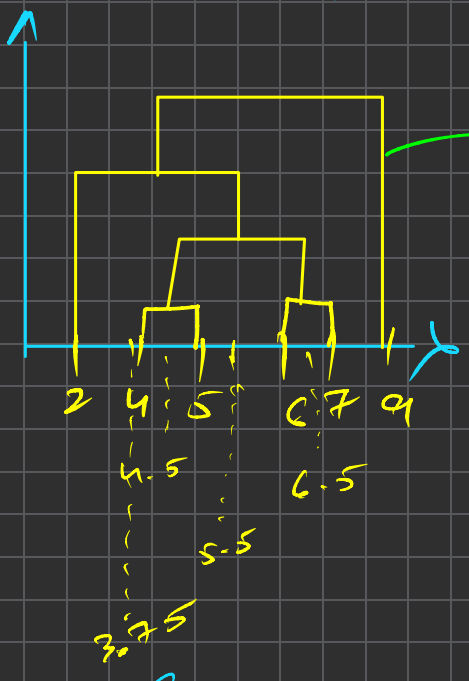
(list of points ↓ list of points of size k

tuple t(x, y)

Hierarchical Clustering

- works poorly on larger datasets.
- K-means works both on small & large datasets.

Dendrogram → The graph created by hierarchical clustering.



Hierarchy → Form a tree like structure.

a) Agglomerative → (Bottom up) → Merge smaller clusters into bigger ones.

- Common

1) Each data point starts as a single cluster.
2) At each step two most similar clusters are combined into a single cluster.
3) Repeat until there's only a single cluster remaining.

b) Divisive → (Top Down) → Split a single cluster into two & repeat until no more splits are possible.

When to use hierarchical clustering →

a) When the dataset is small.
b) When detailed cluster hierarchy is required.
   - To visualize & make decisions.

K-Mean Clustering using sklearn library -

```
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_score

x = df[[ feature x, feature y ]]
scaler = StandardScaler( )
xs = scaler.fit_transform(x)

wcss = [ ]
silhouettes = [ ]
krange = range(2, 11)

for k in krange:
    kmean = KMeans( n = k, random_state = 42, n_init = 10 )
    kmean.fit( xs )
    wcss.append( kmean.inertia_ )
    silhouettes.append( silhouette_score( xs, kmeans.labels_ ))

bestk = krange[ silhouettes.index( max ( silhouettes )) ]
```

-> This was to find the best k.

-> Below code is for using the k mean model.

- Finding clusters.

kmean.cluster_centers_ -> List of Position of every cluster center.

scaler.inverse_transform(kmean.cluster_centers_)
- Position of cluster centers scaled to the dataset.

```
clusters = kmeans.fit_predict( xs )
```

-> create k means with the best cluster.

```
kmeans = KMeans( n = bestk, seed = 42, n_init = 10)
```

Clustering on text data

```
from vaderSentiment import SentimentIntensityAnalyzer
from sklearn.feature_extraction.text import CountVectorizer

tweets = df['tweets'].tolist()
analyser = SentimentIntensityAnalyzer()
for tweet in tweets:
    ps = analyser.polarity_scores(tweet)
    tsentiments.append({'tweet': tweet, 'compound': ps['compound']})
tdf = pd.DF(tsentiments)
sentiments = []
for itweet in tdf.compound:
    if itweet > 0:
        sentiment += 'Pos'
    elif itweet < 0:
        sentiment += 'neg'
    else
        sentiment += 'neutral'
```