

What is Unsupervised Learning ?

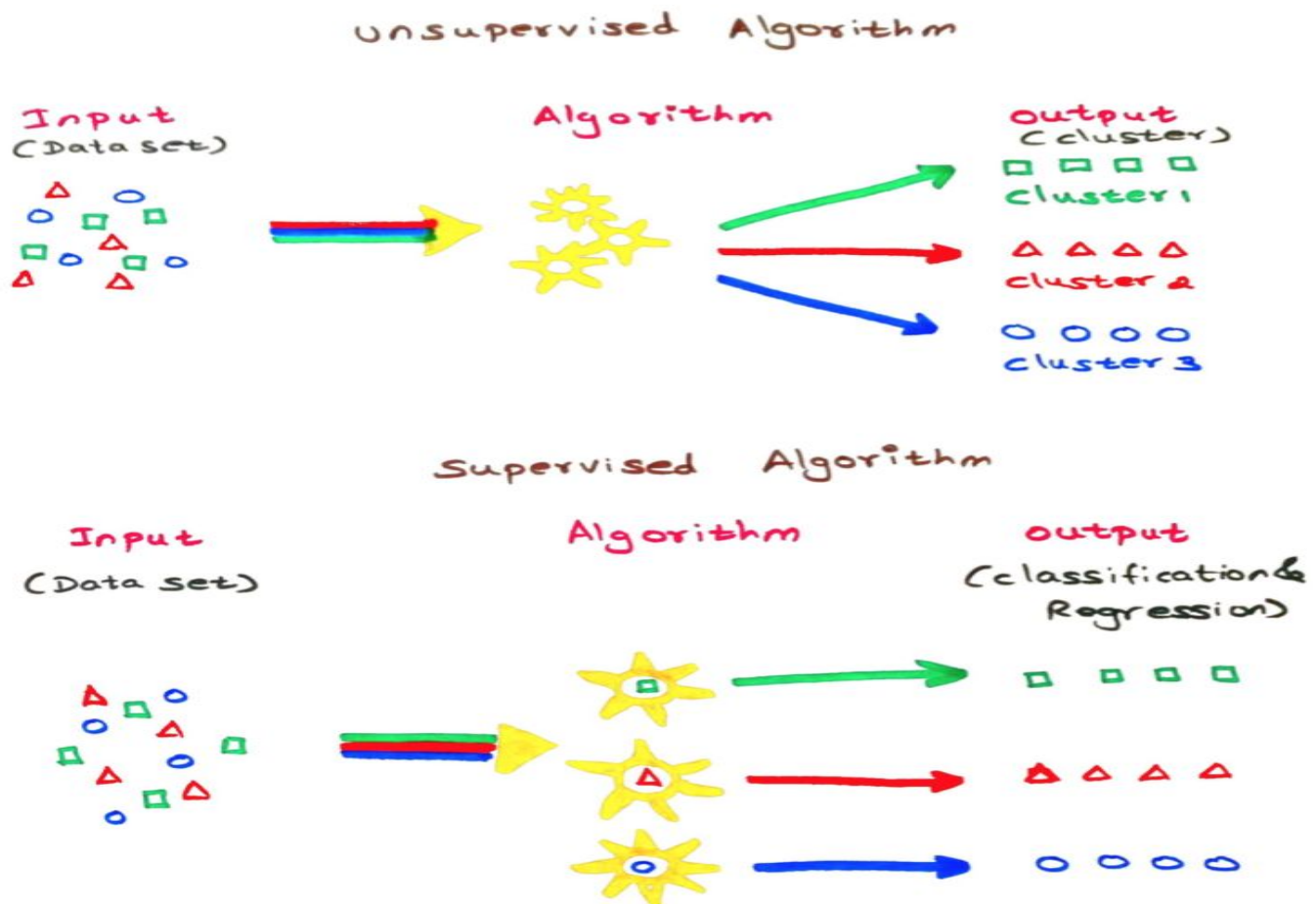
Recap of Supervised learning

Up until this point in your journey every machine learning algorithm that you have come across (except PCA) are variants of Supervised methods. Supervised learning methods intake data points containing feature/features as well as an associated output/response. The goal of supervised learning is to try and predict the output/response from the same features. It is called "supervised" because in the training (or supervision), phase of the learning process the algorithm has access to the "ground truth" as the outputs are known. By making use of them supervised methods adjust their model parameters such that when it is exposed to new features it can attempt to make an estimate of the response.

So, how will one proceed with analysis in case he doesn't have access to the so-called "ground truth" labels? This is where unsupervised methods come into the picture. Lets discuss about them in more details.

What is Unsupervised learning?

Both supervised and unsupervised learning are similar in the context that one has access to the features but in unsupervised setting you don't have responses i.e. there is difference in the kind of training data used. Instead we are interested solely in attributes of the features themselves. This might include whether the features form specific clusters or sub-groups in feature space. It might also include whether we are able to describe very high-dimensional data in a much lower-dimensional setting.



In the image above, for supervised case we already had the background knowledge about observations i.e. they are some geometrical shapes (squares, triangles and circles) with some color. On feeding this data to a supervised algorithm we can predict what a new observation's shape would be.

However, for unsupervised learning tasks, there is no such background knowledge available i.e. no information about shapes and colors. When we feed this kind of data to an unsupervised algorithm, it tries to find patterns within our observations based on certain features and finally leads to groups / clusters within data. Considering your features are good enough and with proper domain knowledge you can arrive at clusters according to geometric shapes as shown in the image. This is what constitutes unsupervised learning.

Motivation for Unsupervised methods

- Supervised methods use labelled data which is often expensive and takes a lot of time to be prepared. Unsupervised methods do not require such labelling making the process cheap and fast.
- Images, videos, natural language documents and scientific research data possess very high dimensionality which require supervised learning techniques with many degrees of freedom, potentially leading to overfitting and thus poor test performance. Unsupervised learning techniques are a partial solution to these problems.

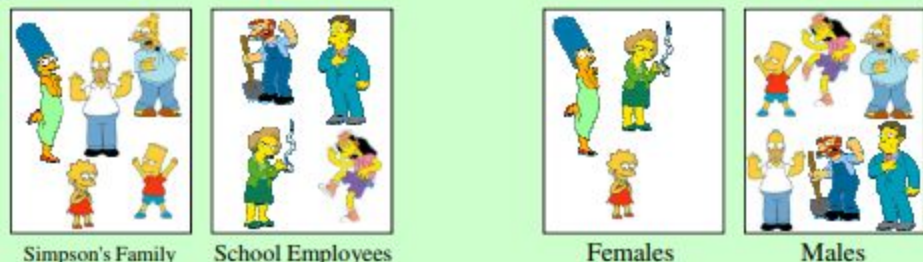
Are there any drawbacks?

- Subjectiveness: The absence of responses pose a big problem while assessing the performance of unsupervised algorithms. This lack of "ground truth" or "supervision" for unsupervised techniques often leads to subjective assessment of their performance. There are no widely agreed approaches for quantifying how well unsupervised algorithms have done. Performance is largely determined on a case-by-case basis using heuristic approaches. The image below represents one such problem if subjectiveness.

What is a natural grouping among these objects?



Clustering is subjective



- Defining similarity: Similarity is the defining criteria for identifying patterns or groups among instances. But how do we measure it and why use that particular method for measuring it? Well it turns out that there is not a best method.

What is Similarity?

The quality or state of being similar; likeness; resemblance; as, a similarity of features.

Webster's Dictionary



Similarity is hard to define, but...
"We know it when we see it"

The real meaning of similarity is a philosophical question. We will take a more pragmatic approach.

Applications of Unsupervised methods

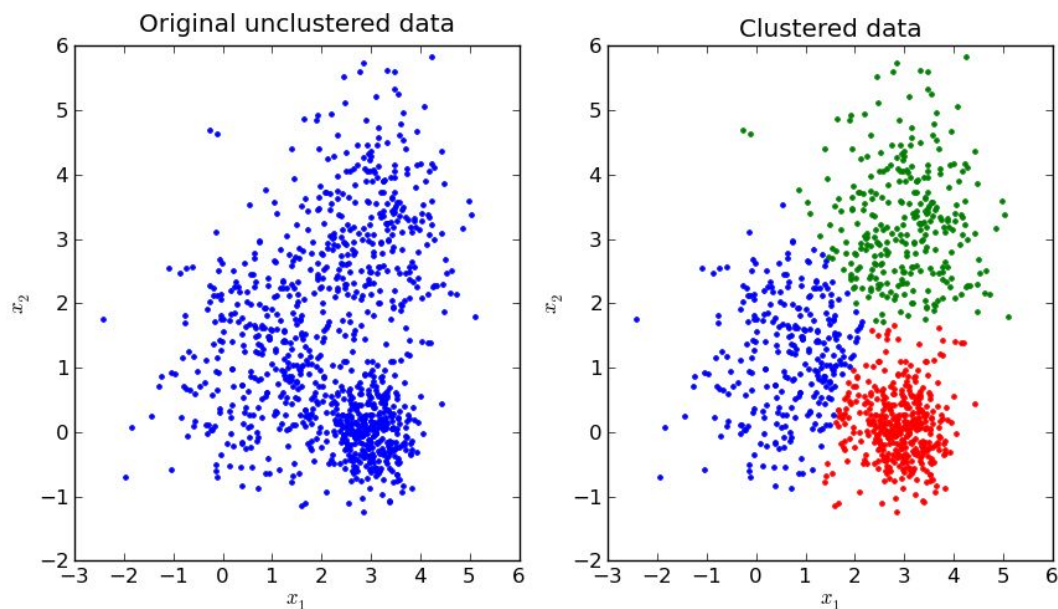
- *Marketing*: In marketing segmentation, when a company wants to segment its customers to better adjust products and offerings
- *Insurance*: Identifying groups of motor insurance policy holders with a high average claim cost.
- *City-planning*: Identifying groups of houses according to their house type, value, and geographical location.
- *Social network analysis*: Identifying possible friends as well as targeted marketing campaigns
- *Anomaly detection*: Identifying anomalies in any process with unsupervised learning

Friendly Introduction to clustering

So you know from the previous topic what are unsupervised methods and how those are different from supervised ones. In supervised learning, you have the labels which tell what to look for whereas in unsupervised learning it is the goal to group instances which are similar according to some predefined measure of similarity. In such a scenario, the features play a vital role (more than features do for supervised setting) and different combinations of features might result in different groups.

What is clustering?

The goal of unsupervised learning is to group similar instances together. In other words objects within a similar group are more similar as compared to objects in other groups. This is called clustering. The image below gives a glimpse of what the end goal is in clustering.



Desirable properties of clustering algorithm

- Scalability (in terms of both time and space)
- Ability to deal with different data types
- Minimal requirements for domain knowledge to determine input parameters
- Able to deal with noise and outliers
- Insensitive to order of input records
- Incorporation of user-specified constraints
- Interpretability and usability

Applications of clustering

- Preprocessing step for supervised methods
- Stand-alone tool to get insight into data
- Knowledge discovery in data (underlying patterns, rules etc)


Types of clustering and clustering algorithms

Soft and Hard clustering

While assigning instances to clusters there are two ways: An instance can belong to only one cluster; OR an instance can fall into a cluster with some probability. The first type is called *Hard clustering* and the second type is called *Soft clustering*.

At present there are more than 100 types of clustering variants. And as usual not all of them are that intuitive and widely used. We will be discussing only K-Means and Hierarchical clustering in this concept but its good to know that there also exist other types of clustering algorithms. Lets take a look at some of the well known types of clustering algorithms:

1. Connectivity models: They are based on the notion that the data points closer in data space exhibit more similarity to each other than the data points lying farther away. These models can follow two approaches. In the first approach, they start with classifying all data points into separate clusters & then aggregating them as the distance decreases. In the second approach, all data points are classified as a single cluster and then partitioned as the distance increases. Also, the choice of distance function is subjective. These models are very easy to interpret but lacks scalability for handling big datasets. Examples of these models are hierarchical clustering algorithm and its variants.
2. Centroid models: These are iterative clustering algorithms in which the notion of similarity is derived by the closeness of a data point to the centroid of the clusters. K-Means clustering algorithm is a popular algorithm that falls into this category. In these models, the no. of clusters required at the end have to be mentioned beforehand, which makes it important to have prior knowledge of the dataset. These models run iteratively to find the local optima.
3. Distribution models: These clustering models are based on the notion of how probable is it that all data points in the cluster belong to the same distribution (For example: Normal, Gaussian). These models often suffer from overfitting. A popular example of these models is Expectation-maximization algorithm which uses multivariate normal distributions.

QUESTIONS	YOUR ANSWER	CORRECT ANSWER
1. Which of the following algorithms are considered for unsupervised learning ?	 k-means	k-means Apriori
<u>Explanation:</u> k-means is used for clustering and apriori is used to derive association rules and both comes under unsupervised machine learning problems.		

Introduction to the problem statement

Context of the problem

One of the most practical use cases of clustering is segmenting customers based on data so that effective campaigning can be carried out. Identifying such groups can be very helpful as customized campaigns can be developed for each and every group. This results in better resource and time utilization besides revenue generation. The problem statement that you have at your disposal is something very similar.

About the dataset

You have an e-commerce dataset which contains the annual income and annual spend of 300 customers. It is an unlabelled dataset and you have been asked by the owner of the dataset to find specific groups of customers with the help of this data so that they can target customers according to individual properties of the groups. The dataset looks something like this:

	A	B
1	INCOME	SPEND
2	233	150
3	250	187
4	204	172
5	236	178
6	354	163
7	192	148
8	294	153
9	263	173
10	199	162
11	168	174
12	239	160
13	275	139
14	266	171
15	211	144
16	283	162
17	219	158

There are two columns `INCOME` and `SPEND` which describe the income and expenditure of customers respectively.

GOAL of the problem statement

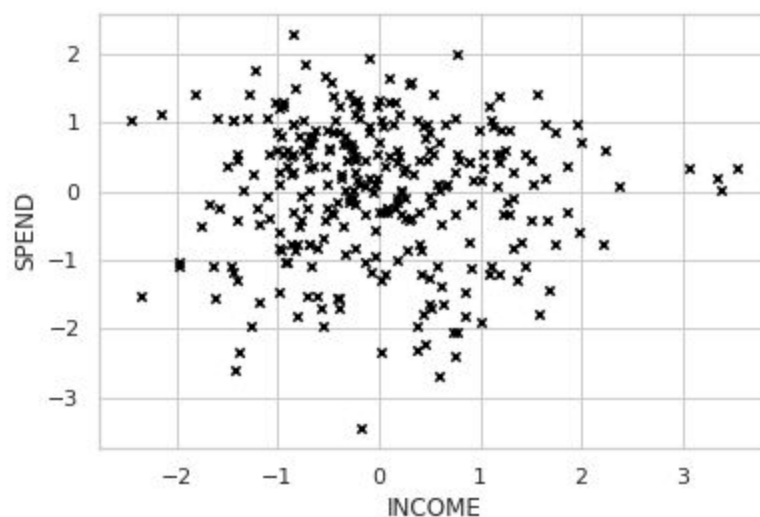
As already specified your main goal is to be able to form certain groups of customers which can be like low income high expenditure or low expenditure low income etc; but without taking the help of a rule-based approach.

Approach towards solving it

You will be taking the help of K-Means clustering to identify the presence of such groups.

How K-Means works

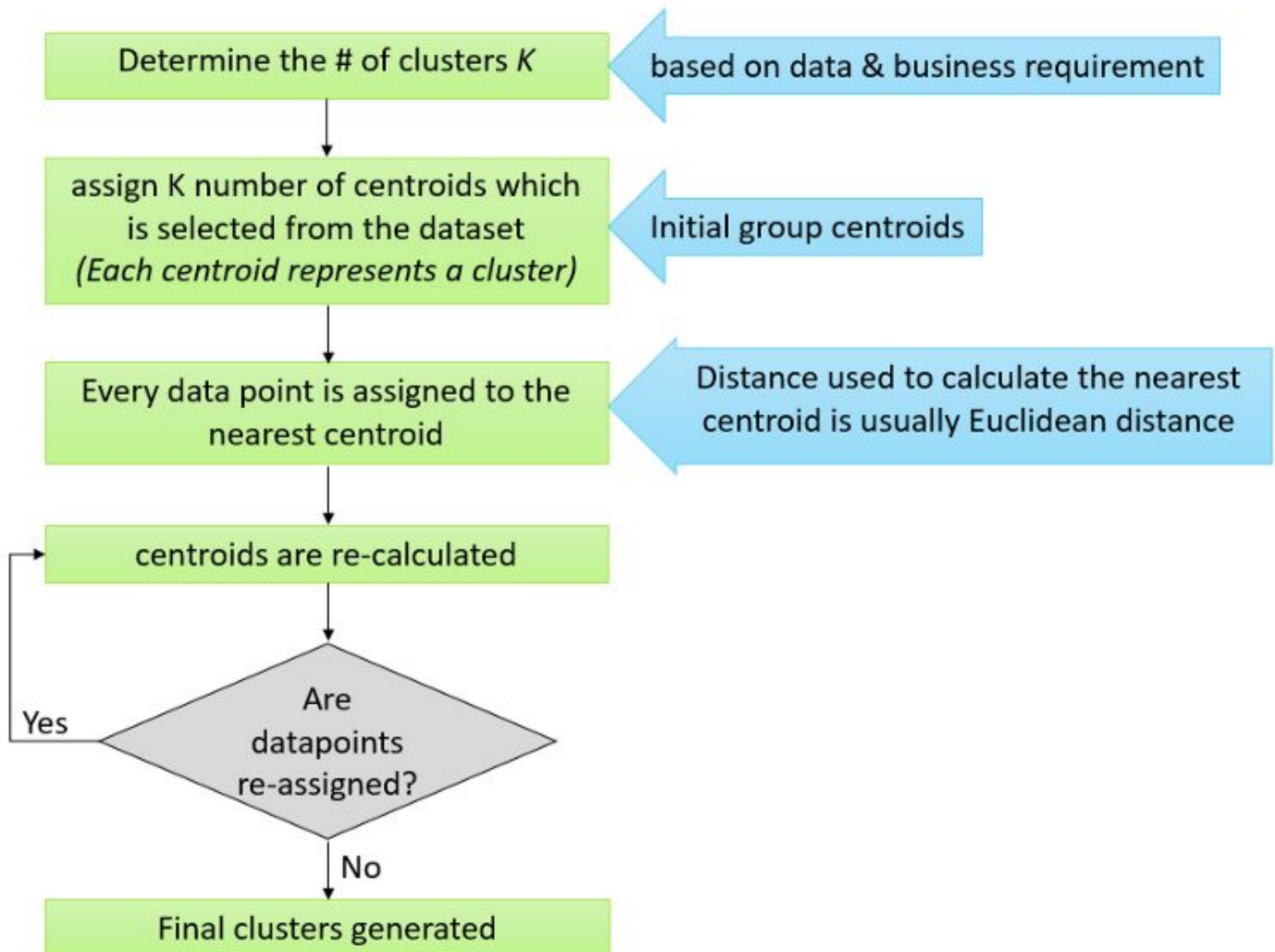
Lets look at the scatter plot of the data before actually proceeding to the working details of the algorithm. The data looks something like this:



Intuition behind K-Means

K-means is an iterative clustering algorithm where we predefine the number of clusters K and the algorithm iteratively assigns each data point to one of the K clusters based on some measure of distance between the data point and the cluster centroid. The pseudocode for K-Means is given below:

- First select a number of classes/groups to use (K) and randomly initialize their respective center points (Its good to take a quick look at the data and try to identify any distinct groupings to determine value of K).
- Each data point is classified by computing the distance between that point and each group center or centroid, and then classifying the point to the centroid closest to it. This step is also sometimes called the Expectation Step.
- Based on these classified points, we recompute the centroids by taking the mean of all the vectors in the group. This step is the Maximization step.
- Repeat these steps for a set number of iterations or until the centroids don't change much between iterations. You can also opt to randomly initialize the group centers a few times, and then select the run that looks like it provided the best results.



K-Means from scratch in Python

Below given is the code for K-Means from scratch on the given dataset. Here, we have taken random initialization for cluster centroids with K=3 and distance measure as the Euclidean distance.

```
# Let's consider a dataset with 2 features f_1 and f_2.
# We take the features f_1 and f_2 and convert it into a numpy array and store it in X

X = np.array(list(zip(f_1,f_2)))

# Euclidean Distance Calculator
def dist(a, b, ax=1):
    return np.linalg.norm(a - b, axis=ax)

# Number of clusters
k = 3

# X coordinates of random centroids
C_x = np.random.randint(0, np.max(X)-20, size=k)

# Y coordinates of random centroids
C_y = np.random.randint(0, np.max(X)-20, size=k)
C = np.array(list(zip(C_x, C_y)), dtype=np.float32)

# To store the value of centroids when it updates
C_old = np.zeros(C.shape)

# Cluster Labels(0, 1, 2)
clusters = np.zeros(len(X))

# Error func. - Distance between new centroids and old centroids
error = dist(C, C_old, None)

# Loop will run till the error becomes zero
while error != 0:
    # Assigning each value to its closest cluster
    for i in range(len(X)):
        distances = dist(X[i], C)
        cluster = np.argmin(distances)
        clusters[i] = cluster
    # Storing the old centroid values
    C_old = deepcopy(C)
    # Finding the new centroids by taking the average value
    for i in range(k):
        points = [X[j] for j in range(len(X)) if clusters[j] == i]
        C[i] = np.mean(points, axis=0)
    error = dist(C, C_old, None)
    print(error)
    print('='*20)
```

The output comes as:

```
246.82677
=====
31.00714
=====
19.493048
```

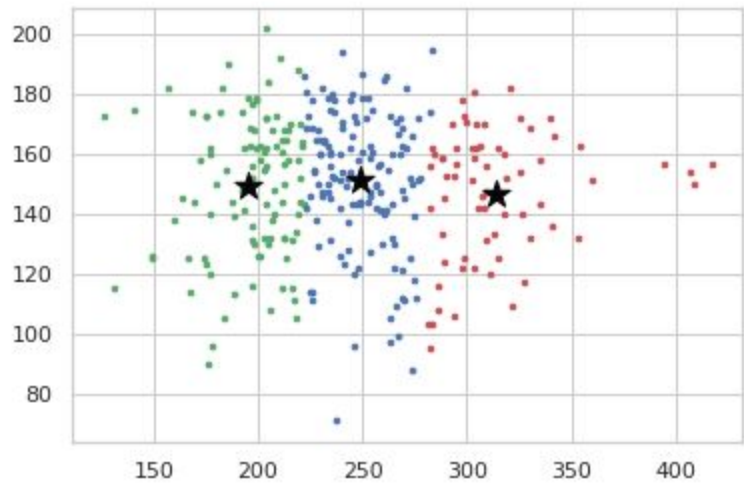


```
.....  
.....  
0.74065053  
=====
```

```
.....  
.....  
0.34212232  
=====
```

```
.....  
.....  
0.0  
=====
```

On plotting we observe the following 3 clusters:



This is how basically K-Means works and finds the underlying pattern within the data itself. In our case it was successful in creating 3 clusters and assigning every data point to one cluster.

Mathematics behind K-Means

Time to dive into the mathematics behind K-Means algorithm. Lets tackle it step-by-step so that you don't get stuck at any step.

Problem

- Given data points $x_1, x_2, x_3, \dots, x_n$ where $x \in R^d$, partition the data into groups called clusters where d is the number of features

Goal



- Observations in same groups are **similar** while those in different groups are **different** by setting a predetermined value for the number of clusters

More specifically, we have the following:

Input: $x_1, x_2, x_3, \dots, x_n$ where $x \in R^d$

Output: Vector c of cluster assignments and K mean vectors μ (denoting centroids). The vector c_i indicates the cluster to which the i^{th} observation falls.

where $c = (c_1, c_2, c_3, \dots, c_n)$ and $c_i \in \{1, 2, 3, \dots, K\}$

If $c_i = c_j = k$, then i -th and j -th observations are clustered in the cluster " k ".

And $\mu = (\mu_1, \mu_2, \dots, \mu_K)$ are the cluster centroids. There are K clusters and hence K centroids and $\mu_k \in R^d$ i.e. each centroid (μ_i) is in the same space as x_i .

OBJECTIVE FUNCTION

From the intuition part of K-Means it is pretty evident that your main goal is to minimize the euclidean distance between points within the same cluster.

Mathematically,

$$\mu^*, c^* = \underset{\mu, c}{*} \arg \min \sum_{i=1}^n \sum_{k=1}^K \mathbf{1}\{c_i = k\} ||x_i - \mu_k||^2$$

where μ^*, c^* are the optimum values of vectors c and μ respectively. Also observe the part $\{c_i = k\}$. It simply means if the point is the cluster " k ", then its value is 1; otherwise it is zero.

You can also consider using some other distance form instead of Euclidean distance, like Manhattan distance, cosine similarity, Pearson correlation etc.

Observations on the objective function

- It uses squared Euclidean distance as specified by the term $||x_i - \mu_k||^2$
- Penalizes distance of x_i to centroid assigned by c_i
- This objective function is non-convex and hence we cannot find optimal μ^*, c^*

Optimizing K-Means Objective Function

We cannot optimize the objective function by gradient descent as the function is non-convex. This is because the vector \mathbf{c} is a discrete valued vector and it doesn't make sense to find gradient of a discrete vector \mathbf{c} .

Since we cannot find best values for both the parameters simultaneously, we will use a different method called **Co-ordinate descent**. It works on the principle

- Fixing μ we can find best \mathbf{c} exactly
- Fix \mathbf{c} we can find best μ exactly

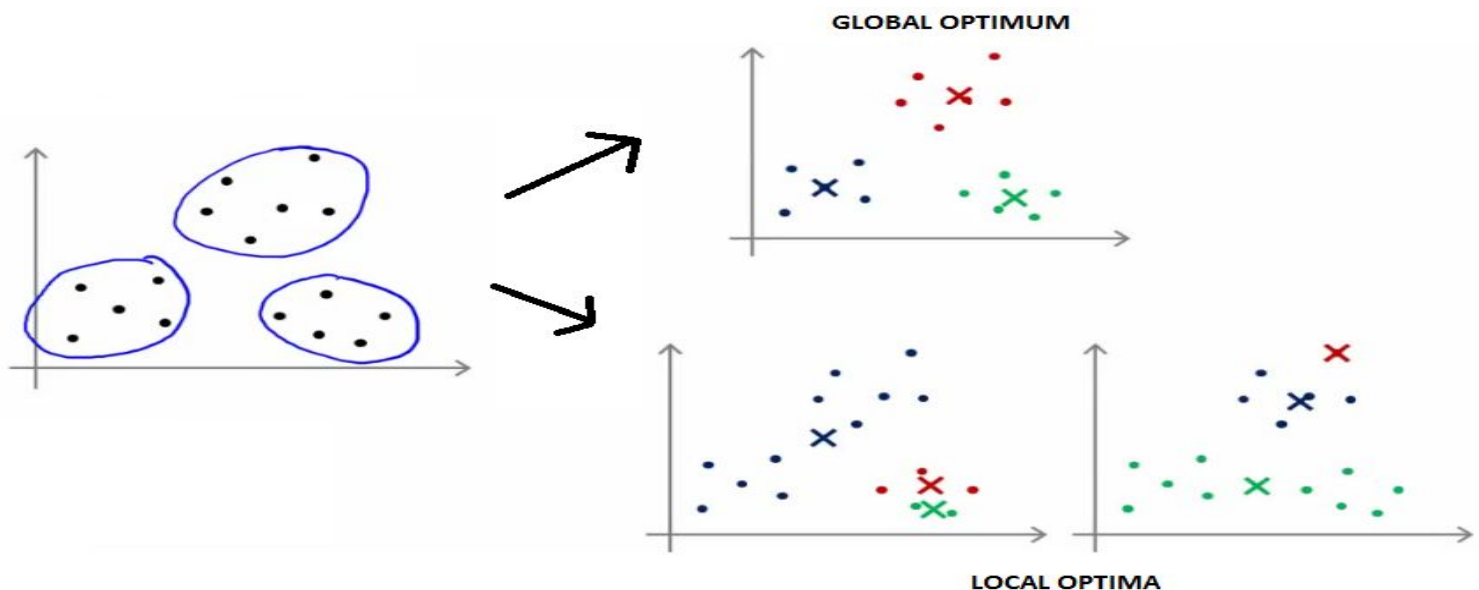
So, what we do is basically:

- With given μ , find best value for $c_i \in \{1, 2, 3, \dots, K\}$ for $i = 1, 2, 3, \dots, n$ (Fixing μ and finding best \mathbf{c})
- For values of \mathbf{c} obtained from previous step, find best vector $\mu_k \in \mathbb{R}^d$ for $k = 1, 2, 3, \dots, K$ (Fixing \mathbf{c} and finding best μ)

K-Means is guaranteed to converge to a minimum (even though local but not global) because every update decreases the objective function and this function is monotonically decreasing.

What does non-convexity of the objective function mean?

It simply means that with different initializations of cluster centroids, we can get different results. Often results obtained with different initializations will be similar in quality i.e. loss would be in the same range and neither too low or too high. But it comes with no guarantee. That is why for practical reasons, it is advised to run K-Means multiple times with different initializations and then choose the setting which gives a lower loss. The below image captures exactly what the problem is with our non-convex objective function.



Scikit-learn implementation of KMeans

With scikit-learn your KMeans implementation becomes very easy as it hides the underlying complications like defining the objective function, iterating over data points, assigning cluster centroids and updating them etc and provides a clean, robust API to work upon. Have a look at the Python code snippet below and you will understand

```
# import packages
from sklearn.cluster import KMeans

# Initialize K-means algorithm
km = KMeans(n_clusters=3, init='k-means++', max_iter=300, n_init=10, random_state=0)
km.fit(df)

# Cluster centers
print("Cluster centers are:", km.cluster_centers_)
print('='*20)

# Within cluster sum of squares
print("Within cluster sum of squares is:", km.inertia_)
```

The output is :

```
Cluster centers are: [[196.68224299 150.57943925]
 [317.55          148.73333333]
 [251.61764706 149.31617647]]
=====
Within cluster sum of squares is: 297101.3764201943
```

We encourage you to go through its official [documentation](#) and try out more such variants.

Here,

- `n_clusters`: Number of clusters (K)
- `init`: Choice of initialization of cluster centroids; `k-means++` is considered a smart way to initialization
- `n_iter`: Number of time the k-means algorithm will be run with different centroid seeds
- `max_iter`: Maximum number of iterations of the k-means algorithm for a single run

Use scikit-learn to form clusters

In this task you will use scikit-learn to form 6 clusters on the dataset. You will also look at the cluster centroids and the within cluster sum of squares (WCSS)

Instructions

- Initialize a Kmeans object `km` with `n_clusters=6`, `init='k-means++'`, `max_iter=300`, `n_init=10` and `random_state=0`
- Fit this `km` object on the dataframe `df`
- Save your cluster centroid co-ordinates as `centroids` using `.cluster_centers_` attribute of `km`
- Save your WCSS as `wcss` using `.inertia_` attribute of `km`
- Print out `centroids` and `wcss`

Skills Covered:

Machine Learning

```
2 from sklearn.cluster import KMeans
3
4 # Code starts here
5 km = KMeans(n_clusters=6, init='k-means++', max_iter=300, n_init=10, random_state=0)
6 km.fit(df)
7 # Cluster centers
8 centroids = km.cluster_centers_
9 print("Cluster centers are:", centroids)
10 # Within cluster sum of squares
11 wcss = km.inertia_
12 print("Within cluster sum of squares is:", wcss)
```

OUTPUT

RESULT

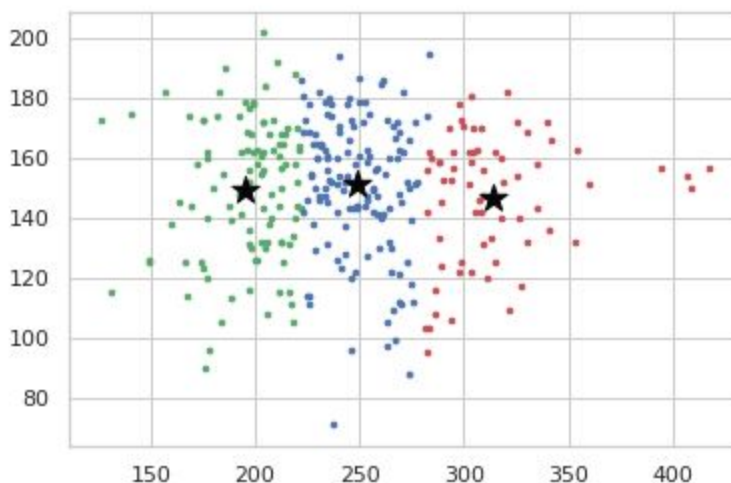
```
Cluster centers are: [[265.58333333 117.44444444]
 [193.93478261 125.63043478]
 [301.07142857 152.91071429]
 [195.05769231 167.82692308]
 [242.03960396 160.46534653]
 [365.58333333 153.25      ]]
Within cluster sum of squares is: 158865.83467962776
```

Practical considerations for K-Means

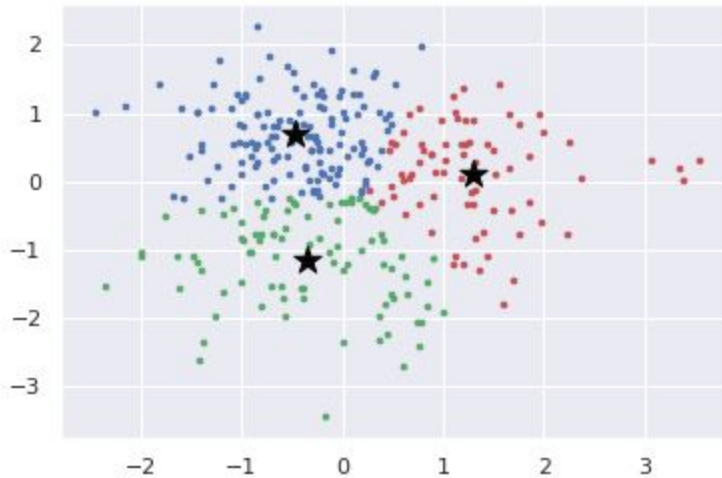
There are many practical considerations while deciding to go for K-Means clustering. Lets go through them in details:

1. **Non-convexity:** In the previous topic, you saw that the objective function is non-convex which means gradient descent is not guaranteed to converge to the global minima. That is why different set of initializations for cluster centroids and also varying the number of clusters will give different outputs. It is best described with the image below.
2. **Handling missing values:** KMeans cannot handle missing values and it must be taken care of before going forward with KMeans
3. **Categorical variables treatment:** KMeans computes distance at its core, so its necessary to convert categorical variables into numerical so that KMeans works properly.
4. **Data Normalization:** Distance computation in KMeans weights each dimension equally and hence care must be taken to ensure that unit of dimension shouldn't distort relative nearness of observations. The standard example is considering age (in year) and height (in cm). The age may range in [20 60], while the height may range in [120 180]. If you use the classical Euclidean distance, the height will have dispoportionately more importance in its computation with respect to the age. But data normalization does not necessarily improve performance; its generally considered good practice. The before and after data normalization images are shown below and look at the evident shape difference between clusters. Also, with data normalization the solution was much faster as compared with the regular data.

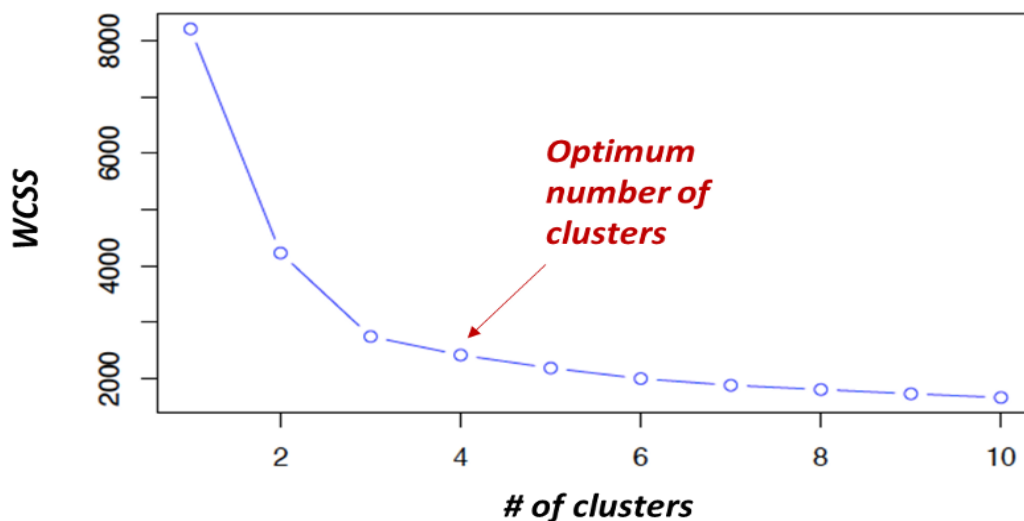
BEFORE DATA NORMALIZATION



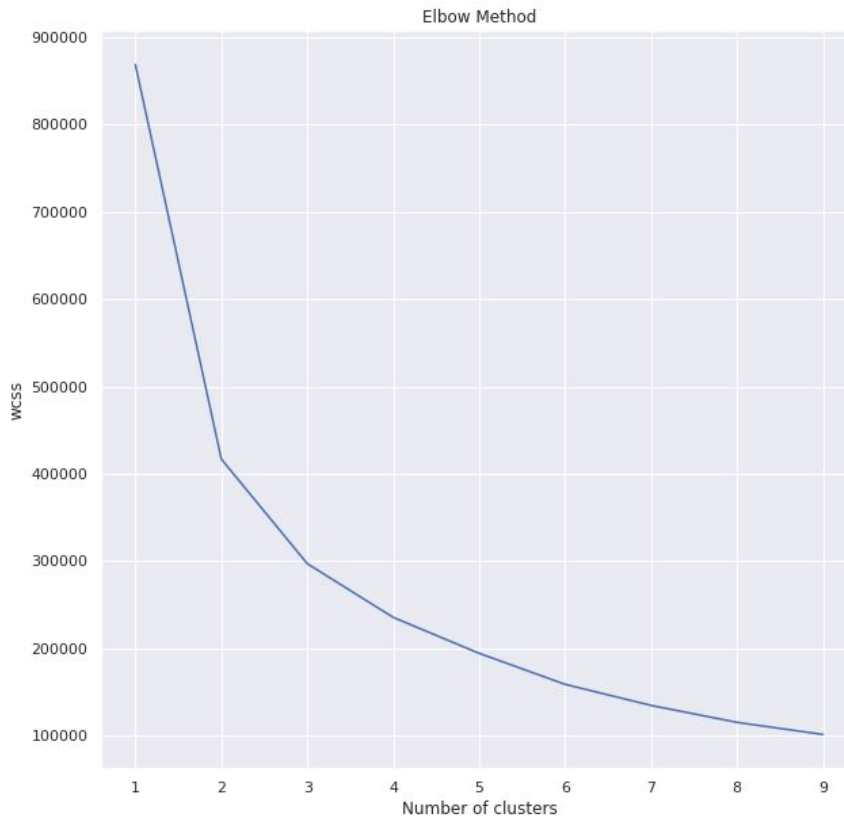
AFTER DATA NORMALIZATION



5. Choice of the number of clusters (K): This is perhaps the burning question while performing KMeans. What is a good value of the number of clusters? Intuitively speaking, increasing the number of clusters will result in lower loss. For ex: if the number of data points equals the number of clusters, the loss would be zero. Therefore, it is absurd to compare the loss functions across different settings of the number of clusters (K). Also, one should not set the number of clusters equal to the number of data points as then it would defeat the purpose of modelling. So, how to choose the value for K?
- *Business Knowledge* : Having a sound domain knowledge gives a fair idea of what an ideal setting for the number of clusters would be. It will save unnecessary time going back and forth running through various iterations and will obviously yield far greater insights than any other variant
 - *Elbow Method* : This is the main input which ultimately decides the output of the clustering method. Since we don't have any beforehand idea or at best have some initial guess about it, its actually possible to generate a plot called scree plot which can give us the number of clusters. It is a plot where number of clusters (K) is in the X-axis and within cluster sum of squares (WCSS) is on the Y-axis. It is shown below:



You know by now that as the number of clusters increase, the WCSS keeps decreasing. This decrease of WCSS is initially steep and then the rate of decrease slows down resulting in an elbow shape of the plot. The number of clusters at the elbow formation usually gives an indication on the optimum number of clusters. This combined with specific knowledge of the business requirement should be used to decide on the optimum number of clusters. With our dataset the elbow plot will look something like this:



Going by the definition of elbow methods we can try out either 4, 5 or 6 clusters.

Elbow method to find optimal number of clusters

In this task you will use the elbow method to find the optimum number of clusters

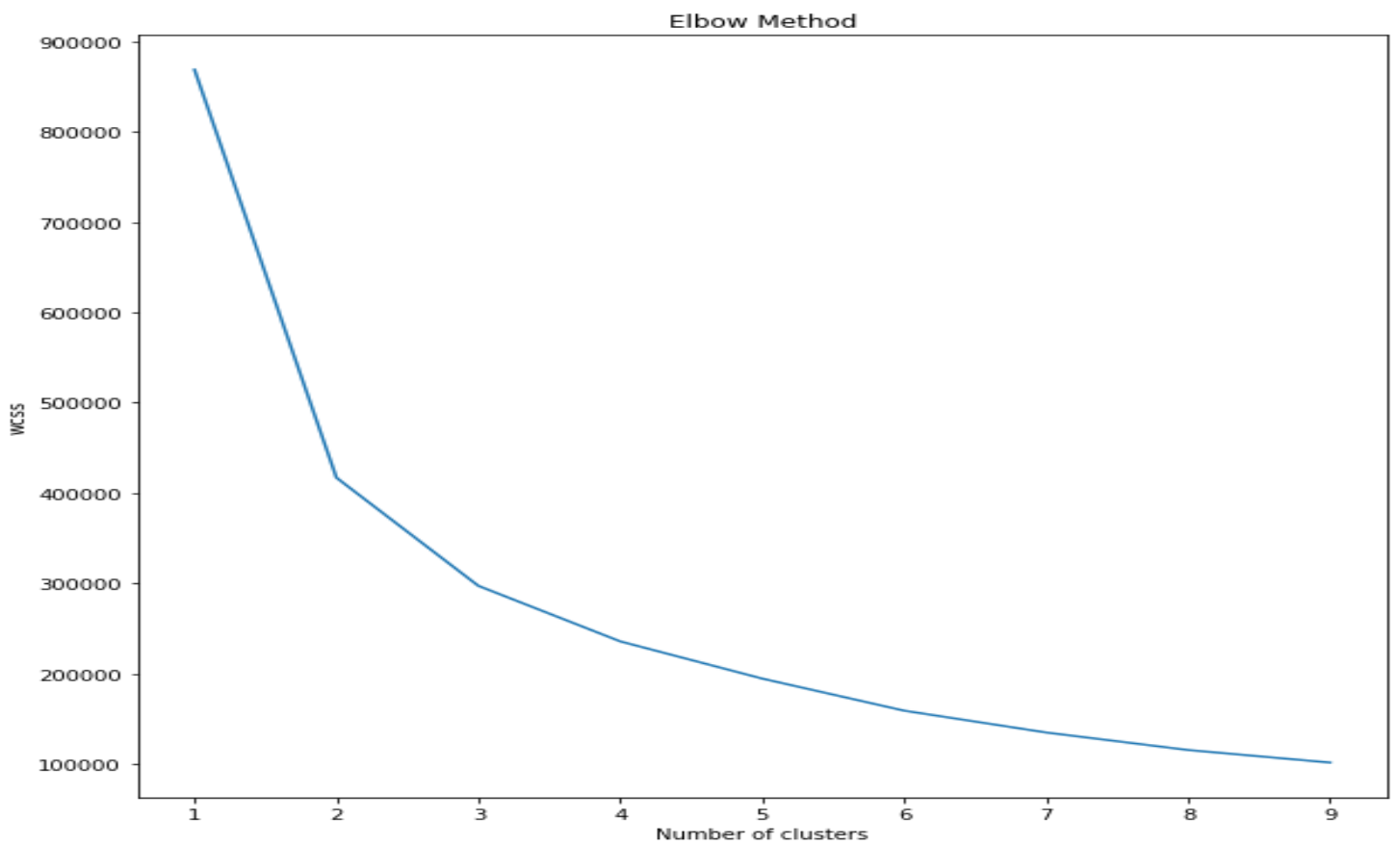
Instructions

- Initialize an empty list for storing WCSS across all possible values of k `dist`
- Iterate from 1 to 10 i.e. you will be calculating the WCSS for k=1 to k=9 using for loop
- Inside the loop initialize a KMeans object `km` as `KMeans(n_clusters=i,init='k-means++', max_iter=300, n_init=10, random_state=0)` and then fit it on `df` using `.fit()` method
- Now append the corresponding WCSS to `dist` using `.append(km.inertia_)`
- Now initialize a figure using matplotlib of any size
- Use `.plot()` method to generate a line plot with number of clusters on X-axis (use `range(1,10)`) and WCSS distances given by `dist` on Y-axis

```

2 from sklearn.cluster import KMeans
3 # Code starts here
4 # Empty list for storing WCSS across all values of k
5 dist = []
6 # Iterate from 1-9
7 for i in range(1,10):
8     # Initialize KMeans algorithm
9     km=KMeans(n_clusters=i,init='k-means++', max_iter=300, n_init=10, random_state=0)
10    # Fit on data
11    km.fit(df)
12    # Append WCSS to list storing WCSS
13    dist.append(km.inertia_)
14 # Initialize figure
15 plt.figure(figsize=(10,10))
16 # Line plot # clusters on X-axis and WCSS on Y-axis
17 plt.plot(range(1,10),dist)
18 plt.title('Elbow Method')
19 plt.xlabel('Number of clusters')
20 plt.ylabel('wcss')
21 plt.show()

```



Trying with six clusters and interpreting the result

On trying with 6 clusters on our dataset, we obtain the following image where the cluster centroids are depicted by red squares and data points by colored dots.



Interpretation



Since data is two dimensional, we can actually plot it and interpret it very easily. In cases where input data is multi-dimensional, it is very hard to visualize and interpret and we often take help of dimensionality reduction techniques like PCA, SVD etc for interpretation. So, the six clusters at our disposal can be interpreted in the following manner:

- Cluster 1: Medium income and low annual spend
- Cluster 2: Low income and low annual spend
- Cluster 3: High income and high annual spend
- Cluster 4: Low income and high annual spend
- Cluster 5: Medium income and low annual spend
- Cluster 6: Very high income and high annual spend

With the help of this interpretation one can effectively design strategies. For example:

- Cluster 4 can be branded as loyal customers where customers in spite of low income spend heavily
- For Cluster 2 category customers special incentivised plans need to be formed so that spend increases

Do you now appreciate the beauty behind KMeans algorithms??

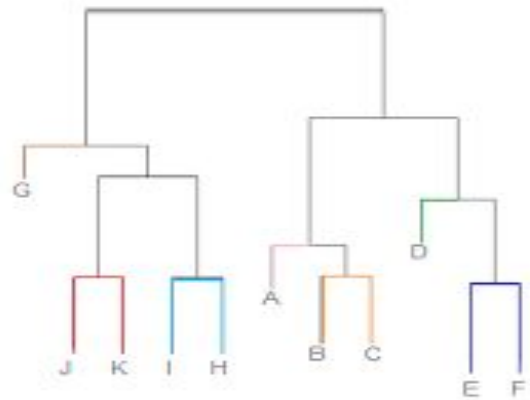
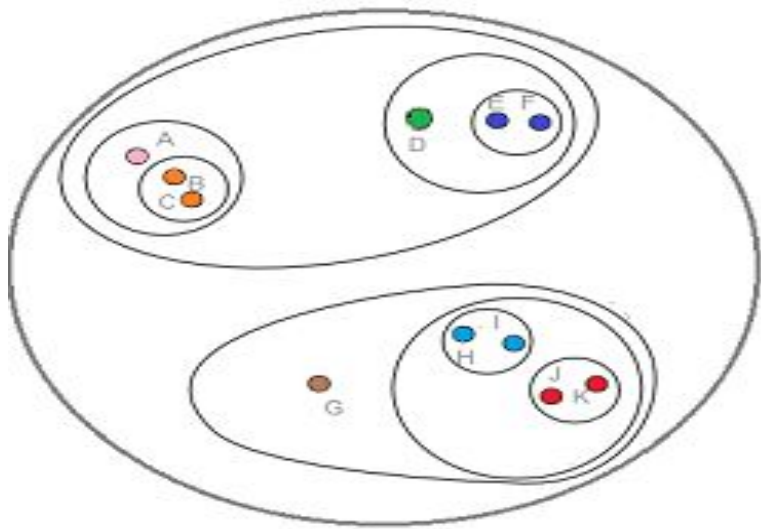
QUESTIONS	YOUR ANSWER	CORRECT ANSWER
1. Which of the following algorithm has similarity with K-Means?	 K-NN	K-NN
<u>Explanation:</u> Both are used to classify the data into different segments but one is used for unsupervised method and another is used for supervised method.		
2. K-means is not deterministic and it also consist of number of iterations.	 True	True
<u>Explanation:</u> K-means clustering produces final estimate of cluster centroids.		

What is Hierarchical clustering

As the name itself suggests, hierarchical clustering does the job of building a hierarchy of clusters using a tree based structure called the dendrogram. For example, all files and folders on the hard disk are organized in a hierarchy of groups.

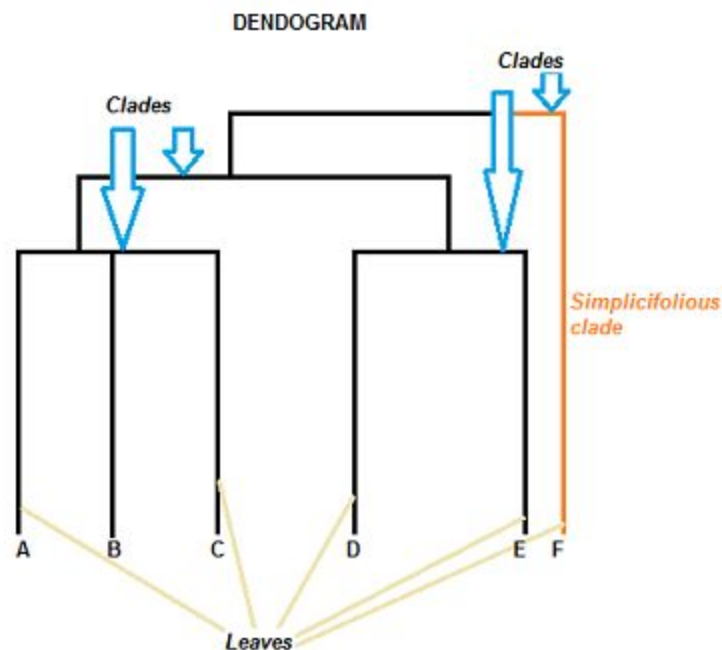
Before proceeding to the types of hierarchical clustering, first lets understand the dendrogram and the terminologies associated with it

Dendrogram



A dendrogram is a type of tree diagram showing hierarchical clustering – relationships between similar sets of data. They are frequently used in biology to show clustering between genes or samples, but they can represent any type of grouped data. As shown in the image above, hierarchical clustering presents us with a convenient way to represent nested clusters and is shown by the dendrogram (right figure).

Parts of Dendrogram



Above shown is the structure of a dendrogram. The parts are discussed below:

- Clade: It is the branch
- Leaves: Each clade has one or more leaves. The leaves in the above image are:
 - *Single (simplicifolius)*: F
 - *Double (bifolius)*: D E
 - *Triple (trifolious)*: A B C

The clades are arranged according to how similar (or dissimilar) they are. Clades that are close to the same height are similar to each other; clades with different heights are dissimilar — the greater the difference in height, the more dissimilarity (you can measure similarity in many different ways).

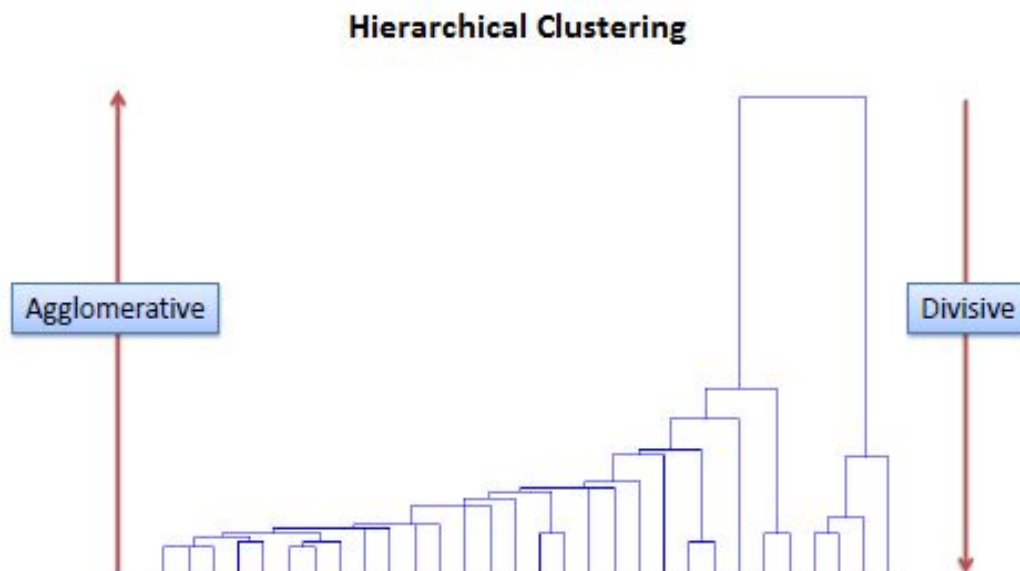
Some observations from the dendrogram above are:

- Leaves A, B, and C are more similar to each other than they are to leaves D, E, or F.
- Leaves D and E are more similar to each other than they are to leaves A, B, C, or F.
- Leaf F is substantially different from all of the other leaves.

Types of hierarchical clustering

There are two types of hierarchical clustering algorithms:

- Agglomerative clustering (Bottom up)
- Divisive clustering (Top down)



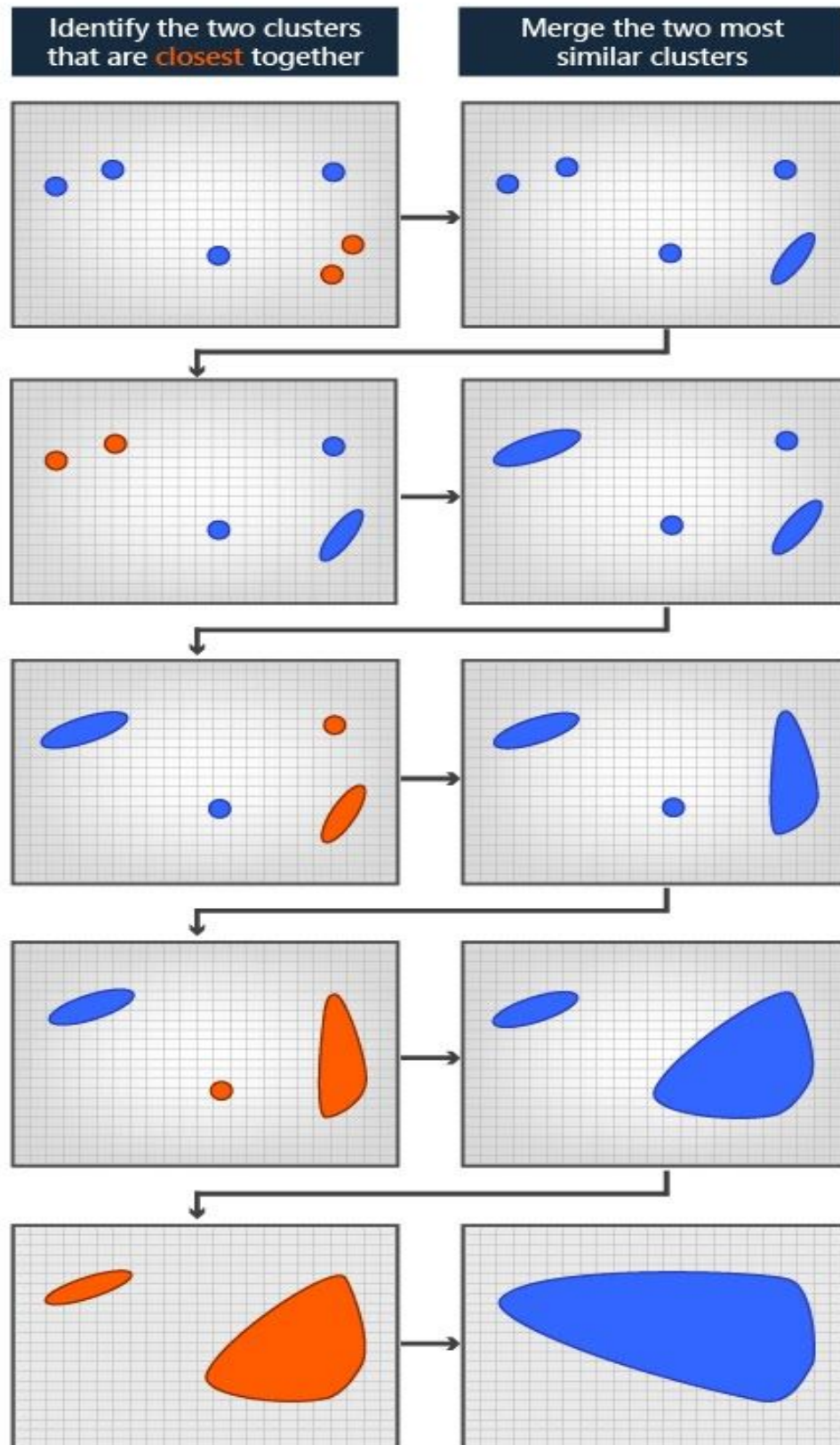
The above image shows both these types in a single figure. Agglomerative follows a bottom up approach while divisive follows a top down approach and is shown in the image with the help of arrows. The direction of the arrows suggest the direction in which both the types move.

Working of hierarchical clustering types

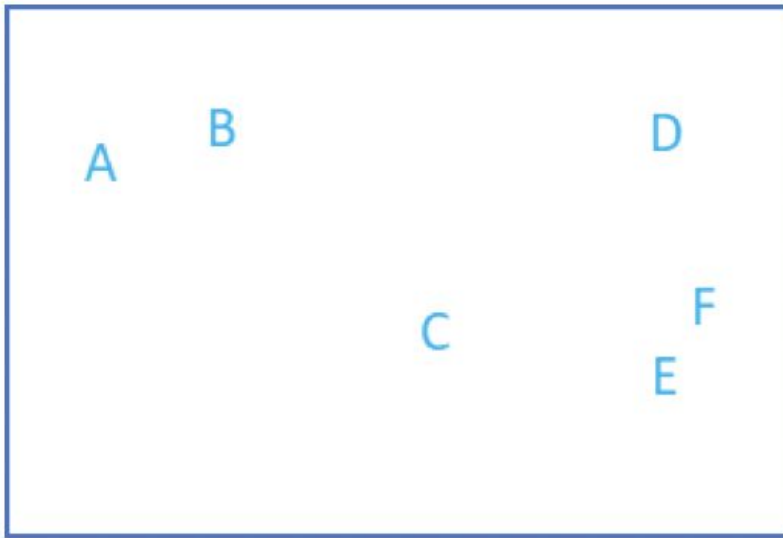
1. *Divisive clustering*: The steps involved in this method are:
 - Data starts as one combined cluster.
 - Cluster splits into two distinct parts, according to some degree of similarity.
 - Clusters split into two again and again until the clusters only contain a single data point.

Although there has been evidence that divisive algorithms produce more accurate hierarchies than agglomerative algorithms in some circumstances; but is conceptually more complex.

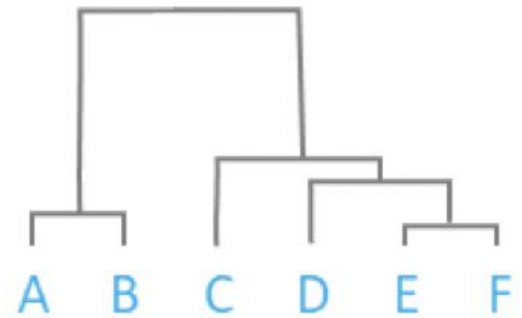
2. *Agglomerative clustering*: It starts by treating each observation as a separate cluster and then repeatedly executes the following two steps:
- Identify two clusters that are closest together
 - Merge the two most similar clusters. This continues until all the clusters are merged together.
- This is illustrated in the diagrams below.



As you can clearly observe from the following sets of images, there are clusters within clusters and the final output of Hierarchical Clustering can be represented by a dendrogram, which shows the hierarchical relationship between the clusters in the following manner:



Dendrogram



The pseudocode will help you in case you want to implement such method from scratch (not advised unless you are extraordinarily motivated):

Given:

A set X of objects $\{x_1, \dots, x_n\}$

A distance function $dist(c_1, c_2)$

for $i = 1$ to n

$c_i = \{x_i\}$

end for

$C = \{c_1, \dots, c_n\}$

$l = n+1$

while $C.size > 1$ **do**

$(c_{min1}, c_{min2}) = \text{minimum } dist(c_i, c_j) \text{ for all } c_i, c_j \text{ in } C$

 remove c_{min1} and c_{min2} from C

 add $\{c_{min1}, c_{min2}\}$ to C

$l = l + 1$

end while

Important parameters for Agglomerative clustering

To recap about the working of Agglomerative clustering,

- Start off by considering every data point as a single cluster i.e if there are n data points in our dataset then we have n clusters. Then select an appropriate distance metric that measures the distance between two clusters.
- On each iteration we combine two clusters into one. The two clusters to be combined are selected as those with the smallest selected distance metric. That is, these two clusters have the smallest distance between each other and therefore are the most similar and should be combined.
- Previous step is repeated until we reach the root of the tree i.e we only have one cluster which contains all data points.

Notice the word in bold i.e. **DISTANCE METRIC** or more commonly called as the **SIMILARITY METRIC**. Most often it is taken as the Euclidean distance between two points but it can also take other forms like Manhattan distance, cosine similarity etc.

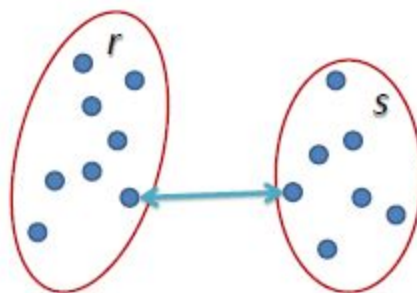
Choice of similarity metric

The choice of distance/similarity metric should be made based on theoretical concerns from the domain of study. It means that distance metric needs to define similarity in a way that is sensible for the field of study. For example, if clustering crime sites in a city, city block distance may be appropriate (or, better yet, the time taken to travel between each location). Where there is no theoretical justification for an alternative, the Euclidean should generally be preferred, as it is usually the appropriate measure of distance in the physical world.

Linkage criteria

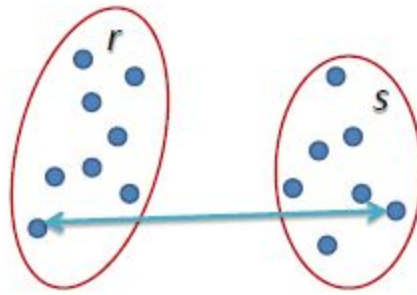
Once you have decided on a similarity metric the next thing to focus on is how this distance/similarity is to be calculated or rather from which point to which point this metric is to be calculated. Since in Agglomerative clustering we are joining clusters as we proceed forward, and the clusters contain data points, the distance between clusters can be calculated in a variety of ways. For example we can take the nearest two points from each of the two clusters. Now let's discuss about the different types of linkage:

- **Single linkage:** Here, the distance between two clusters is defined as the shortest distance between two points in each cluster. For example, the distance between clusters "r" and "s" to the left is equal to the length of the arrow between their two closest points.



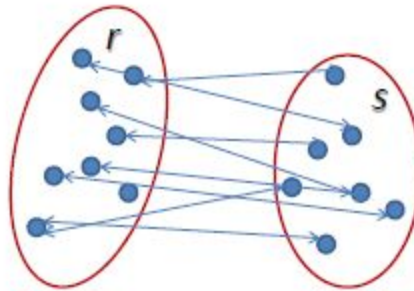
$$L(r, s) = \min(D(x_{ri}, x_{sj}))$$

- Complete linkage: Here the distance between two clusters is defined as the longest distance between two points in each cluster. For example, the distance between clusters "r" and "s" to the left is equal to the length of the arrow between their two furthest points.



$$L(r, s) = \max(D(x_{ri}, x_{sj}))$$

- Average linkage: In average linkage hierarchical clustering, the distance between two clusters is defined as the average distance between each point in one cluster to every point in the other cluster. For example, the distance between clusters "r" and "s" to the left is equal to the average length each arrow between connecting the points of one cluster to the other.

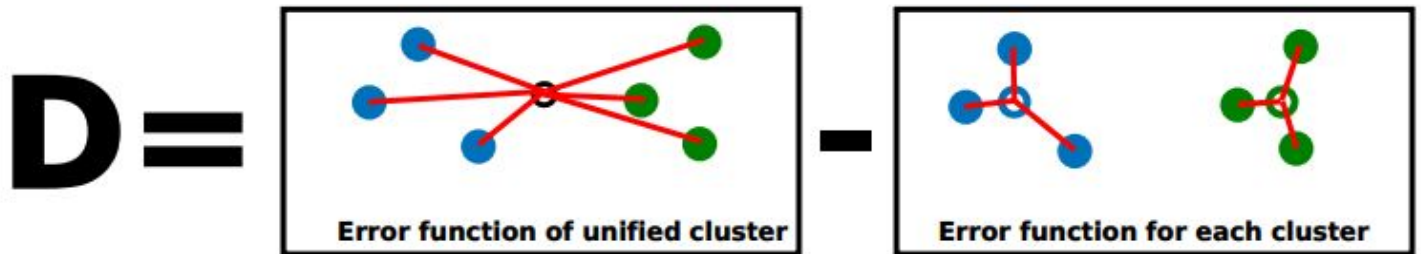


$$L(r, s) = \frac{1}{n_r n_s} \sum_{i=1}^{n_r} \sum_{j=1}^{n_s} D(x_{ri}, x_{sj})$$

- Ward linkage: In the ward linkage two errors are calculated:
 - Error Sum of Squares (ESS): To find it, we sum over all variables, and all of the units within each cluster. We compare individual observations for each variable against the cluster means for that variable. Note that when the Error Sum of Squares is small, it suggests that our data are close to their cluster means, implying that we have a cluster of like units.
 - Total Sum of Squares (TSS): Here, we compare the individual observations for each variable against the grand mean for that variable. Basically it implies taking considering two clusters as one, recomputing the cluster center and then finding the ESS.

Using the ESS and TSS we then calculate the value of r^2 where $r^2 = \frac{TSS - ESS}{TSS}$

This r^2 value is interpreted as the proportion of variation explained by a particular clustering of the observations. Two clusters are merged based on which combination yields the lowest value of ESS or the largest value of r^2 .



Agglomerative clustering with scikit-learn

With scikit-learn, it's very easy to implement agglomerative clustering. In fact you can also visualize the dendrogram generated which can give you further insights on how data is clustered.

```
# import packages
import scipy.cluster.hierarchy as sch

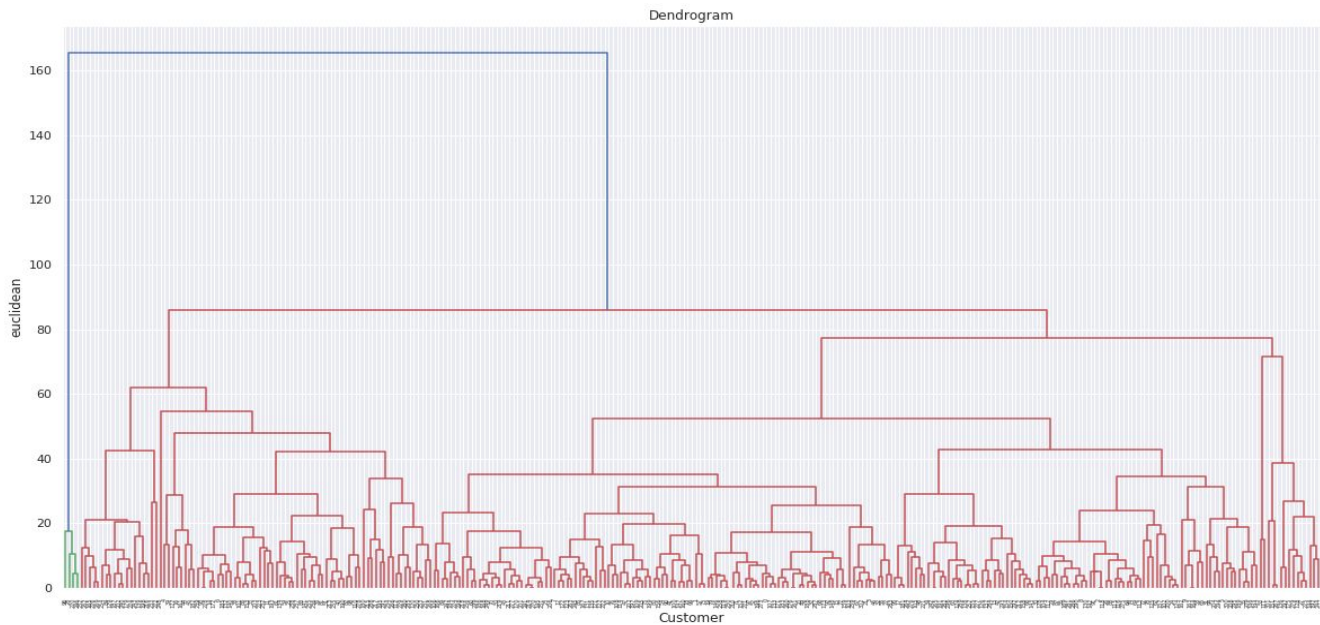
# initialize figure and axes
fig, ax_1 = plt.subplots(figsize=(20,10))

# dendrogram with "ward" linkage
dend = sch.dendrogram(sch.linkage(df, method='single'), ax=ax_1)

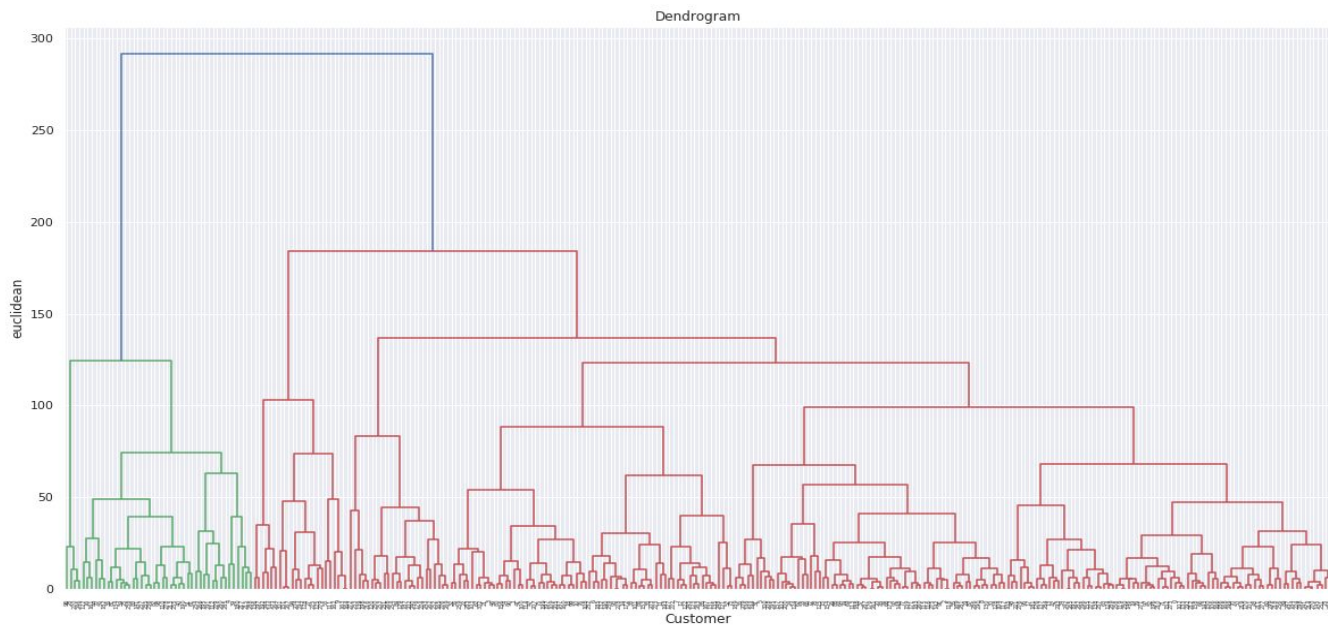
# plot on a figure
ax_1.set_title("Dendrogram")
ax_1.set_xlabel('Customer')
ax_1.set_ylabel('euclidean')
plt.show()
```

Notice line number 6 where select the method for `dend` variable. You can select any of the above discussed linkage types and arrive at totally different results. We will show you two images; one with average and the other with complete linkage below:

DENDROGRAM WITH AVERAGE LINKAGE



DENDROGRAM WITH COMPLETE LINKAGE



Generate dendrogram

In this task you will use Euclidean distance with Ward's linkage to cluster your data points

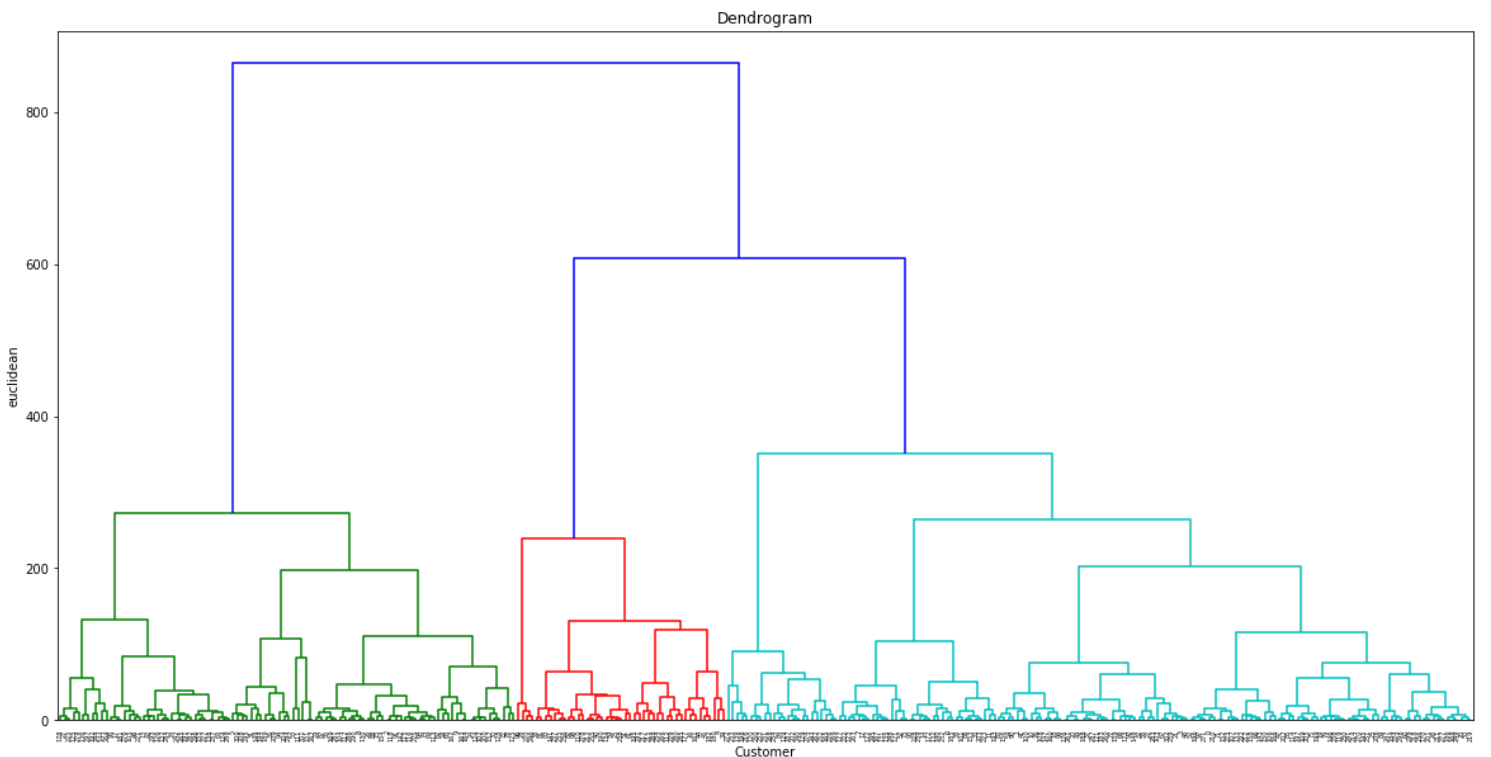
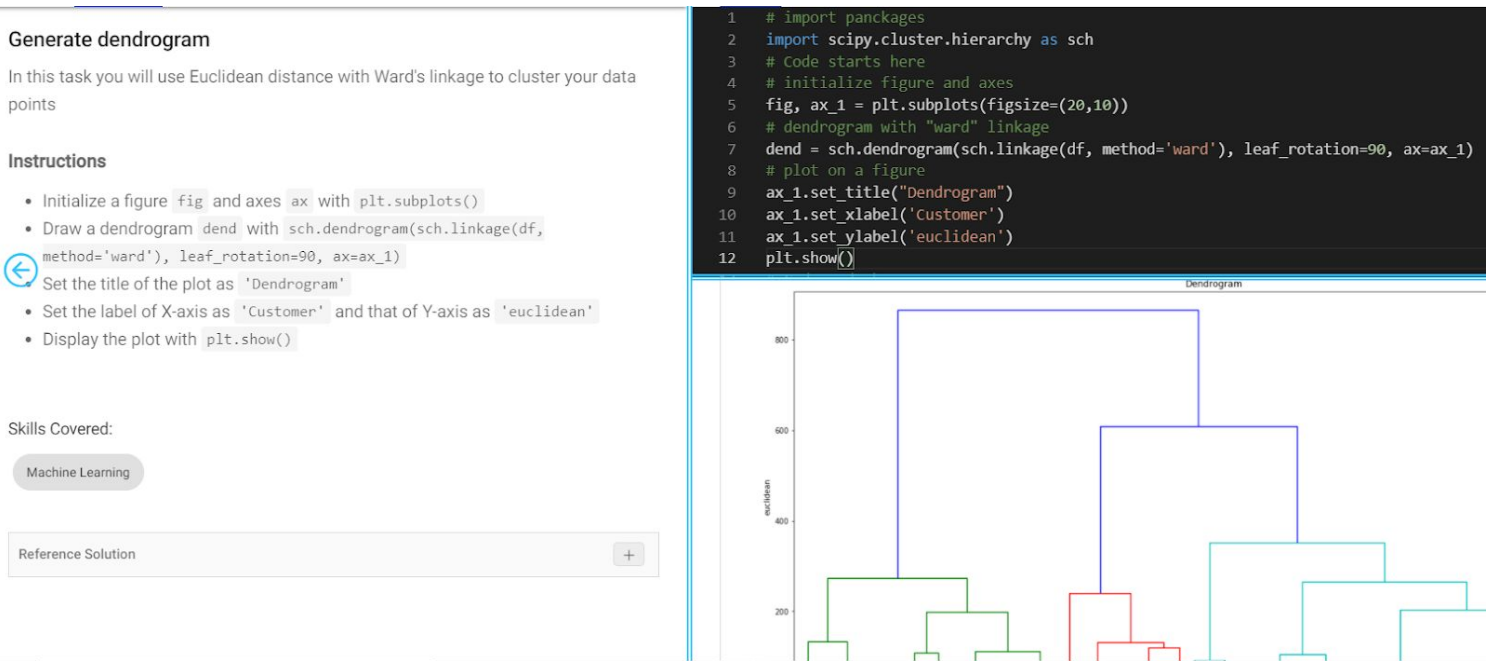
Instructions

- Initialize a figure `fig` and axes `ax` with `plt.subplots()`
- Draw a dendrogram `dend` with `sch.dendrogram(sch.linkage(df, method='ward'), leaf_rotation=90, ax=ax_1)`
- Set the title of the plot as `'Dendrogram'`
- Set the label of X-axis as `'Customer'` and that of Y-axis as `'euclidean'`
- Display the plot with `plt.show()`

Skills Covered:

Machine Learning

Reference Solution



Practical considerations with Agglomerative clustering

So now that you have a fair idea about how to go about performing Bottom-up hierarchical clustering, let's discuss some of the finer details.

Finding the optimum number of clusters

You had the elbow method which served as a guide for selecting the number of clusters in KMeans. But unlike KMeans where you had to select the number of clusters before running the algorithm, hierarchical clustering does not require us to specify the number of clusters and we can even select which number of clusters we require after taking a look at the dendrogram generated.

Like the elbow method, the method for finding optimum number of clusters in hierarchical clustering is not a definitive answer since cluster analysis is essentially an exploratory approach; the interpretation of the resulting hierarchical structure is context-dependent and often several solutions are equally good from a theoretical point of view.

The best choice of the number of clusters is the number of vertical lines in the dendrogram cut by a horizontal line that can transverse the maximum distance vertically without intersecting a cluster. Look at the image below and observe the horizontal red line; it covers the maximum vertical distance without intersecting any other cluster. So, the number of desired clusters can be considered as 3.

After having decided the number of clusters, you would want to cluster individual data points and if possible visualize with a scatter plot. The code for our dataset (with Ward's linkage and Euclidean distance as similarity) as well as the image is given.

CODE:

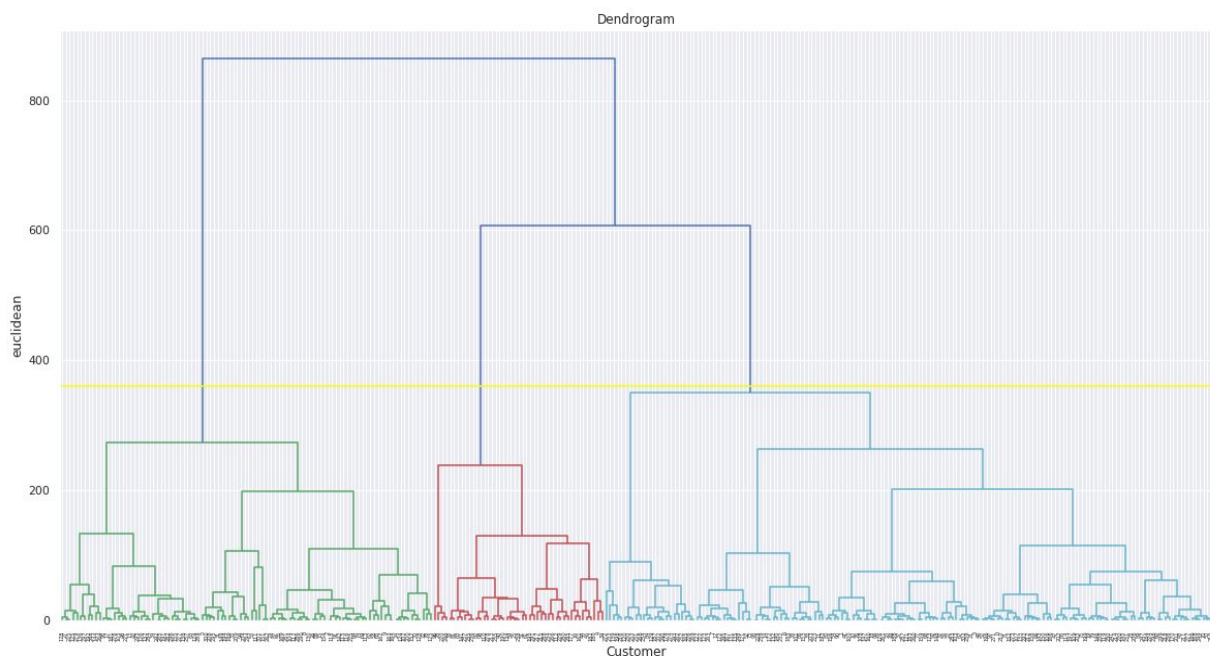
```
# import packages
from sklearn.cluster import AgglomerativeClustering

# initialize Agglomerative clustering object with 3 clusters
cluster = AgglomerativeClustering(n_clusters=3, affinity='euclidean', linkage='ward')

# predict on dataset
cluster.fit_predict(df)

# scatter plot
plt.figure(figsize=(10, 7))
plt.scatter(df.iloc[:,0], df.iloc[:,1], c=cluster.labels_, cmap='rainbow')
```

IMAGE:



KMeans vs Agglomerative clustering

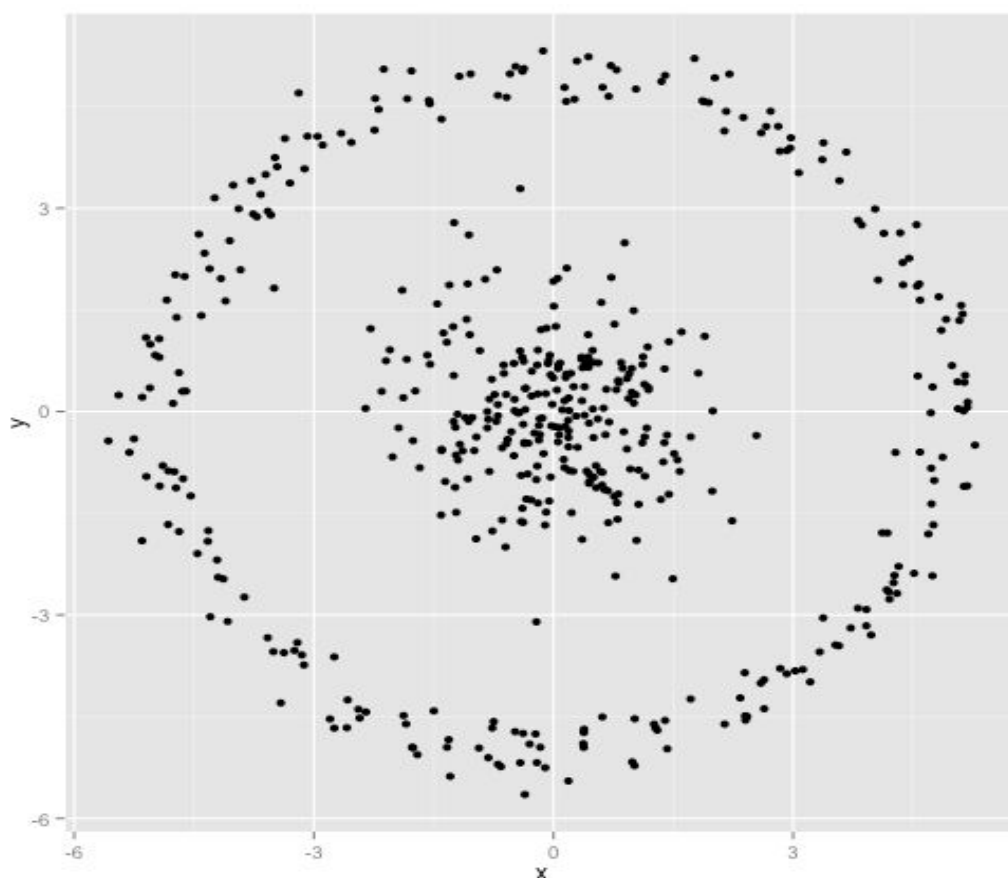
- KMeans has a linear time complexity i.e. $O(n)$ while that of hierarchical clustering is $O(n^2)$
- That's why hierarchical can't handle big data well but KMeans clustering can.
- Results might differ in KMeans depending on cluster center initialization but hierarchical clustering can produce reproducible results.
- KMeans clustering requires prior knowledge of the number of clusters k . But, you can stop at whatever number of clusters you find appropriate in hierarchical clustering by interpreting the dendrogram

End notes on clustering

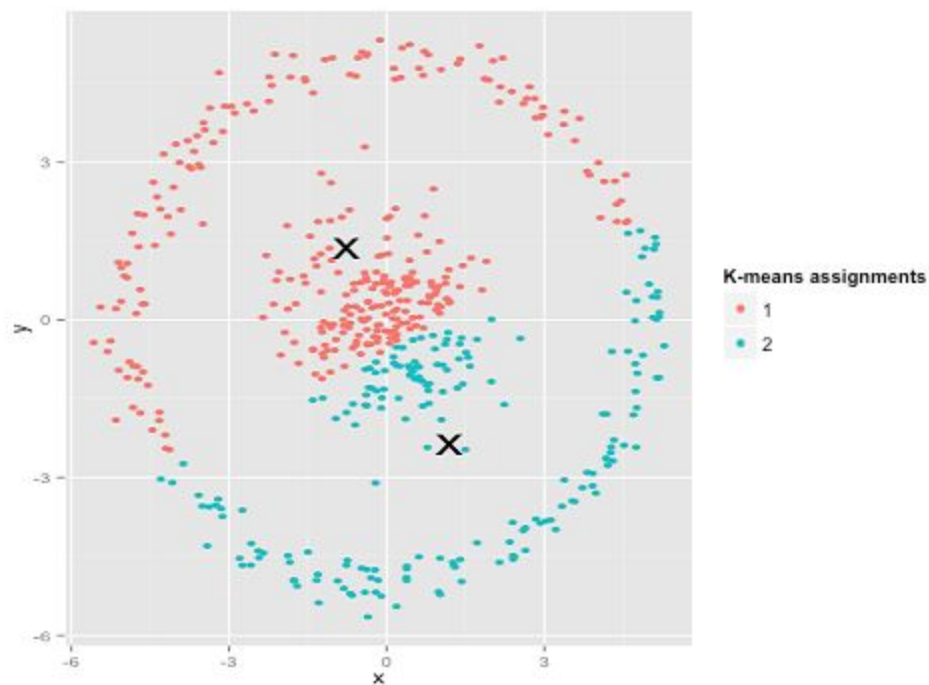
You have now learnt a fair deal about clustering algorithms, specifically about KMeans and Agglomerative clustering. Although both have their own set of advantages, it cannot be denied that they come with their own set of drawbacks. A perennial issue with cluster analysis is how seriously to treat the clusters we get. (This issue is of course linked to picking the best number of clusters.) At one extreme we can regard them as pure fictions, merely more-or-less convenient ways of summarizing some parts of the data, with no other meaning. At the other end we can insist that they reflect real divisions of the world into distinct types. Let us look at some situations where one algorithm fails while the other triumphs.

Spherical data

The human eye can clearly distinguish the data points into two clusters; one with the inner circle and the other with the outer circle. **How does KMeans perform?**

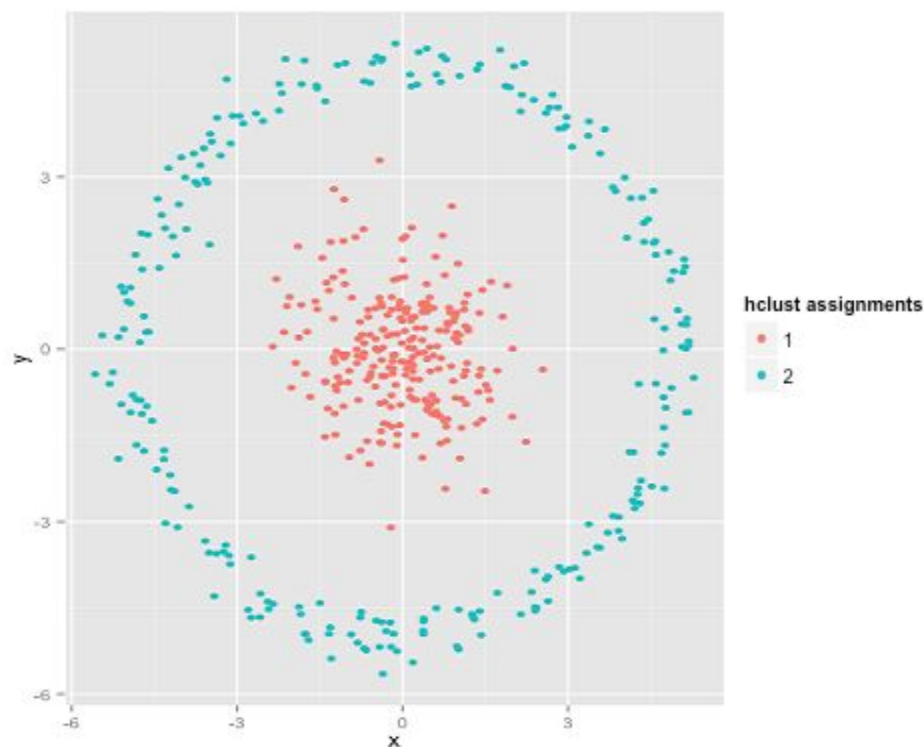


Well! KMeans does a pretty terrible job at partitioning into clusters as shown below:

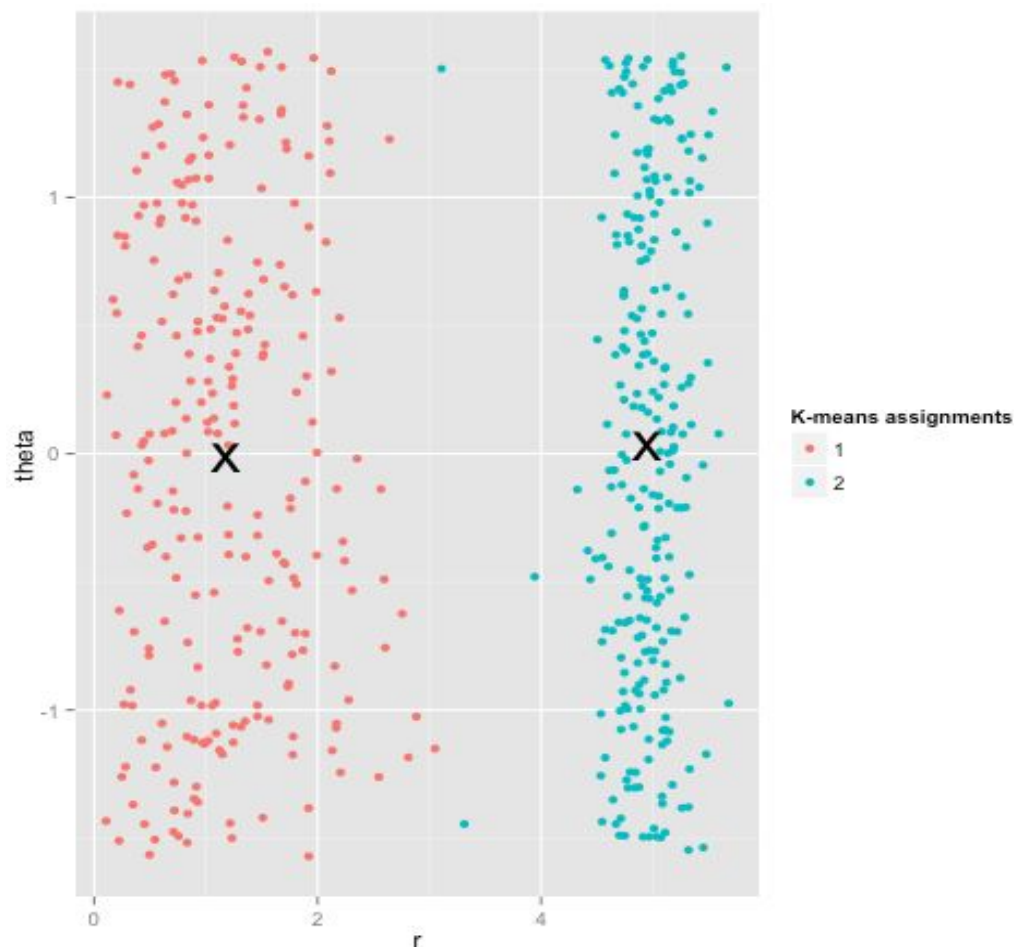


KMeans does nothing wrong here (algorithmically speaking); it is still minimizing the total sum-of-squared distance to the cluster centroids. This produces undesirable results when the clusters are elongated in certain directions - particularly when the between-cluster distance is smaller than the maximum within-cluster distance.

How about hierarchical clustering with single linkage? Well it does a pretty good job at it because it makes the right set of assumptions for this data. If you remember, hierarchical clustering points are clustered together based on their nearest neighbor, which facilitates clustering along *paths* in the dataset.

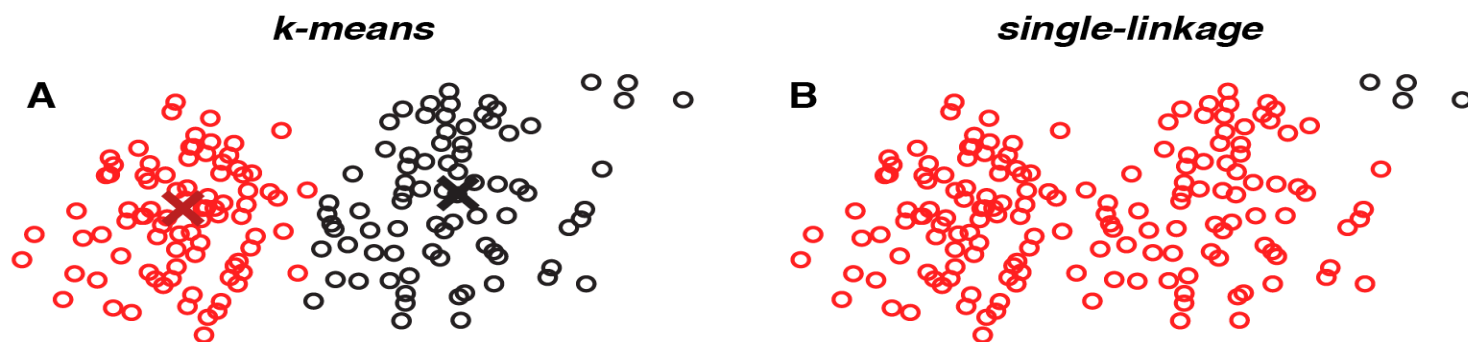


How about transforming data points into polar coordinates and then applying KMeans on top of it? Well, KMeans works perfectly this time.



Noisy data



How about noisy data? Which one between KMeans and Single linkage hierarchical will perform better? Single-linkage clustering is more sensitive to noise, because each clustering assignment is based on a single pair of datapoints (the pair with minimal distance). This can cause paths to form between overlapping clouds of points. In contrast, k-means uses a more global calculation – minimizing the distance to the nearest centroid summed over all points. As a result, k-means typically does a better job of identifying partially overlapping clusters.



The whole point of elaborating on these two specific sets of failures was to establish this point: Understanding the assumptions underlying a method is essential: it doesn't just tell you when a method has drawbacks, it tells you how to fix them

What are good clusters?

So how do we arrive at the conclusion that a cluster is good? Similar to any machine learning algorithm, a cluster is considered good which *generalizes well* i.e. *clusters should continue to describe new observations of the same features*.

1.	Which of the following is finally produced by Hierarchical Clustering ?		all of the Mentioned	tree showing how close things are to each other
<u>Explanation:</u> Hierarchical clustering is an agglomerative approach.				
2.	Hierarchical clustering should be primarily used for exploration.		True	True
<u>Explanation:</u> Hierarchical clustering is deterministic.				

Customer Segmentation

Problem Statement

Customer segmentation lies at the heart of every consumer facing business nowadays which involves being able to identify different types of customers and then figuring out ways to find more of those individuals so you can... you guessed it, get more customers!

In this assessment, you'll be doing exactly that, albeit with the help of K-Means clustering and observe how you can form groups of customers exhibiting similar behaviour.

About the dataset

You have a dataset consisting of two sheets, with names `OfferInformation` and `Transactions`. `OfferInformation` has 32 rows and 7 columns while `Transactions` has 324 rows and 3 columns. Below given are snapshots of both:

OfferInformation Sheet

	B	C	D	E	F	G
1	Campaign	Varietal	Minimum Qty (kg)	Discount (%)	Origin	Past Peak
2	January	Malbec	72	56	France	FALSE
3	January	Pinot Noir	72	17	France	FALSE
4	February	Espumante	144	32	Oregon	TRUE
5	February	Champagne	72	48	France	TRUE
6	February	Cabernet Sauvignon	144	44	New Zealand	TRUE
7	March	Prosecco	144	86	Chile	FALSE
8	March	Prosecco	6	40	Australia	TRUE
9	March	Espumante	6	45	South Africa	FALSE
10	April	Chardonnay	144	57	Chile	FALSE
11	April	Prosecco	72	52	California	FALSE
12	May	Champagne	72	85	France	FALSE
13	May	Prosecco	72	83	Australia	FALSE
14	May	Merlot	6	43	Chile	FALSE
15	June	Merlot	72	64	Chile	FALSE
16	June	Cabernet Sauvignon	144	19	Italy	FALSE
17	June	Merlot	72	88	California	FALSE
18	July	Pinot Noir	12	47	Germany	FALSE
19	July	Espumante	6	50	Oregon	FALSE
20	July	Champagne	12	66	Germany	FALSE
21	August	Cabernet Sauvignon	72	82	Italy	FALSE
22	August	Champagne	12	50	California	FALSE
23	August	Champagne	72	63	France	FALSE
24	September	Chardonnay	144	39	South Africa	FALSE
25	September	Pinot Noir	6	34	Italy	FALSE
26	October	Cabernet Sauvignon	72	59	Oregon	TRUE
27	October	Pinot Noir	144	83	Australia	FALSE
28	October	Champagne	72	88	New Zealand	FALSE
29	November	Cabernet Sauvignon	12	56	France	TRUE
30	November	Pinot Grigio	6	87	France	FALSE
31	December	Malbec	6	54	France	FALSE
32	December	Champagne	72	89	France	FALSE
33	December	Cabernet Sauvignon	72	45	Germany	TRUE

Feature description is given below:

Feature	Description
Offer #	Order ID
Campaign	Month of the campaign
Varietal	Grape variety
Minimum Qty(kg)	Minimum quantity ordered in kgs

Discount(%)	Discount on that order
Origin	Country where the variety of grape is grown
Past Peak	If wine flavor has faded

Transactions Sheet

	A	B
1	Customer Last Name	Offer #
2	Smith	2
3	Smith	24
4	Johnson	17
5	Johnson	24
6	Johnson	26
7	Williams	18
8	Williams	22
9	Williams	31
10	Brown	7
11	Brown	29
12	Brown	30
13	Jones	8
14	Miller	6
15	Miller	10
16	Miller	14
17	Miller	15
18	Miller	22
19	Miller	23
20	Miller	31
21	Davis	12
22	Davis	22
23	Davis	25
24	Garcia	14
25	Garcia	15
26	Rodriguez	2
27	Rodriguez	26
28	Wilson	8
29	Wilson	30
30	Martinez	12
31	Martinez	25
32	Martinez	28
33	Anderson	24
34	Anderson	26
35	Taylor	7
36	Taylor	18
37	Taylor	29
38	Taylor	30

Feature description is given below:

Feature	Description
Customer Last Name	Name of the customer purchasing
Offer #	Offer ID

Why solve this project

Successfully solving this project will help you apply the following skills:

- Data Manipulation
- Dimensionality Reduction Technique with PCA
- KMeans clustering
- Interpreting clusters with visualization

Combine into a single dataframe

There are two Excel sheets at your disposal. The first one is called `OfferInformation` which contains information on the offers extended on products. The other sheet is `Transactions` which contains information on customers' purchase records. Your task will be to load these two sheets as dataframes and then merge them according to a common column present in both sheets. This will enable you to visualize both the offers and transactions data in a single frame.



Instructions

- File path is given as `path`. Load dataframe for `Offers` as `.read_excel(path, sheet_name=0)` and save it as `offers`
- Load `Transactions` sheet as in the same way but with `sheet_name=1` and save it as `transactions`
- Add a new column `'n'` to `transactions` and set the value as 1 across all observations. This is done to identify customers who have purchased a particular offer
- Now merge `offers` and `transactions` into a single dataframe `df`
- Print out the first five rows of `df` using `.head()`

Skills Covered:

```
1 # import packages
2 import pandas as pd
3 import numpy as np
4 import seaborn as sns
5 import matplotlib.pyplot as plt
6 # Load Offers
7 offers = pd.read_excel(path,sheet_name=0)
8 print(offers.shape)
9 # Load Transactions
10 transactions = pd.read_excel(path,sheet_name=1)
11 print(transactions.shape)
12 # Merge dataframes
13 transactions['n'] = transactions['n'] = 1
14 df = pd.merge(transactions,offers)
15 # Look at the first 5 rows
16 print(df.head())
```

```
(32, 7)
(324, 2)
Customer Last Name Offer # n Campaign Varietal Minimum Qty (kg) \
0 Smith 2 1 January Pinot Noir 72
1 Rodriguez 2 1 January Pinot Noir 72
2 Martin 2 1 January Pinot Noir 72
3 Jackson 2 1 January Pinot Noir 72
4 Campbell 2 1 January Pinot Noir 72
Discount (%) Origin Past Peak
0 17 France False
1 17 France False
2 17 France False
3 17 France False
4 17 France False
```

Create an Offer-Transaction pivot table

Now that you have merged both `offers` and `transactions` sheets, time to group this data by `Offer #` for every customer. This can be done effectively with the help of a pivot table where every observation is indexed by the name of the customer (`'Customer Last Name'`), columns will represent the offer ID (`'Offer #'`) and the values denote whether the offer was purchased by the customer (`'n'`)



Instructions

- Create a pivot table `matrix` using `.pivot_table()` on `df` with arguments `index='Customer Last Name'`, `columns='Offer #'` and `values='n'`
- You will have a lot of missing values (since every customer did not purchase all the offers). Fill in the missing values by 0s. You can do it by `.fillna(0, inplace=True)`
- The last names of customers are on the index; translate it to a column using `.reset_index(inplace=True)` so that it is a column now
- Print out the first five rows of `matrix` now

Skills Covered:

```
1 # Code starts here
2
3 # create pivot table
4 matrix = df.pivot_table(index='Customer Last Name', columns='Offer #', values='n')
5 # replace missing values with 0
6 matrix.fillna(0,inplace=True)
7
8 # reindex pivot table
9 matrix.reset_index(inplace=True)
10
11 # display first 5 rows
12 print(matrix.head())
13
14 # Code ends here
```

```
Offer # Customer Last Name 1 2 3 4 5 6 7 8 9 ... \
0 Adams 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 ...
1 Allen 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 ...
2 Anderson 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 ...
3 Bailey 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 ...
4 Baker 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 ...
Offer # 23 24 25 26 27 28 29 30 31 32
0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 1.0 0.0 0.0
1 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0
2 0.0 1.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0
3 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0
4 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0
[5 rows x 33 columns]
```


Use Kmeans to cluster data

The management has made a decision to segment customers into 5 categories in order to analyse customer behaviour. Now that you have data in the proper format, time to apply KMeans clustering algorithm to form the required 5 clusters and look if any such meaningful clusters are formed.

Instructions

- Initialize a KMeans object `cluster` with `KMeans()` and arguments as `n_clusters=5, init='k-means++', max_iter=300, n_init=10` and `random_state=0`
- Create a new column `'cluster'` in the dataframe `matrix` where you store the cluster centers for every observation from `matrix`. These cluster centers can be obtained using `.fit_predict(matrix[matrix.columns[1:]])` method of `matrix`. The first column isn't required as it contains the customer names
- Print out first five rows using `.head()` method of `matrix`

Skills Covered:

Machine Learning

```
1 # import packages
2 from sklearn.cluster import KMeans
3
4 # Code starts here
5
6 # initialize KMeans object
7 cluster = KMeans(n_clusters=5, init='k-means++', max_iter=300, n_init=10, random_state=0)
8 # create 'cluster' column
9 matrix['cluster'] = cluster.fit_predict(matrix[matrix.columns[1:]])
10 print(matrix.head())
11
12 # Code ends here
```

> TRY IT

RESULT

Offer #	Customer Last Name	1	2	3	4	5	6	7	8	9	\
0	Adams	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
1	Allen	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	
2	Anderson	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
3	Bailey	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	
4	Baker	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	
Offer #	...	24	25	26	27	28	29	30	31	32	cluster
0	...	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0
1	...	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	2
2	...	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1
3	...	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0
4	...	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	2

[5 rows x 34 columns]

Visualize clusters using PCA

Now that you have clustered your data, it will be great if you can perform some sort of visualization for effective interpretation. But since the data at hand is multidimensional, you cannot simply do it on a 2-D graph. A way around is using PCA to reduce the dimensionality to two dimensions so that you can visualize the clusters

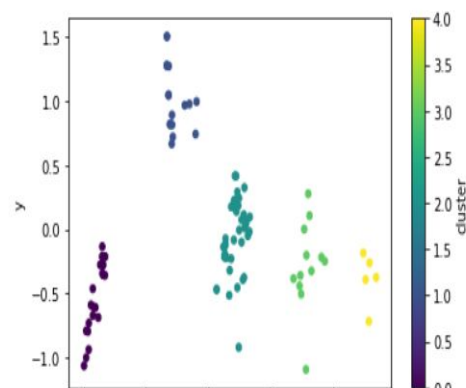
Instructions

- Initialize a PCA object `pca` using `PCA(n_components=2, random_state=0)`
- Create a new column `'x'` for `matrix` dataframe which denotes the X co-ordinates of every observation in decomposed form using `.fit_transform(matrix[matrix.columns[1:]][:,0])` method of `pca`
- Similarly create a new column `'y'` which denotes the decomposed Y-co-ordinates of every observation using `.fit_transform(matrix[matrix.columns[1:]][:,1])`
- Create a new dataframe `clusters` containing the column numbers `[0, 33, 34, 35]` i.e. customer names, cluster it belongs to and X and Y co-ordinates of every observation respectively. You can use `.iloc()` method to achieve this
- Visualize clusters using `.scatter(x='x', y='y', c='cluster', colormap='viridis')`. This will help you visualize data points according to clusters

```
1 # import packages
2 from sklearn.decomposition import PCA
3
4 # Code starts here
5 # initialize pca object with 2 components
6 pca = PCA(n_components=2, random_state=0)
7 # create 'x' and 'y' columns denoting observation locations in decomposed
8 matrix['x'] = pca.fit_transform(matrix[matrix.columns[1:]][:,0])
9 matrix['y'] = pca.fit_transform(matrix[matrix.columns[1:]][:,1])
10 # dataframe to visualize clusters by customer names
11 clusters = matrix.iloc[:,[0,33,34,35]]
12 print(clusters)
13 # visualize clusters
14 clusters.plot.scatter(x='x', y='y', c='cluster', colormap='viridis')
15 plt.show()
```

30	Wright	4	2.450114	-0.104744
99	Young	3	1.817034	-0.216389

[100 rows x 4 columns]



Task:

Which cluster orders the most `Champagne`?

In this task you will be performing a set of data manipulation to bring data into the required form and then answer this question regarding `Varietal` feature: Which cluster orders the most Champagne?

Instructions

- First merge dataframes `clusters` and `transactions` into a single dataframe using `.merge()` method of pandas. This will merge dataframes on `Customer Last Name` column and save it as `data`
- Again, use `.merge()` to merge `offers` and `data` and save it as `data` again
- Initialize an empty dictionary `champagne`. Here you will store cluster number (you have 5 clusters from the previous task) as keys and values as the number of champagne ordered (only if that cluster has maximum orders for type 'Champagne' in `Varietal` column)
- Iterate using a for loop over the cluster numbers. Since the clusters are numbered [0,1,2,3,4], you can iterate over `range(0,5)` or you can use `data['cluster'].unique()`
- Create a new dataframe `new_df` within this loop where you create dataframes for every cluster value (for example you will have a dataframe consisting of observations belonging to cluster number 0)
- Initialize a variable `counts` where you will sort the counts for every value of 'Varietal' column of `new_df` in a descending order using `.value_counts(ascending=False)`
- Now you will check if the very first entry of `counts` is 'Champagne'. Since variants of 'Varietal' are in the index of `counts`, use `.index[0]` and use an if condition to check if its value is 'Champagne'. If it is yes, append the cluster number (iterator in for loop) as key in `champagne` dictionary and its corresponding value `counts[0]` as the value of that key
- Once you have finished the for loop, it is time to find out the cluster which purchases the most 'Champagne'. You have the dictionary `champagne` and all you have to do is to return the cluster number (key of the dictionary) with the maximum value. Save the cluster number as `cluster_champagne`.
- Print out `cluster_champagne` to see which cluster purchases the most champagne!

Code:

```
# Code starts here

# merge 'clusters' and 'transactions'

data = pd.merge(clusters, transactions)

print(data.head())

print('='*25)

# merge `data` and `offers`

data = pd.merge(offers, data)

print(data.head())

print('='*25)
```

```
# initialzie empty dictionary

champagne = {}

# iterate over every cluster

for val in data.cluster.unique():

    # observation falls in that cluster

    new_df = data[data.cluster == val]

    # sort cluster according to type of 'Varietal'

    counts = new_df['Varietal'].value_counts(ascending=False)

    # check if 'Champagne' is ordered mostly

    if counts.index[0] == 'Champagne':

        # add it to 'champagne'

        champagne[val] = (counts[0])

# get cluster with maximum orders of 'Champagne'

cluster_champagne = max(champagne, key=champagne.get)

# print out cluster number

print(cluster_champagne)

#     print('='*50)
```

Which cluster of customers favours discounts more on an average?

In this task you will find the cluster which repsonds most to discounts so that the management can target them specifically for discounted pricing options.

Instructions

- Create an empty dictionary `discount` where keys are going to be cluster numbers and values will be average percentage of discount for the entire cluster
- Iterate over all the clusters using a for loop and create a dataframe `new_df` for every cluster within the loop
- Calculate the average percentage of discounts for `new_df` by first adding all the discount values and then dividing the total by total number of observations of `new_df`. Store it in `counts`
- Add the cluster number as key and its value `counts` as value to dictionary `discount`
- Now find out the cluster number with the maximum average discount percentage from `cluster_discount` and save it as `cluster_discount`

```
1 # Code starts here
2
3 # empty dictionary
4 discount = {}
5
6 # iterate over cluster numbers
7 for val in data.cluster.unique():
8     # dataframe for every cluster
9     new_df = data[data.cluster == val]
10    # average discount for cluster
11    counts = (new_df['Discount (%)'].values.sum()) / len(new_df)
12    # adding cluster number as key and average discount as value
13    discount[val] = counts
14 # cluster with maximum average discount
15 cluster_discount = max(discount, key=discount.get)
16 print(cluster_discount)
17 # Code ends here
```

OUTPUT

RESULT

4

