DATA SOCIETY®

Clustering - day 1

"One should look for what is and not what he thinks should be."
-Albert Einstein.

Module completion checklist

Objective	Complete
Define what makes k-means an unsupervised method of machine learning	
Prepare the dataset to be clustered using k-means	
Run and visualize k-means with k=2	
Find the optimal number of k using elbow method and visualize	
Inspect the results of clustering the dataset using the optimal k	
Discuss common pitfalls of clustering	
Summarize hierarchical clustering and its two types	
Understanding agglomerative clustering and types of linkage methods	
Implement hierarchical clustering on the data and visualize its dendrogram	

Directory settings

- In order to maximize the efficiency of your workflow, you should encode your directory structure into `variables`
- Let the `main_dir` be the variable corresponding to your `af-werx` folder

```
# Set `main_dir` to the location of your `af-werx` folder (for Linux).
main_dir = "/home/[username]/desktop/af-werx"
```

```
# Set `main_dir` to the location of your `af-werx` folder (for Mac).
main_dir = '/Users/[username]/desktop/af-werx'
```

```
# Set `main_dir` to the location of your `af-werx` folder (for Windows).
main_dir = "C:\\Users\\[username]\\desktop\\af-werx"
```

```
# Make `data_dir` from the `main_dir` and
# remainder of the path to data directory.
data_dir = main_dir + "/data"
```

Loading packages

- Load the packages we will be using

```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn.cluster import KMeans
from sklearn.preprocessing import MinMaxScaler
from scipy.cluster.vq import kmeans
from scipy.spatial.distance import cdist,pdist
from matplotlib import cm
from scipy import cluster
from scipy.spatial.distance import pdist
from scipy.cluster.hierarchy import dendrogram, linkage
from sklearn.preprocessing import MinMaxScaler
from sklearn.cluster import AgglomerativeClustering
```

Working directory

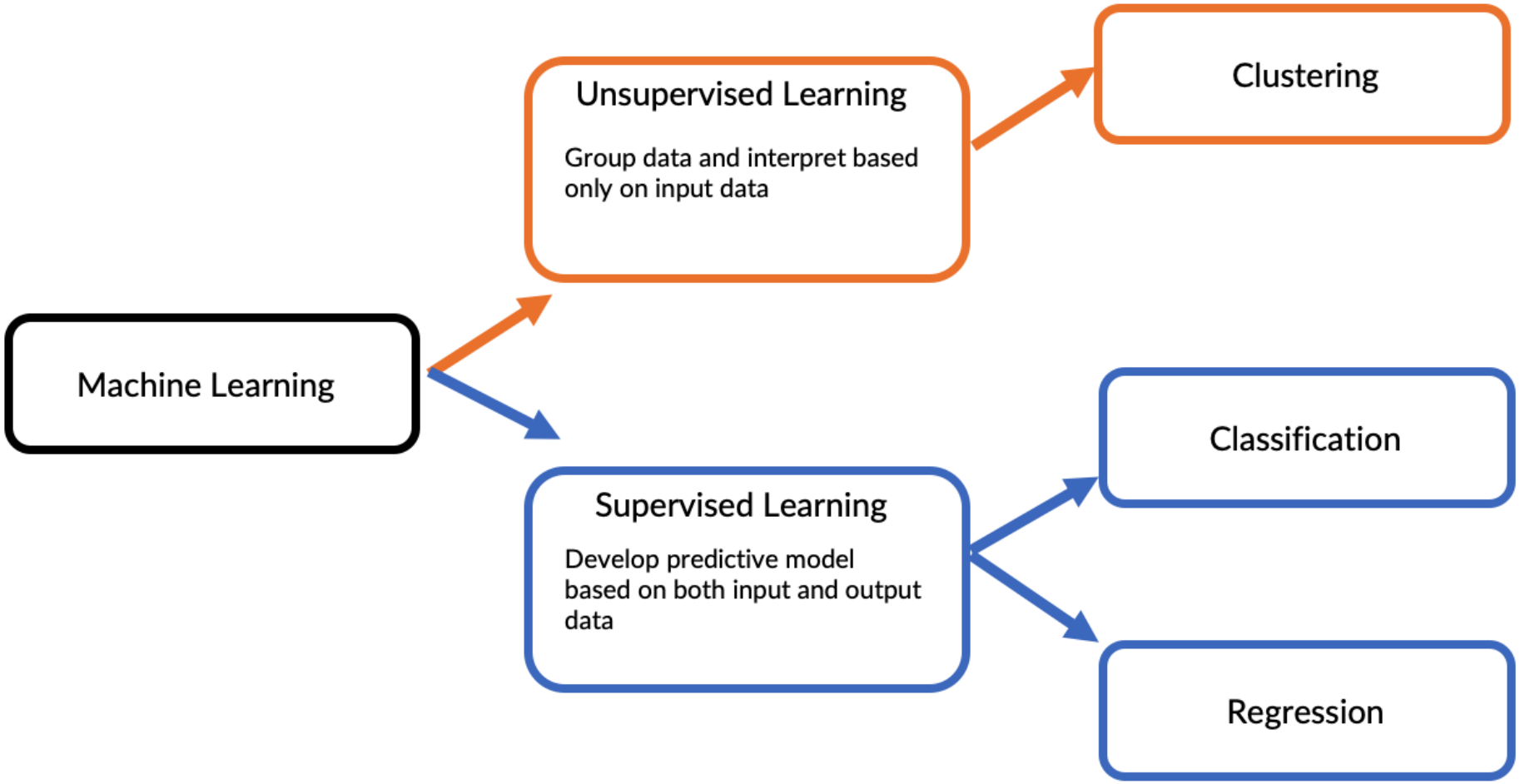
- Set working directory to `data_dir`

```
# Set working directory.
os.chdir(data_dir)
```

```
# Check working directory.
print(os.getcwd())
```

```
/home/[user-name]/Desktop/af-werx/data
```

Supervised vs. unsupervised learning



What is clustering?

1. Technique for **finding similarity** between groups
2. Type of **unsupervised machine learning** (it is one of its many methods)
3. There are many algorithms for clustering, but **similarity** needs to be defined for each one
    i. It depends on **attributes** of data
    ii. Usually measured by a *distance metric*
    iii. The way the *distance metrics* are used and the way the algorithm works **differs** based on the **clustering method**

Examples of clustering

- **Marketing**: Help marketers discover unique groups in their customer bases, providing knowledge to develop targeted marketing programs
- **Land use**: Allows for discerning areas of similar land use in a land observation database - Insurance: Assists in identifying groups of car insurance policyholders with a high average claim cost
- **City-planning**: Locating groups of houses according to their house type, value, and geographical location

- You can ask questions like **"What patterns are in this data?"** and **"Which points are similar to each other?"**

What is k-means?

- k-means is a clustering algorithm that **allows us to look for patterns** within our data
- This is useful when we have a lot of data on a subject matter, but it is **unstructured** and **not labeled**
- k-means most commonly uses **Euclidean distance**
- k-means is often used on **larger** datasets because it has minimal calculations of distance compared to other methods

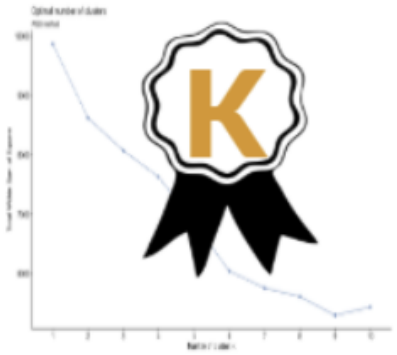
Steps of k-means clustering

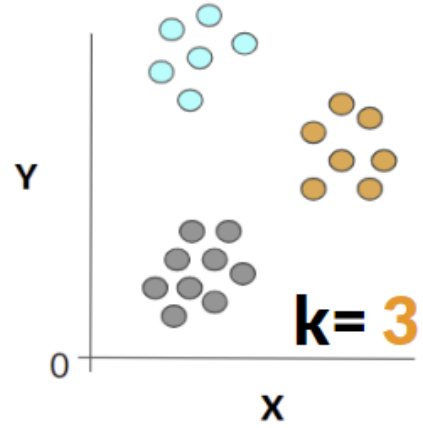
Step 1:
Find optimal k

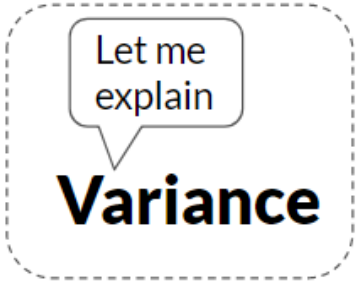
Step 2:
Use optimal k to cluster

Step 3:
Caculate metrics

Step 4:
Inspect clusters





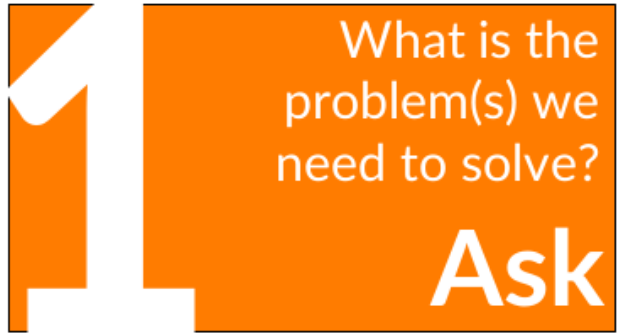


Module completion checklist

Objective	Complete
Define what makes k-means an unsupervised method of machine learning	✔
Prepare the dataset to be clustered using k-means	
Run and visualize k-means with k=2	
Find the optimal number of k using elbow method and visualize	
Inspect the results of clustering the dataset using the optimal k	
Discuss common pitfalls of clustering	
Summarize hierarchical clustering and its two types	
Understanding agglomerative clustering and types of linkage methods	
Implement hierarchical clustering on the data and visualize its dendrogram	

Business case: confirm subgroups

There is a new medicine out that has been questioned regarding the effects it has on heart rate; the medication will increase heart rate and the researchers are not certain if this will vary by gender



We have been given a sample of participants who took the medication for a study. We have:

- **Gender**
- **Heart rate**
- **Temperature**

We want to inspect the data and identify whether the patterns we see in data (i.e. the increase in heart rate) matches the division of participants according to their gender

This is a **data mining** & **clustering** task, rather than a *modeling* task

K-means: two variable dataset

We will now walk through k-means using a sample dataset with only two variables:

1. **Body temperature**
2. **Heart rate**

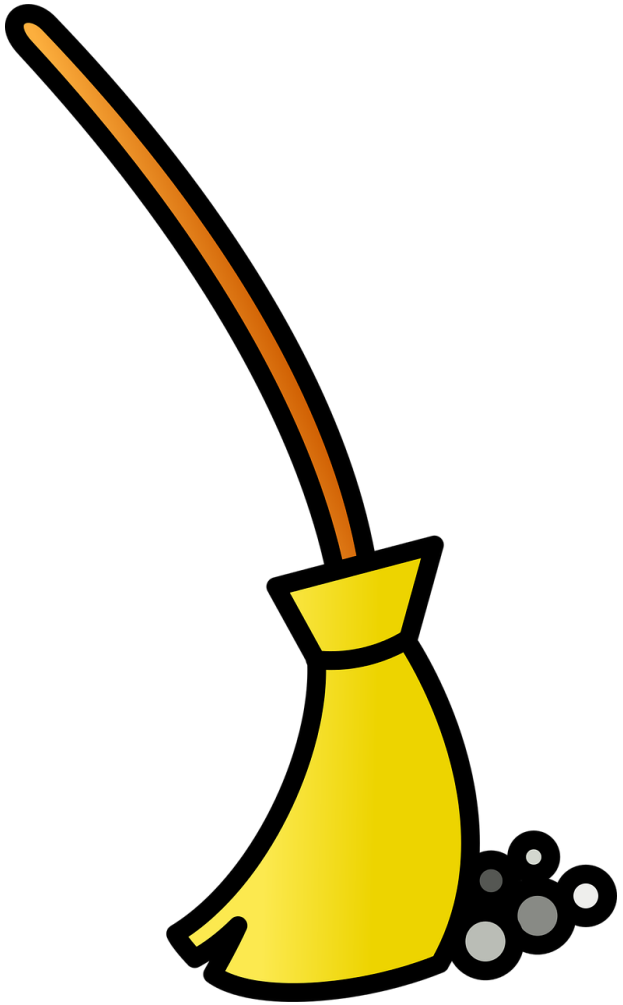
The original dataset has labels (i.e. `M` and `F`), but we are going to illustrate k-means and finding patterns by using the **temperature** and **rate** variables on their own to group data points without using those labels

K-means: data prep

Before we use any algorithm, step 0 is to inspect, prepare, and clean the data. For unsupervised clustering, we need to:

- Prepare:
  - Remove the labels so that we can use unsupervised learning
- Clean:
  - Address **NAs**, missing values
  - Confirm or convert variable types to numeric
  - Scale data
  - Address outliers (*This is outside the scope of this course, and is not an issue with this dataset*)

Depending on your data, all these items can come into play



Data prep

- We will now load the dataset and save it as `temp_heart`

```
temp_heart = pd.read_csv("temp_heart_rate.csv")
```

```
print(temp_heart.head())
```

```
  Gender  Body Temp  Heart Rate
0   Male        96.3          70
1   Male        96.7          71
2   Male        96.9          74
3   Male        97.0          80
4   Male        97.1          73
```

Data cleaning: NAs

- First, we check how many NAs there are in each column

```
print(temp_heart.isnull().sum())
```

```
Gender         0
Body Temp      0
Heart Rate     0
dtype: int64
```

Subset data

- We will subset `Body Temp` and `Heart Rate` to `temp_heart_cluster`

```
# Subset even further to just have 'Body Temp' and 'Heart Rate'.
temp_heart_cluster = temp_heart[['Body Temp','Heart Rate']]
print(temp_heart_cluster.head())
```

```
   Body Temp  Heart Rate
0       96.3          70
1       96.7          71
2       96.9          74
3       97.0          80
4       97.1          73
```

Numeric variables

- In k-means clustering, we use **numeric data**
- In some cases, we can **convert categorical data to integer values**
- However, here, our data is numeric by default
- Let's check if our variables are numeric, and if not, separate out all numeric variables
- If there were columns that were not numeric, they would not be in the `numeric_data` dataframe
- We see that all the columns are numeric

```
# Check data type of our variables.
print(temp_heart_cluster.dtypes)
```

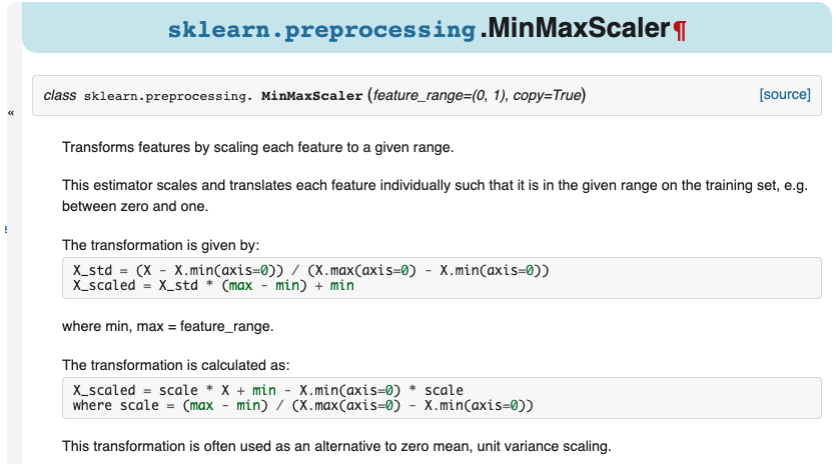
```
Body Temp     float64
Heart Rate      int64
dtype: object
```

Why do we scale our data?

- In k-means, the math behind the algorithm involves **calculating distance**
- Distance calculations are **very sensitive to scale**
- If we do not scale our variables to all be on the same numeric scale, our **distances will be relative to numbers** instead of to each other

MinMaxScaler

- Once the data is converted to `numeric` (if necessary), we **scale** the dataset
- There are a few methods to scale data; we will use the `MinMaxScaler` function, which is from `sklearn`



sklearn.preprocessing.MinMaxScaler¶

class sklearn.preprocessing. **MinMaxScaler** (*feature_range=(0, 1), copy=True*) [source]

Transforms features by scaling each feature to a given range.

This estimator scales and translates each feature individually such that it is in the given range on the training set, e.g. between zero and one.

The transformation is given by:

```
X_std = (X - X.min(axis=0)) / (X.max(axis=0) - X.min(axis=0))
X_scaled = X_std * (max - min) + min
```

where min, max = feature_range.

The transformation is calculated as:

```
X_scaled = scale * X + min - X.min(axis=0) * scale
where scale = (max - min) / (X.max(axis=0) - X.min(axis=0))
```

This transformation is often used as an alternative to zero mean, unit variance scaling.

K-means data prep using MinMaxScaler

- Let's instantiate the scaler and transform our `temp_heart_cluster` dataset
- We create a new object `temp_heart_cluster_scaled`

```
# Instantiate MinMaxScaler.
scaler = MinMaxScaler()

# Scale the dataframe.
temp_heart_cluster_scaled = scaler.fit_transform(temp_heart_cluster)
```

```
# Convert back to dataframe, making sure to name the columns again.
temp_heart_kmeans = pd.DataFrame(temp_heart_cluster_scaled, columns = temp_heart_cluster.columns)
print(temp_heart_kmeans.head())
```

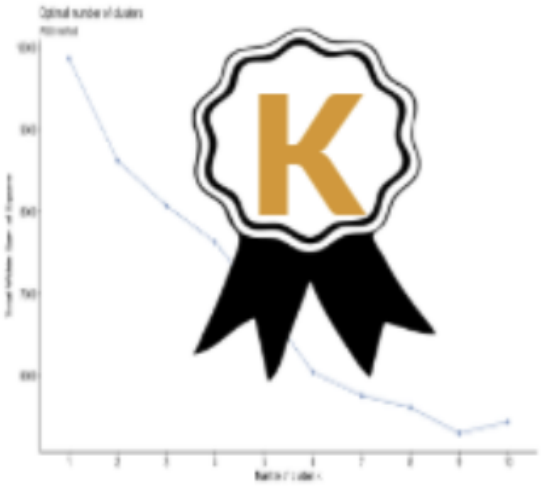
```
   Body Temp  Heart Rate
0   0.000000     0.40625
1   0.088889     0.43750
2   0.133333     0.53125
3   0.155556     0.71875
4   0.177778     0.50000
```

Module completion checklist

Objective	Complete
Define what makes k-means an unsupervised method of machine learning	✔
Prepare the dataset to be clustered using k-means	✔
Run and visualize k-means with k=2	
Find the optimal number of k using elbow method and visualize	
Inspect the results of clustering the dataset using the optimal k	
Discuss common pitfalls of clustering	
Summarize hierarchical clustering and its two types	
Understanding agglomerative clustering and types of linkage methods	
Implement hierarchical clustering on the data and visualize its dendrogram	

Find the best k

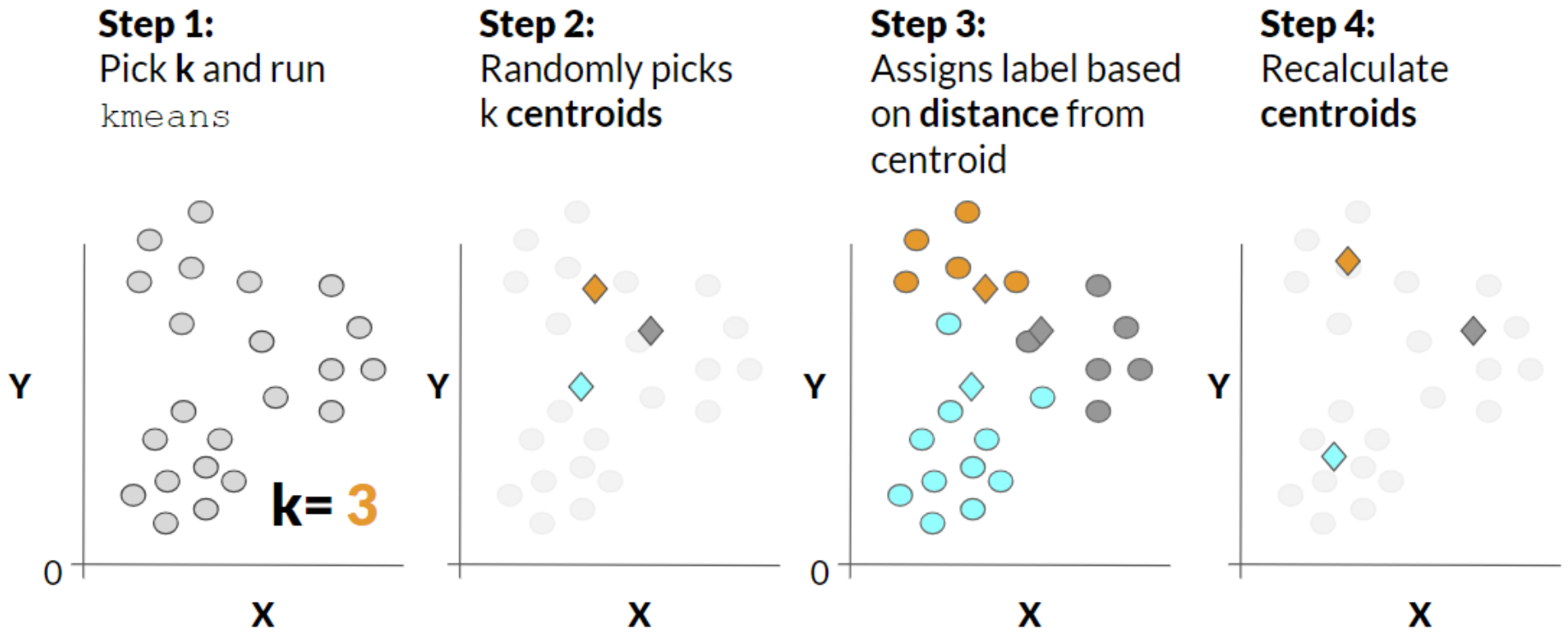
- First, we need to **run the k-means algorithm** on the dataset for `k = 1` through `k = 10`
- We will start with k = 2 since we hope to find more than one cluster in this dataset



Implementing k-means

- We will be using two different packages today: `Kmeans` from `sklearn.cluster` and `kmeans` from `scipy.cluster.vq`
- They both allow you to perform **k-means clustering** on a dataframe object
- `sklearn.cluster` is less robust as far as the output
- Because we are learning about `kmeans`, we will also use `scipy.cluster.vq`

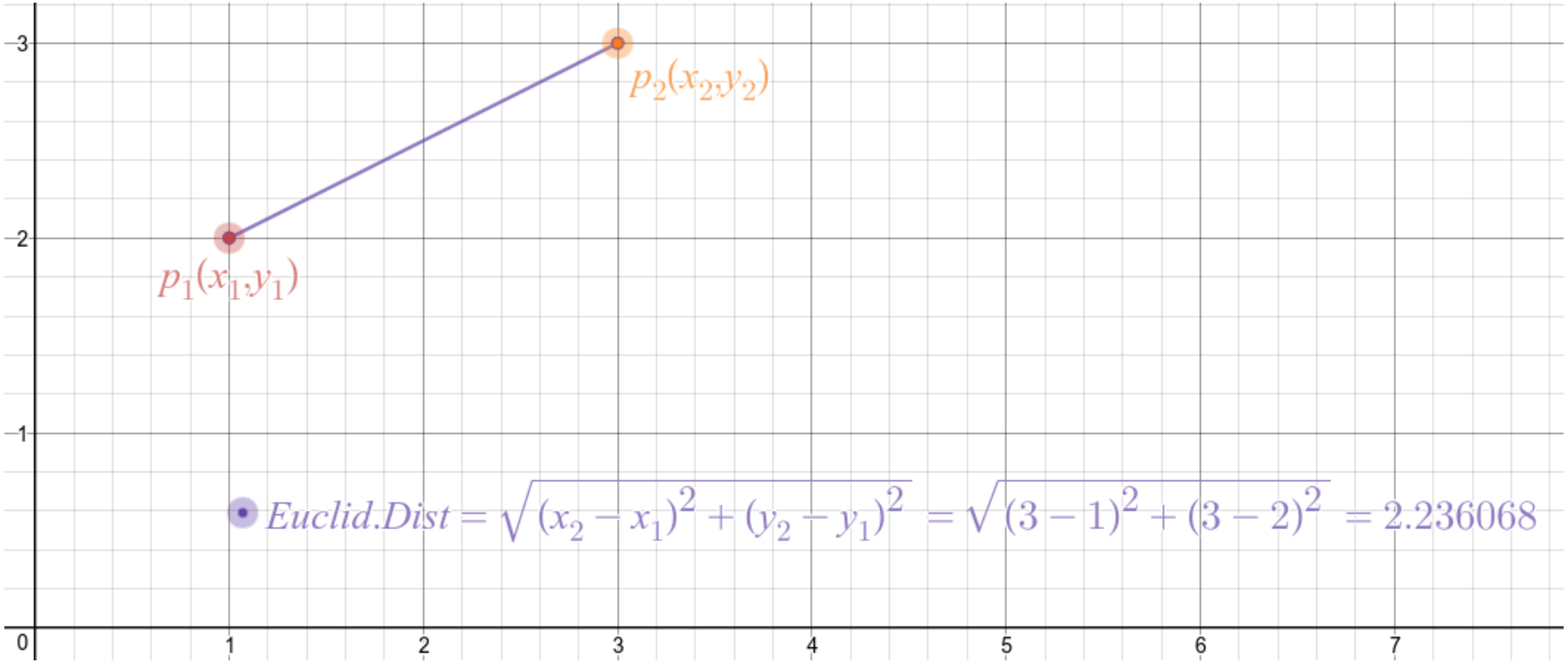
K-means algorithm



- **Repeats steps 3 and 4** again until **no more reassignments** are made, and **optimal** clusters are formed

K-means: uses Euclidean distance

- **Euclidean distance**: distance metric most commonly used with k-means

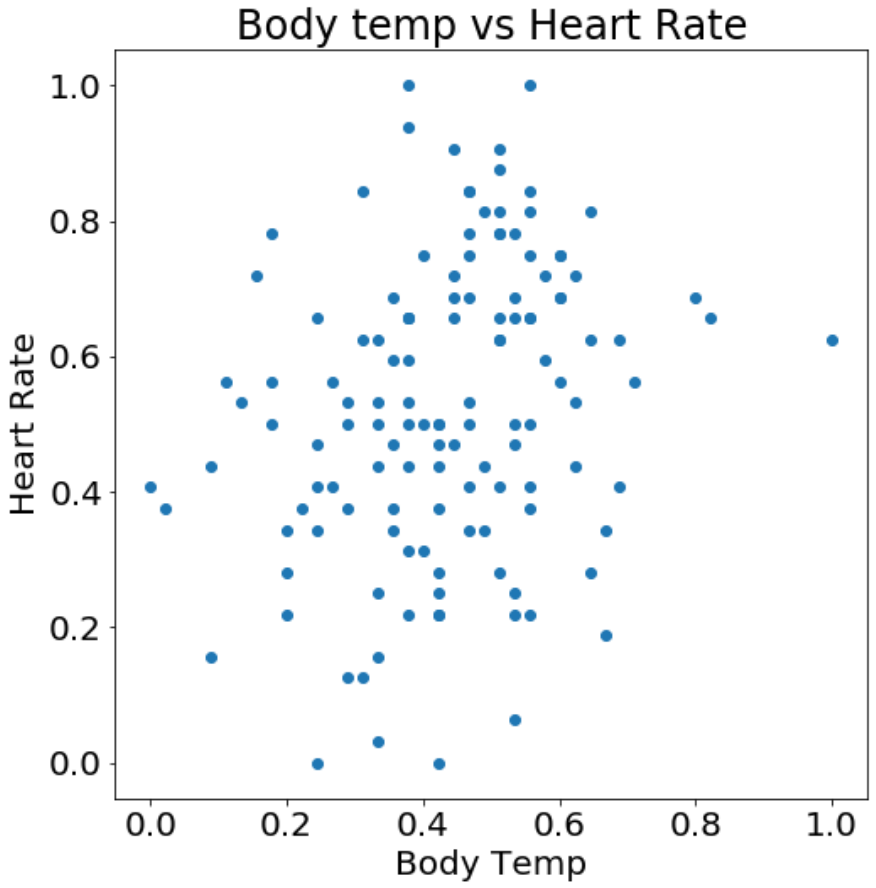


K-means: Body Temp vs Heart Rate

- Let's plot these two variables just to see what the interaction looks like
- **What are your thoughts?**

```
# Plot the data.
plt.scatter(temp_heart_kmeans['Body Temp'],
            temp_heart_kmeans['Heart Rate'])

plt.title('Body temp vs Heart Rate')
plt.ylabel('Heart Rate')
plt.xlabel('Body Temp')
```



K-means: k-means with k=2

- Let's first start with an arbitrary `k = 2`
- `sklearn.cluster` allows us to plot our clusters easily, so we are using it now

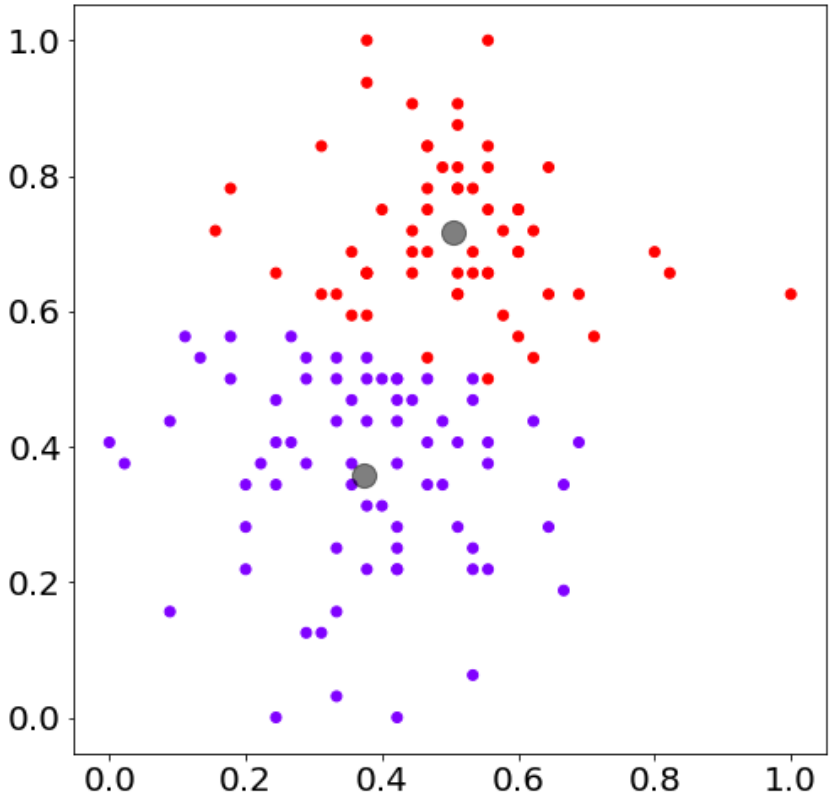
```
# K-means - start with 2 clusters.
# Initializing k-means.
kmeans_2 = KMeans(n_clusters=2)
# Fitting with inputs.
kmeans_2 = kmeans_2.fit(temp_heart_kmeans)
# Predicting the clusters.
labels = kmeans_2.predict(temp_heart_kmeans)
# Getting the cluster centers.
C_2 = kmeans_2.cluster_centers_
print(C_2)
```

```
[[0.37206349 0.35848214]
 [0.50444444 0.71666667]]
```

K-means: plot k=2

- Now let's plot our data with the clusters colored in, with each centroid plotted

```
# First, we plot our clusters, colored in by the
labels.
plt.scatter(temp_heart_kmeans.iloc[:,0],
            temp_heart_kmeans.iloc[:,1],
            c=kmeans_2.labels_,
            cmap='rainbow')
# Second, we plot the optimized centroids over
the clusters.
plt.scatter(C_2[:, 0],
            C_2[:, 1],
            c='black',
            s=200,
            alpha=0.5)
```



Knowledge check 1

Exercise 1

Module completion checklist

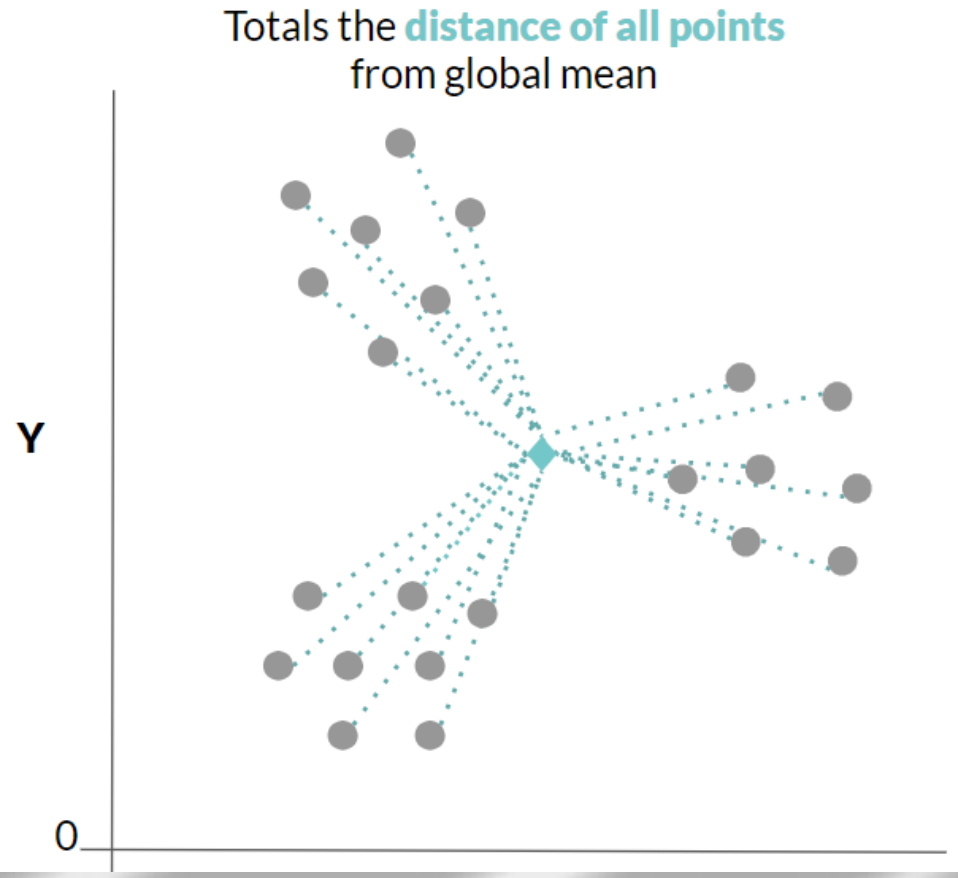
Objective	Complete
Define what makes k-means an unsupervised method of machine learning	✔
Prepare the dataset to be clustered using k-means	✔
Run and visualize k-means with k=2	✔
Find the optimal number of k using elbow method and visualize	
Inspect the results of clustering the dataset using the optimal k	
Discuss common pitfalls of clustering	
Summarize hierarchical clustering and its two types	
Understanding agglomerative clustering and types of linkage methods	
Implement hierarchical clustering on the data and visualize its dendrogram	

Evaluating your clusters

- There are many ways to evaluate your clusters, and they will be **determined by subject matter** mostly
- The evaluation metrics we will discuss today are:
  - **Between sum of squares (`BSS`)**: population variance based on the variance **between** the sample means - larger is better
  - **Within sum of squares (`WSS`)**: population variance based on the variance **within** each of the samples - smaller is better
  - **Total sum of squares (`TSS`)**: total distance of data points from global mean of data
  - *Explained variance*: the percentage of variance in the data explained by `k` clusters, this is equal to `BSS` / `TSS`

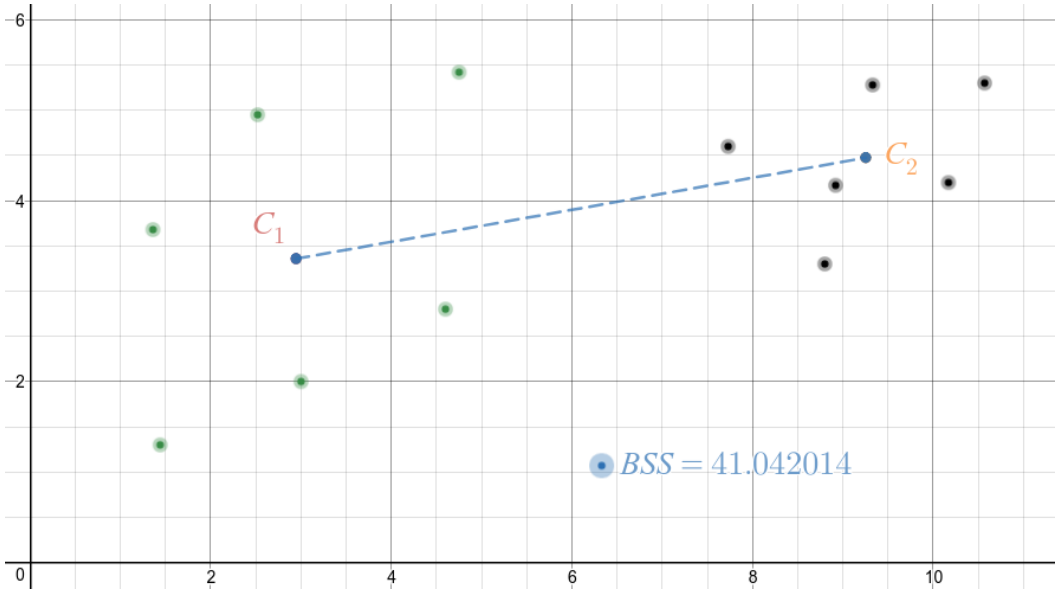
Total sum of squares

- **Total sum of squares (TSS)**: the total variance in the data measured by the sum of squares of the distance between **all** points and the **global center** of the data
- This is used as the denominator for explained variance



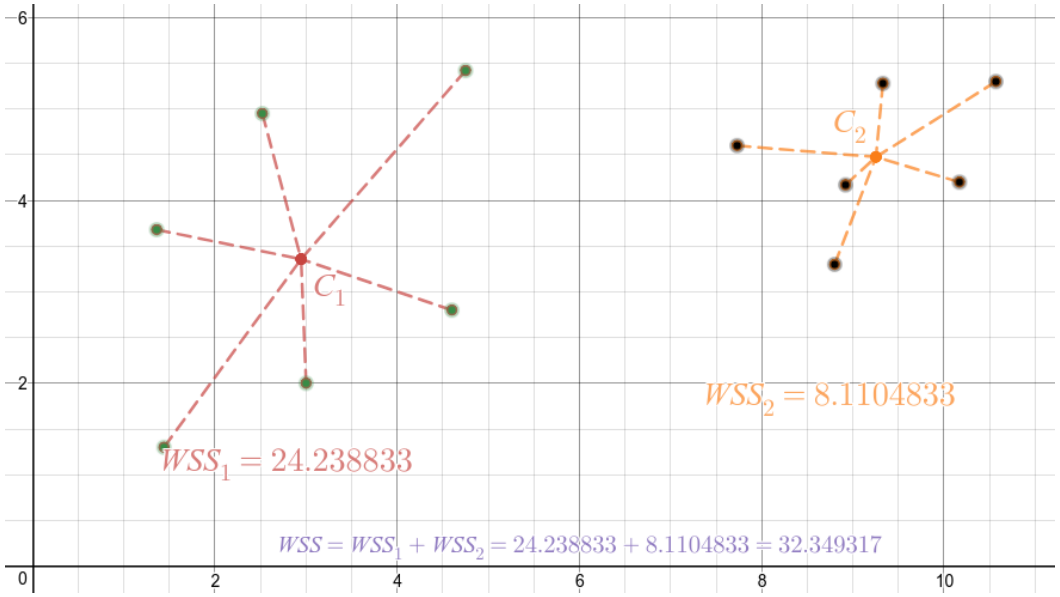
BSS & WSS

- **Between sum of squares (BSS)** or **inter-cluster distance**: the sum of squares of the distances between points and the centroid of other clusters
- BSS is a measure of group **separation**



We want to **MAXIMIZE** the BSS

- **Within sum of squares (WSS)** or **intra-cluster distance**: the sum of squares of the distances between points and the centroid, all within a cluster
- WSS is a measure of group **cohesion**



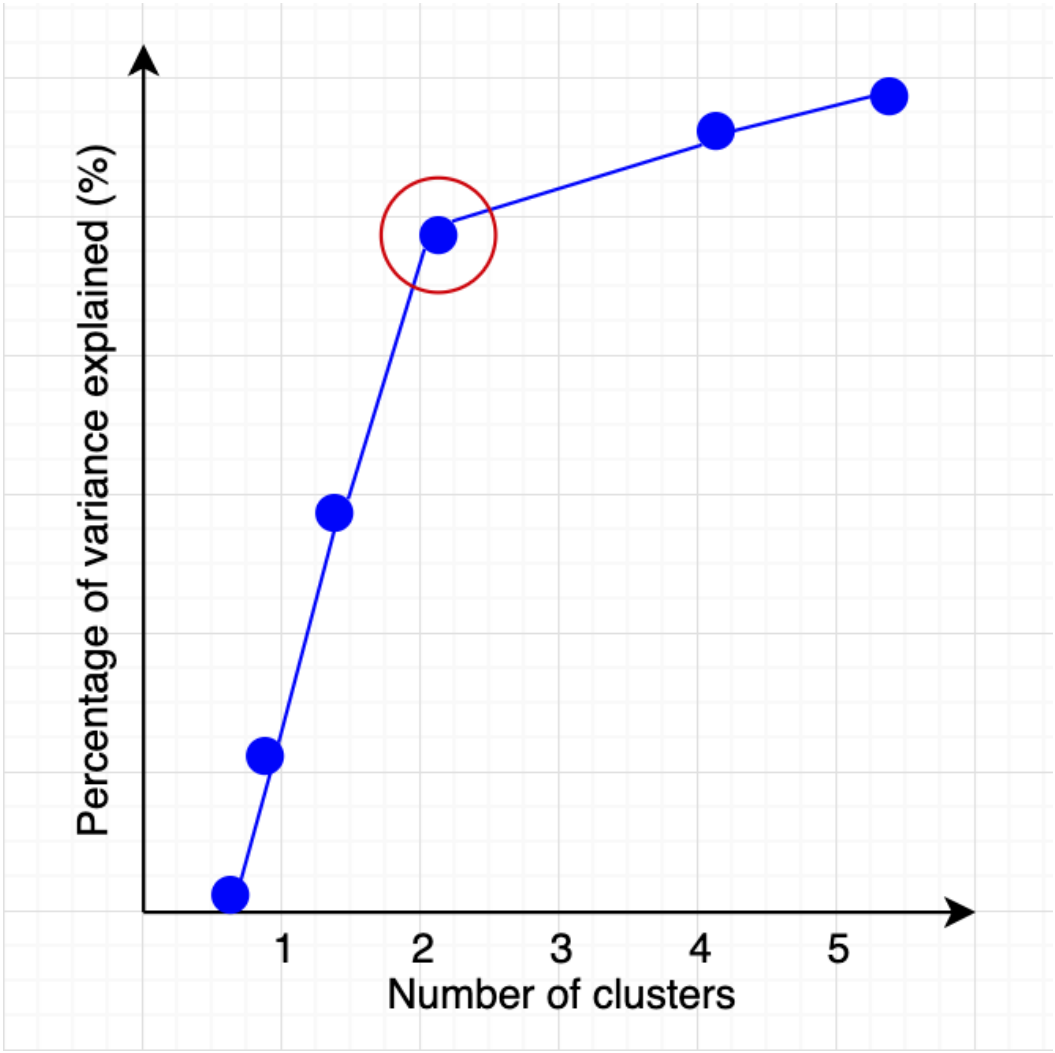
We want to **MINIMIZE** the WSS

Evaluate explained variance

- `Explained variance = BSS/TSS`
- K-means is a good fit for your data if enough variance is explained
- "Enough" being the key subjective word here
- **Depending on subject matter and the actual data, that will mean different things for each case**
- **Next, we will use `scipy.cluster.vq` to:**
  - Find the **optimal k** two different ways
  - Calculate these metrics for `k = 2` as well as the optimal number of clusters

Elbow method

- The method we use will plot explained variance to visualize different `k`s and choose the optimal number
- This is known as the `WSS` or **elbow method**
- For the **elbow method**, we choose `k` based on the inflection point (at the "elbow") so it maximizes the explained variance
- If there is a very close **tie** within the elbow plot, **use both** `k`s and then use **explained variance** to determine the best one



Elbow method

- We will now use `scipy.cluster.vq` to get the metrics we need for building our elbow plot

```
# Set the range of k.
K_MAX = 20
KK = range(1,K_MAX+1)

# Run `kmeans` for values in the range k = 1-20.
KM = [kmeans(temp_heart_kmeans,k) for k in KK]

# Find the centroids for each KM output.
centroids = [cent for (cent,var) in KM]

# Calculate centroids for each iteration of k.
D_k = [cdist(temp_heart_kmeans, cent, 'euclidean') for cent in centroids]
cIdx = [np.argmin(D,axis=1) for D in D_k]
dist = [np.min(D,axis=1) for D in D_k]

tot_withinss = [sum(d**2) for d in dist]                    # Total within-cluster sum of squares
totss = sum(pdist(temp_heart_kmeans)**2)/temp_heart_kmeans.shape[0]   # The total sum of squares
betweenss = totss - tot_withinss                            # The between-cluster sum of squares
```

Building our elbow plot

- Now we take the metrics we have just calculated and build our elbow plot
- We will be using **explained variance** as our measure in this plot
- The larger the explained variance, the better
- **Remember, we do not want to overfit!**
- We choose optimal `k` when the increase in explained variance starts to flatten out

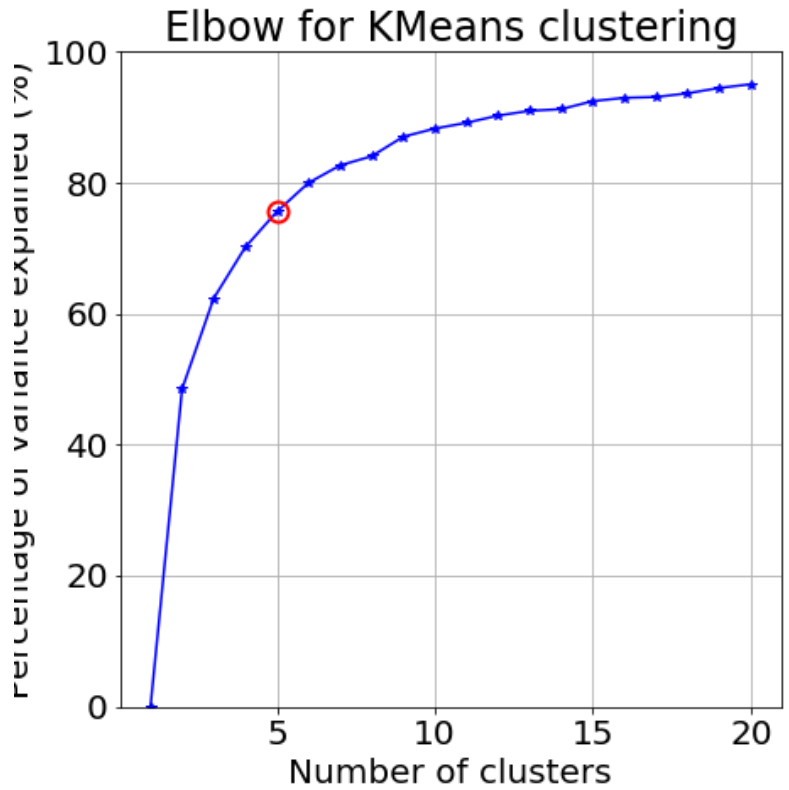
Elbow plot method

```
# Set range for k.
kIdx = 4          # K=5
clr = cm.Spectral( np.linspace(0,1,10)
).tolist()
mrk = 'os^p<dvh8>+x.'

# Elbow curve - explained variance.
fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(KK, betweenss/totss*100, 'b*-')
ax.plot(KK[kIdx], betweenss[kIdx]/totss*100,
marker='o', markersize=12,
    markeredgewidth=2, markeredgecolor='r',
markerfacecolor='None')
ax.set_ylim((0,100))
plt.grid(True)
plt.xlabel('Number of clusters')
plt.ylabel('Percentage of variance explained
(%)')
plt.title('Elbow for KMeans clustering')
```

- Looks like our optimal explained variance is when `k = 5`

```
(0, 100)
```



Baseline vs. optimal k

- Just for comparison, let's look at the **explained variance** for both `k = 2` and `k = 5`

```
# Explained variance for optimal number of clusters at `k = 2`.
print(betweenss[1]/totss * 100)
```

```
48.51255790975613
```

```
# Explained variance for optimal number of clusters at `k = 5`.
print(betweenss[4]/totss * 100)
```

```
75.61029297260023
```

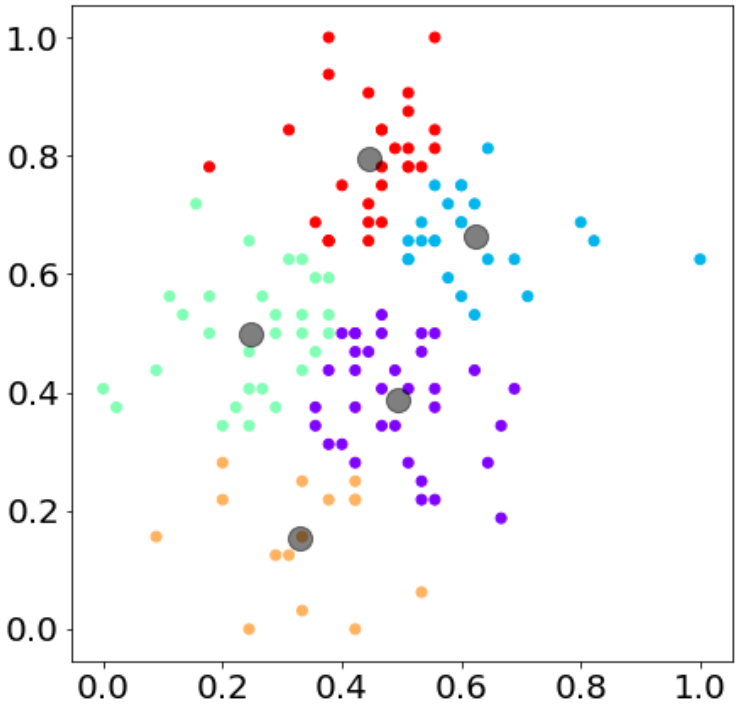
Run optimal k

- Now, let's look at what `k = 5` looks like
- We are going to run this in `sklearn` and plot
- For comparison, we will then show both `k = 2` and `k = 5` next to each other
- First, let's run the algorithm and store the results

```
# Initializing K-means.
kmeans_5 = KMeans(n_clusters = 5)
# Fitting with inputs.
kmeans_5 = kmeans_5.fit(temp_heart_kmeans)
# Predicting the clusters.
labels = kmeans_5.predict(temp_heart_kmeans)
# Getting the cluster centers.
C_5 = kmeans_5.cluster_centers_
```

Plot k = 5

```
# First we plot our clusters, colored in by the
labels.
plt.scatter(temp_heart_kmeans.iloc[:,0],
            temp_heart_kmeans.iloc[:,1],
            c = kmeans_5.labels_,
            cmap = 'rainbow')
# Second, we plot the optimized centroids over
the clusters.
plt.scatter(C_5[:, 0],
            C_5[:, 1],
            c = 'black',
            s = 200,
            alpha = 0.5)
```



Plot k = 2 vs. k = 5

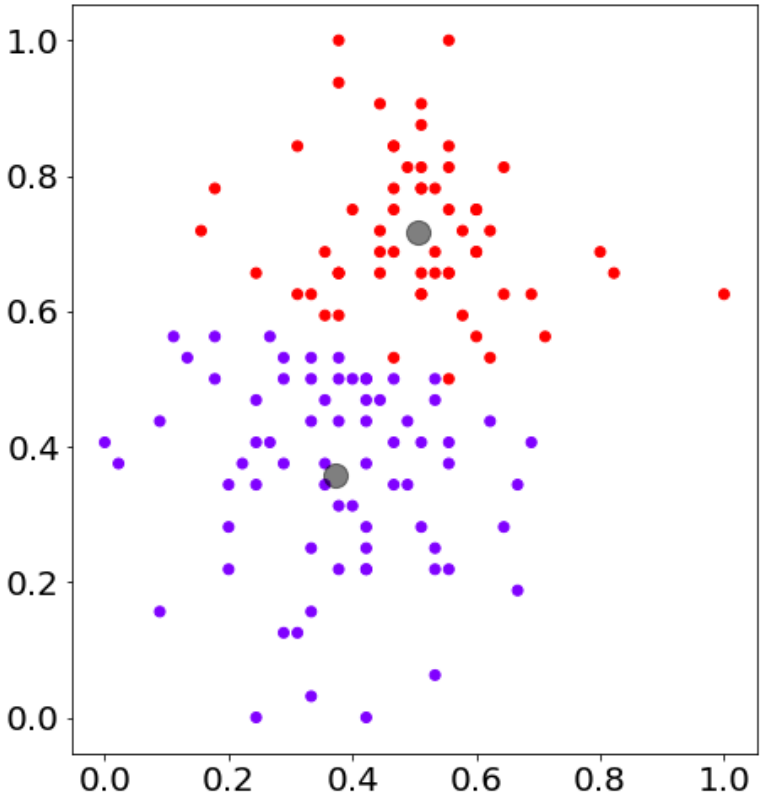
- Let's use `.subplot()` function to plot scatterplots of `k = 2` and `k = 5` together

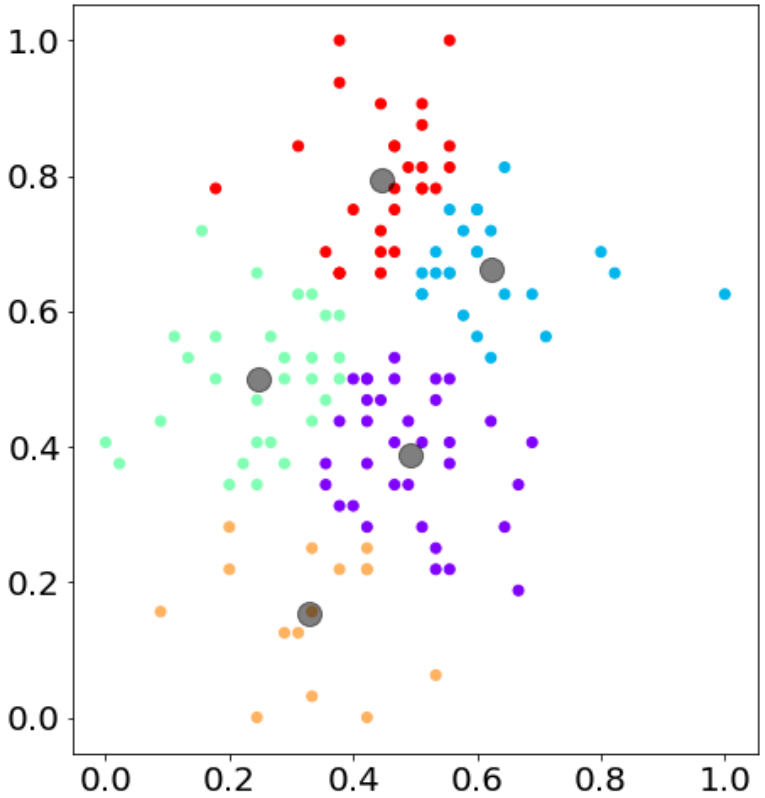
```
plt.clf()
plt.rcParams.update({'font.size': 20})
plt.figure(figsize = (16, 8))

plt.subplot(1, 2, 1)
plt.scatter(temp_heart_kmeans.iloc[:,0],temp_heart_kmeans.iloc[:,1],c=kmeans_2.labels_,cmap='rainbow')
plt.scatter(C_2[:, 0], C_2[:, 1], c='black', s=200,alpha=0.5)

plt.subplot(1, 2, 2)
plt.scatter(temp_heart_kmeans.iloc[:,0],temp_heart_kmeans.iloc[:,1],
c=kmeans_5.labels_, cmap='rainbow')
plt.scatter(C_5[:, 0], C_5[:, 1], c='black', s=200, alpha=0.5)
```

Plot k = 2 vs. k = 5





Inspect clusters

- We should always inspect our clusters when we workwith `kmeans`
- We do this to understand our newly found groups better
- We may want to bring back other variables from our dataset too
- Let's join our cluster labels back to our original subset `temp_heart_cluster`
- We could also join back our entire dataset, but for now we are just looking at a few columns

```
# Append the other variables back to the dataframe with clusters.
clustered_temp_heart = temp_heart_cluster
# Add cluster numbers.
clustered_temp_heart['clusters'] = pd.Series(labels)
```

```
clustered_temp_heart.head()
```

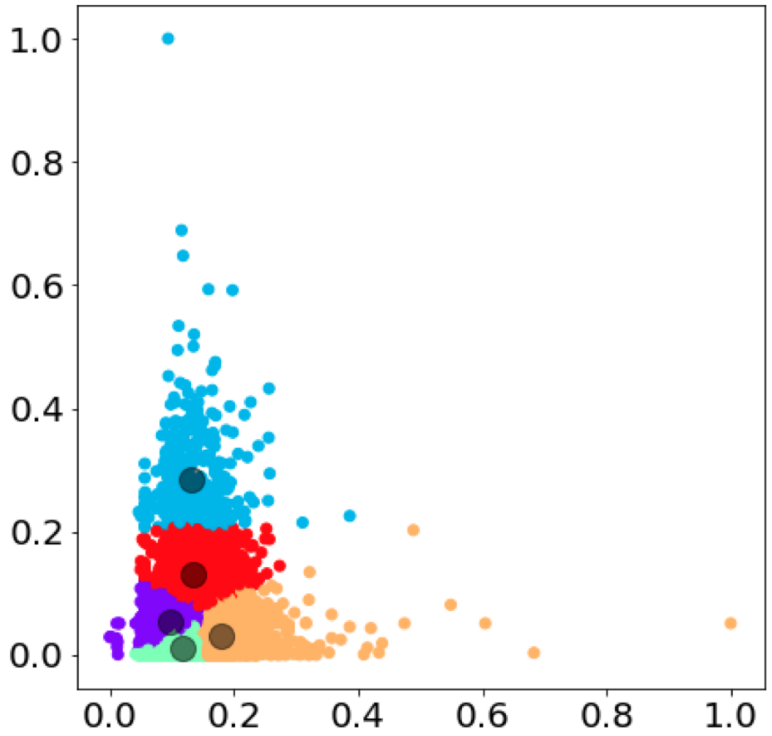
```
   Body Temp  Heart Rate  clusters
0       96.3          70         2
1       96.7          71         2
2       96.9          74         2
3       97.0          80         2
4       97.1          73         2
```

Inspect clusters

- **What are some observations you can make?**

```
# Group by `clusters` column to see the group mean of each
variable.
cluster_groups_means =
clustered_temp_heart.groupby('clusters').mean()
print(cluster_groups_means)
```

```
            Body Temp  Heart Rate
clusters
0           98.517647   69.382353
1           99.108333   78.208333
2           97.410345   72.965517
3           97.780000   61.933333
4           98.307143   82.428571
```



Inspect clusters

- Through these newfound groups, you could:
  - target audiences thoughtfully
  - build new variables to be used in predictive models
  - gain insight about your data
  - find patterns you did not expect in your data
- **What else could you do with these groups? How would this apply to the work you're doing?**

K-means: the good, bad, and the evil

The good and bad

- +: **CHEAP**: NO LABELS, labels are expensive to create and maintain
- +/−: clustering **always works**, but not always **well**

The evil

- Curse of **dimensionality**
- Clusters may result from **poor data quality**
- k-means does not get better with more data
- **Unequal cluster size** may result in poor **clustering**

Knowledge check 2

Exercise 2

Module completion checklist

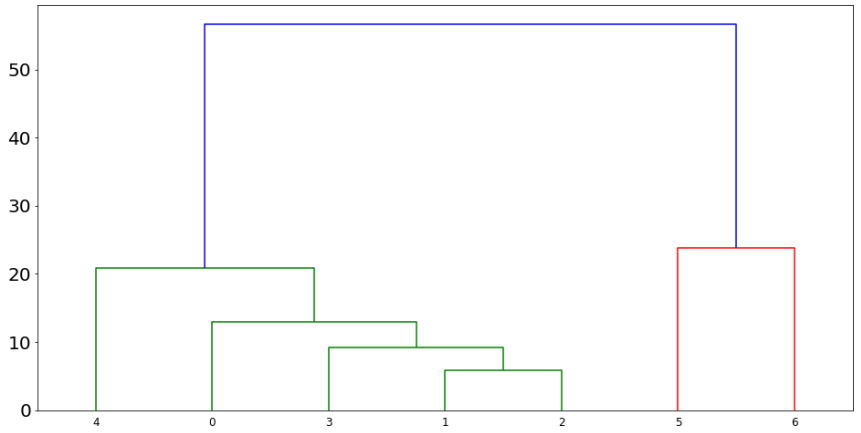
Objective	Complete
Define what makes k-means an unsupervised method of machine learning	✔
Prepare the dataset to be clustered using k-means	✔
Run and visualize k-means with k=2	✔
Find the optimal number of k using elbow method and visualize	✔
Inspect the results of clustering the dataset using the optimal k	✔
Discuss common pitfalls of clustering	✔
Summarize hierarchical clustering and its two types	
Understanding agglomerative clustering and types of linkage methods	
Implement hierarchical clustering on the data and visualize its dendrogram	

Hierarchical clustering

- Hierarchical clustering (or hierarchic clustering) produces a **hierarchy** of the data **without any prior knowledge about the initial structure** of the data
- It is a **slow technique**, since the algorithm does an exhaustive search through the data and then outputs similar data points close to each other in a **tree-like structure**

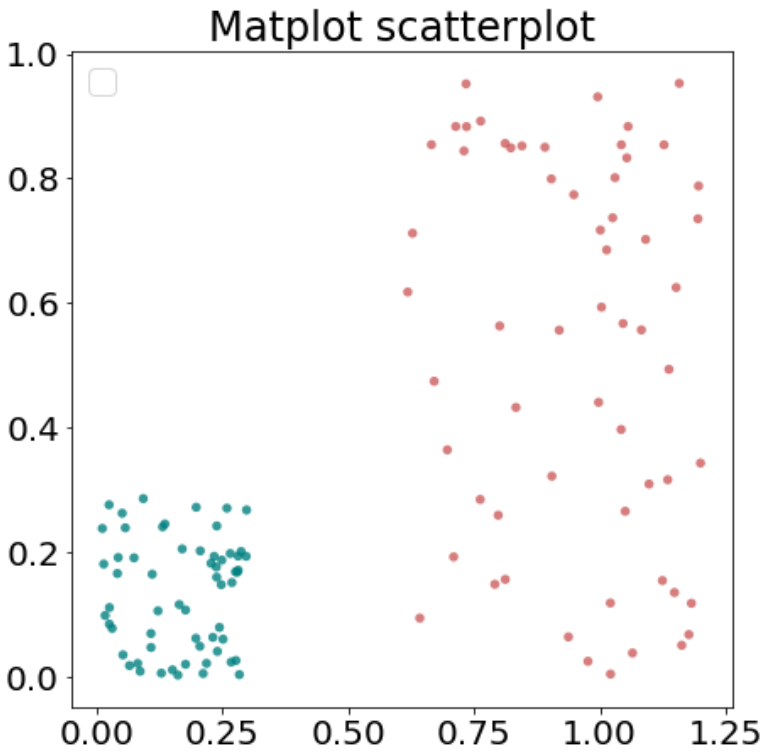
Why use hierarchical clustering?

- K-means suffers from **dimensionality issues**, hence it works better with **larger datasets**
- On the other hand, hierarchical clustering works very well with **smaller datasets**
- It also allows us to **avoid choosing the number of clusters beforehand**



Types of hierarchical clustering

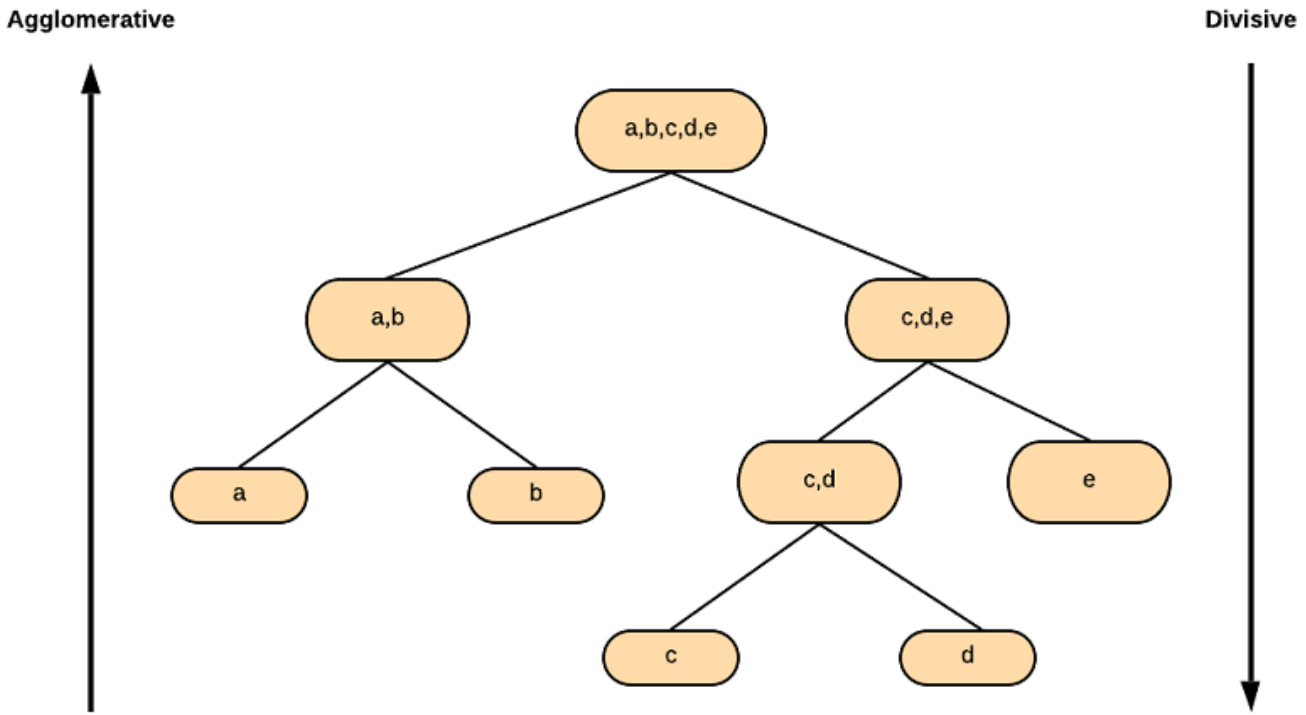
- We will be working with **agglomerative clustering algorithms (AGNES)**, which will use different **similarity measures** between the data points
- The other type of hierarchical clustering that we will not be covering in this course is **Divisive (DIANA)**



Types of hierarchical clustering

Hierarchical clustering can be divided into **two parts**:

- **Agglomerative clustering**, where all observations are initially considered as clusters of their own (**leaf**)
- **Divisive clustering**, where all observations are initially included in one cluster (**root**)



Module completion checklist

Objective	Complete
Define what makes k-means an unsupervised method of machine learning	✔
Prepare the dataset to be clustered using k-means	✔
Run and visualize k-means with k=2	✔
Find the optimal number of k using elbow method and visualize	✔
Inspect the results of clustering the dataset using the optimal k	✔
Discuss common pitfalls of clustering	✔
Summarize hierarchical clustering and its two types	✔
Understanding agglomerative clustering and types of linkage methods	
Implement hierarchical clustering on the data and visualize its dendrogram	

Agglomerative clustering

- **Agglomerative clustering** is a very common type of hierarchical clustering used to group objects in clusters based on similarities.
- The main idea behind an agglomerative clustering technique is that **one observation is one cluster**
- So we start out with **as many clusters as observations**

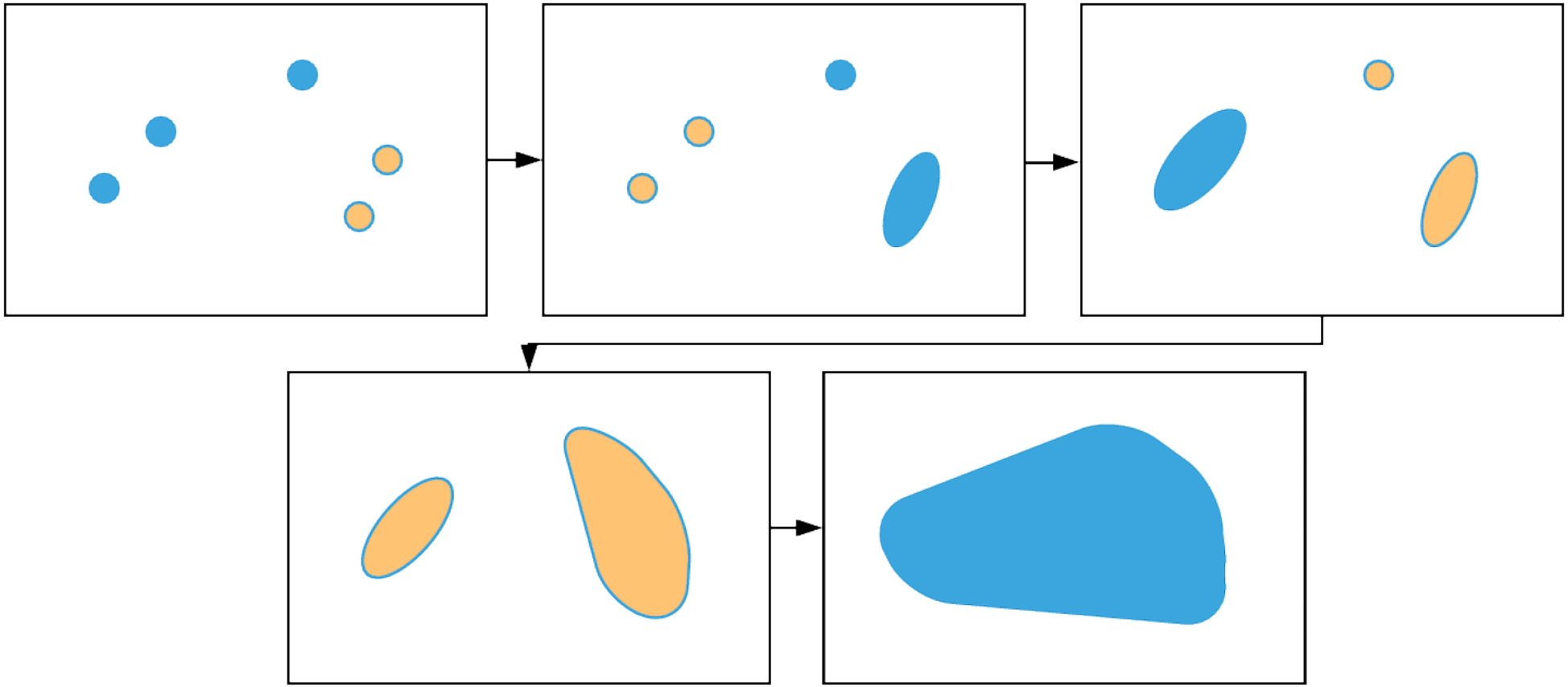
Process

We then follow these steps:

1. Compute the **similarity** (e.g. distance) between each of the clusters and **join the two most similar clusters**
2. The two similar clusters become one observation, everything else stays the same
3. **Repeat** steps 1 & 2 until there is only **one single cluster left**

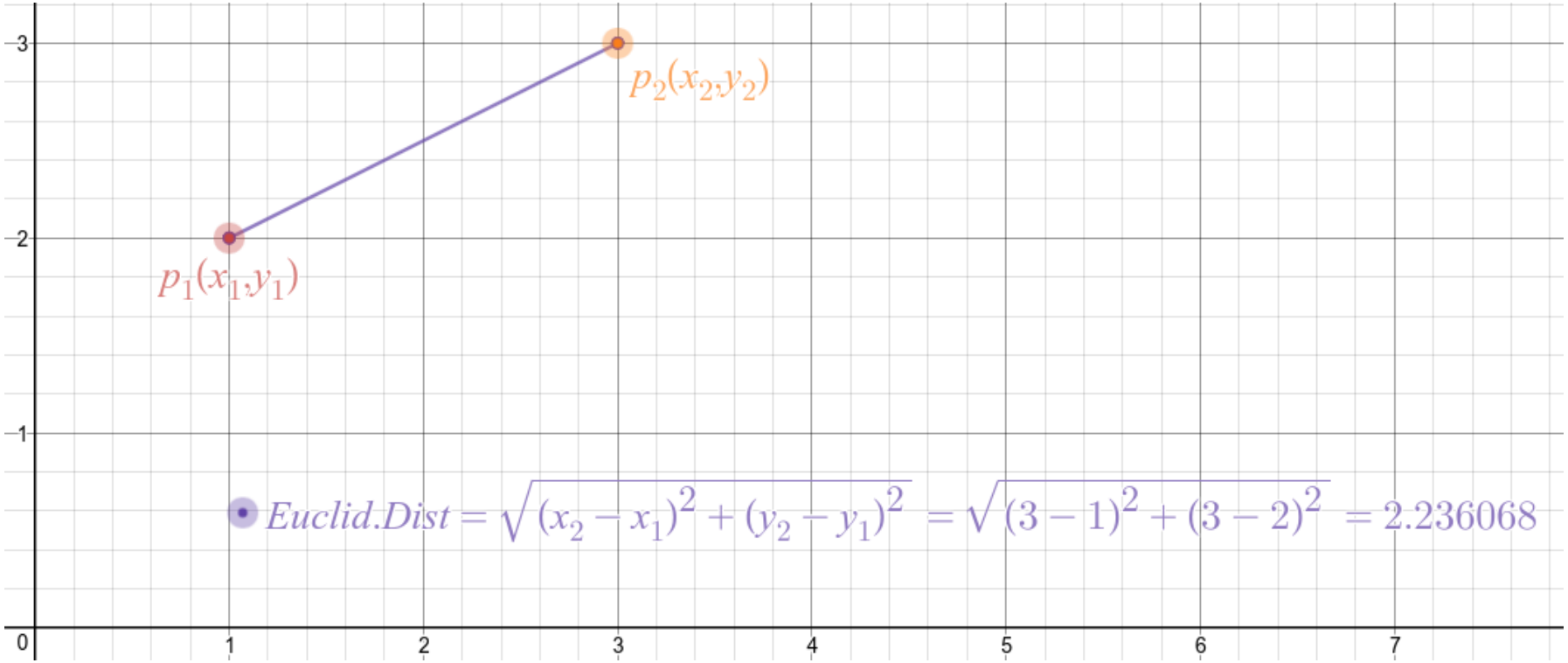
Process

- We can see that all observations eventually **combine into a single cluster**



Uses Euclidean distance

- **Euclidean distance**: distance metric most commonly used with hierarchical clustering

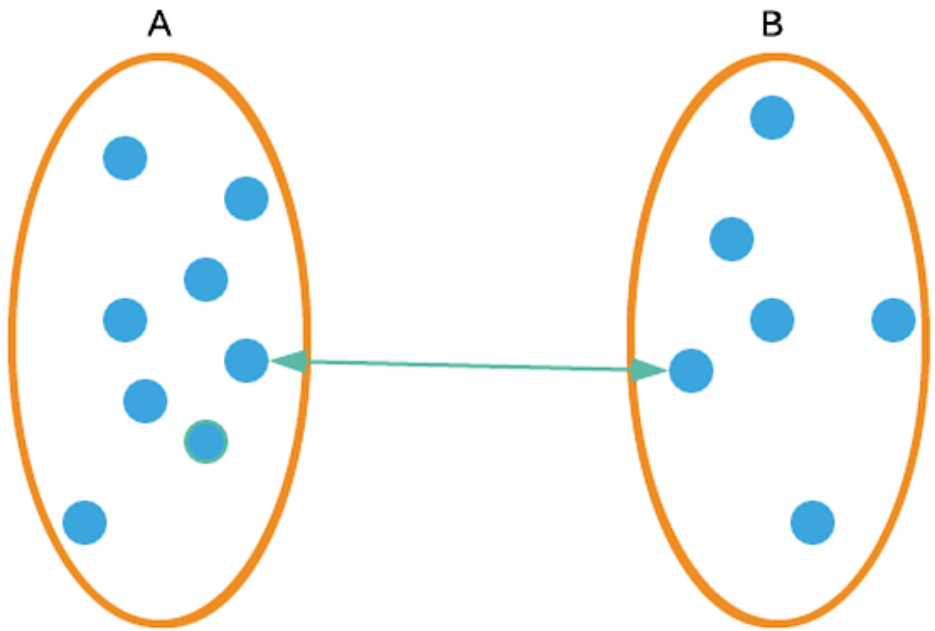


Methodology

- There are multiple agglomeration methods we can use to measure distance between clusters in agglomerative clustering
- They are:
  - **Single linkage**
  - **Complete linkage**
  - **Average linkage**
- We will walk through each of them now

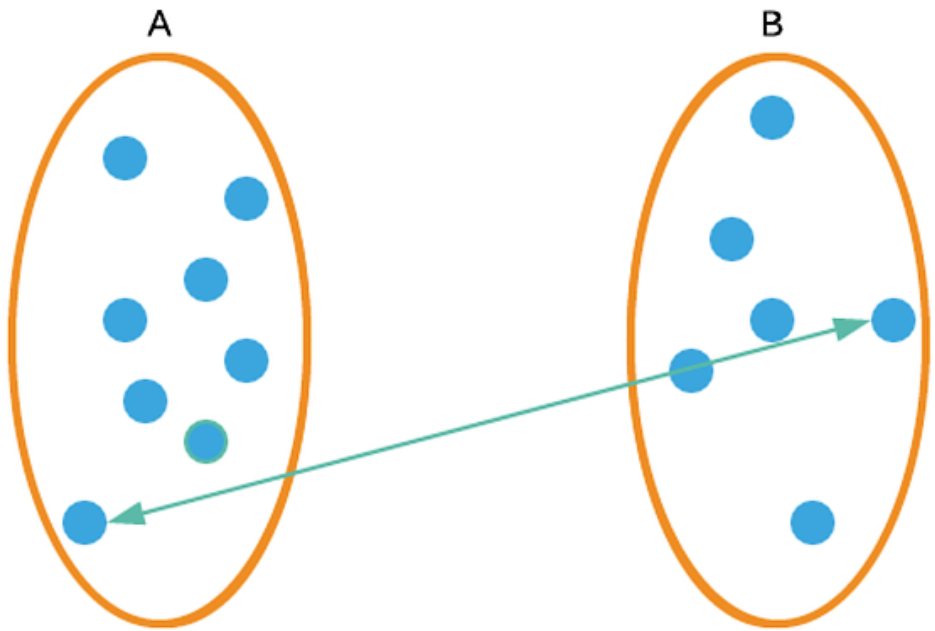
Single linkage

- **Distance: shortest distance** between two points in each cluster



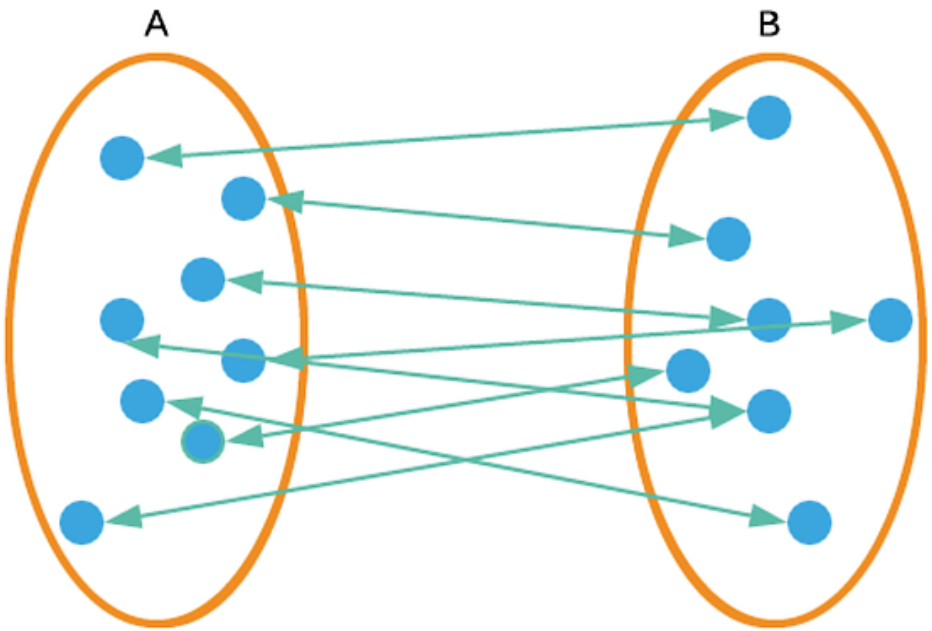
Complete Linkage

- **Distance**: **longest distance** between two points in each cluster



Average Linkage

- **Distance:** **average distance** between each point in one cluster to each point in the other



Expected outcome

- The expected outcome of hierarchical clustering is that **every data point will be placed into a cluster**
- You are able to see **which points are placed where** and just how exactly **the clusters get larger and decrease in number**

Knowledge check 3

Module completion checklist

Objective	Complete
Define what makes k-means an unsupervised method of machine learning	✔
Prepare the dataset to be clustered using k-means	✔
Run and visualize k-means with k=2	✔
Find the optimal number of k using elbow method and visualize	✔
Inspect the results of clustering the dataset using the optimal k	✔
Discuss common pitfalls of clustering	✔
Summarize hierarchical clustering and its two types	✔
Understanding agglomerative clustering and types of linkage methods	✔
Implement hierarchical clustering on the data and visualize its dendrogram	

Distance matrix

We can use the function `pdist` to find the distance matrix of a dataframe:

```
pdist(X,
      metric)
```

- Main arguments of `pdist`:
  - `X`: a numeric matrix or data.frame
  - `metric`: the distance metric to use

scipy.spatial.distance.pdist

scipy.spatial.distance.pdist(*X*, *metric='euclidean'*, *p=2*, *w=None*, *V=None*, *VI=None*) [source]

Pairwise distances between observations in n-dimensional space.

The following are common calling conventions.

1. `Y = pdist(X, 'euclidean')`

   Computes the distance between m points using Euclidean distance (2-norm) as the distance metric between the points. The points are arranged as m n-dimensional row vectors in the matrix X.

Distance matrix

- We have already scaled our dataset
- Let's calculate the distance from the scaled data

```
temp_heart_hier = temp_heart_kmeans
```

- Let's find the distance matrix of `temp_heart_hier`

```
# Calculate the distance matrix.
temp_heart_dist = pdist(temp_heart_hier, 'euclidean')
temp_heart_dist
```

```
array([0.09422206, 0.18276427, 0.34907561, ..., 0.03834566, 0.20953818,
       0.18050346])
```

scipy.cluster.hierarchy.linkage

- One method for agglomerative clustering is to use the function `linkage`

```
linkage(y,
        method = ...)
```

- Main arguments of `linkage`:
  - `y`: a distance matrix
  - `method`: the linkage algorithm to use

scipy.cluster.hierarchy.linkage

scipy.cluster.hierarchy.linkage(*y*, *method='single'*, *metric='euclidean'*) [source]

Performs hierarchical/agglomerative clustering on the condensed distance matrix y.

y must be a $\binom{n}{2}$ sized vector where n is the number of original observations paired in the distance matrix. The behavior of this function is very similar to the MATLAB linkage function.

Parameters: **y** *: ndarray*

A condensed or redundant distance matrix. A condensed distance matrix is a flat array containing the upper triangular of the distance matrix. This is the form that `pdist` returns. Alternatively, a collection of m observation vectors in n dimensions may be passed as an m by n array.

method *: str, optional*

The linkage algorithm to use. See the `Linkage Methods` section below for full descriptions.

metric *: str, optional*

The distance metric to use. See the `distance.pdist` function for a list of valid distance metrics.

Returns: **Z** *: ndarray*

The hierarchical clustering encoded as a linkage matrix.

Agglomerative clustering

- Let's get the clustered linkage matrix for our distance matrix `temp_heart_dist`:

```
# Linkage matrix
temp_heart_linked = linkage(temp_heart_dist, 'ward')
temp_heart_linked[:5]
```

```
array([[ 25.,  30.,   0.,   2.],
       [ 82., 130.,   0.,   3.],
       [ 50., 101.,   0.,   2.],
       [ 86.,  89.,   0.,   2.],
       [ 42.,  99.,   0.,   2.]])
```

Visualizing dendrogram

We use `dendrogram` to visualize dendrograms from `scipy`

```
dendrogram(Z,
           ...)
```

- Main arguments of `dendrogram`:
  - `Z`: the linkage matrix encoding the hierarchical clustering to render as a dendrogram

scipy.cluster.hierarchy.dendrogram

scipy.cluster.hierarchy.dendrogram(*Z, p=30, truncate_mode=None, color_threshold=None, get_leaves=True, orientation='top', labels=None, count_sort=False, distance_sort=False, show_leaf_counts=True, no_plot=False, no_labels=False, color_list=None, leaf_font_size=None, leaf_rotation=None, leaf_label_func=None, no_leaves=False, show_contracted=False, link_color_func=None, ax=None***)** [source]

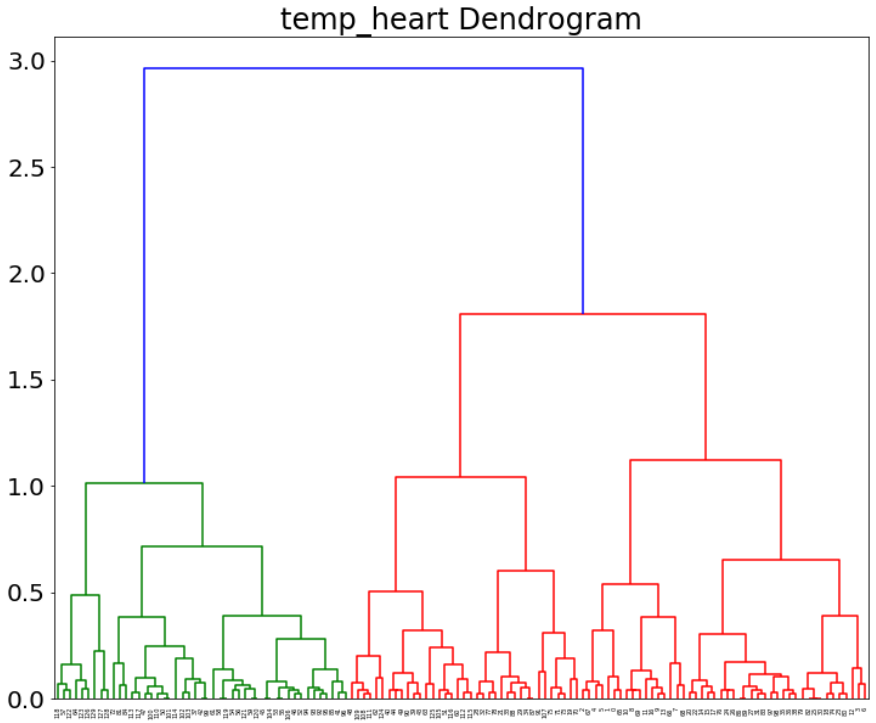
Plots the hierarchical clustering as a dendrogram.

Visualizing dendrogram

- Let's visualize the dendrogram for `temp_heart_linked`

```
# Plot the dendrogram.
plt.figure(figsize = (12, 10))
plt.title( "temp_heart Dendrogram")
dendo = dendrogram(temp_heart_linked)
plt.show()
```

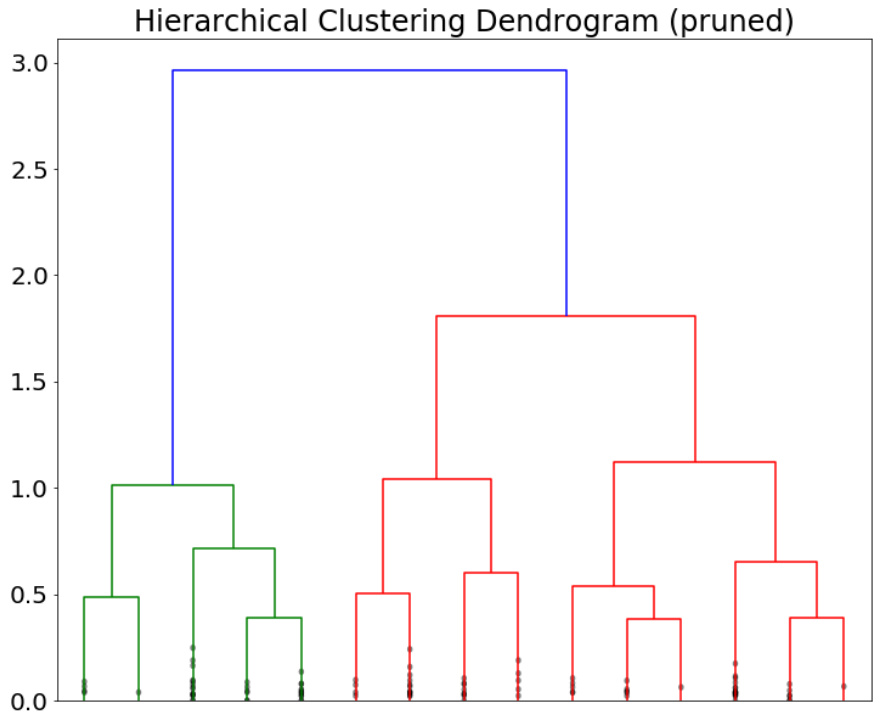
- You will see from this example that hierarchical clustering may be **informative**, but it is **hard to interpret**



Optimizing and pruning the dendrogram

- Let's say we want to view the optimized number of clusters
- Suppose we wish to see only the last 15 merges or lead nodes from the data

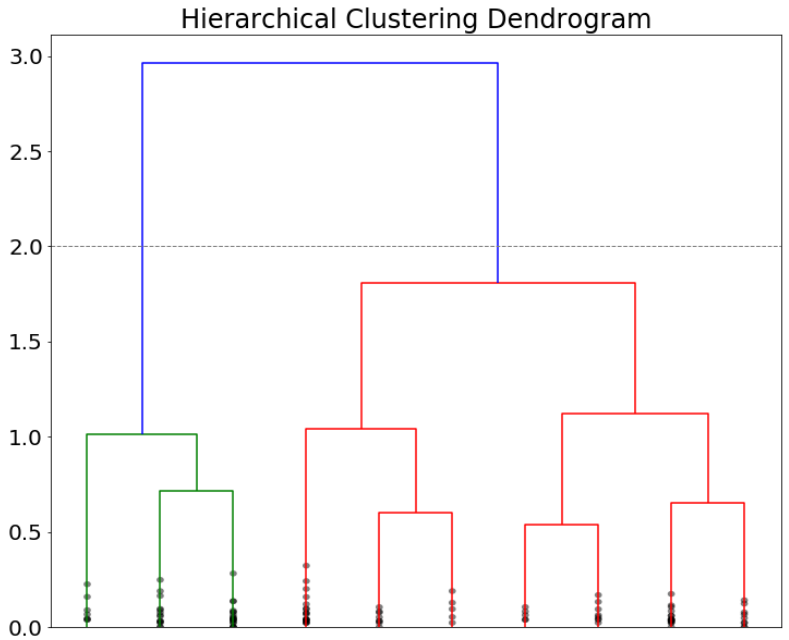
```
# Prune the tree to 15 leaf nodes.
plt.figure(figsize=(12,10))
plt.title('Hierarchical Clustering Dendrogram
(pruned)')
dendrogram(temp_heart_linked,
           truncate_mode='lastp',   # show only
the last p merged clusters
           p = 15,                  # show only
the last p merged clusters
           show_leaf_counts=False,  # otherwise
numbers in brackets are counts
           leaf_rotation=90,
           leaf_font_size=12,
           show_contracted=True,   # to get a
distribution impression in truncated branches
)
plt.show()
```



Optimizing and pruning the dendrogram

- We can use `dendrogram` to cut the tree where it makes most sense
- Now let's view the last 10 merges, with a cutoff at 2 clusters
- We can set the distance on the y-axis to cut the dendrogram using `axhline` from `matplotlib.pyplot`

```
# View only the last 5 merges.
plt.figure(figsize = (12, 10))
plt.title('Hierarchical Clustering Dendrogram')
dendrogram(temp_heart_linked,
           truncate_mode = 'lastp',
           p = 10,
           show_leaf_counts = False,
           leaf_rotation = 90,
           leaf_font_size = 12,
           show_contracted = True,
           color_threshold = 2   # <- nodes greater
than the threshold are colored blue
)
# Add horizontal line.
plt.axhline(y = 2, c='grey', lw=1,
linestyle='dashed')
```



Assign data points to clusters

- We can also use `AgglomerativeClustering` from `scikit-learn` for clustering
- While it is useful in determining the clusters to which the data points belong to, it doesn't return the distance between clusters

```
# Assign the data points to clusters.

temp_heart_clusters = AgglomerativeClustering(n_clusters = 2,        # <- 2 clusters
                                              affinity = 'euclidean',
                                              linkage = 'ward')

temp_heart_clusters.fit_predict(temp_heart_hier)
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0,
       0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0,
       0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0,
       1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1])
```

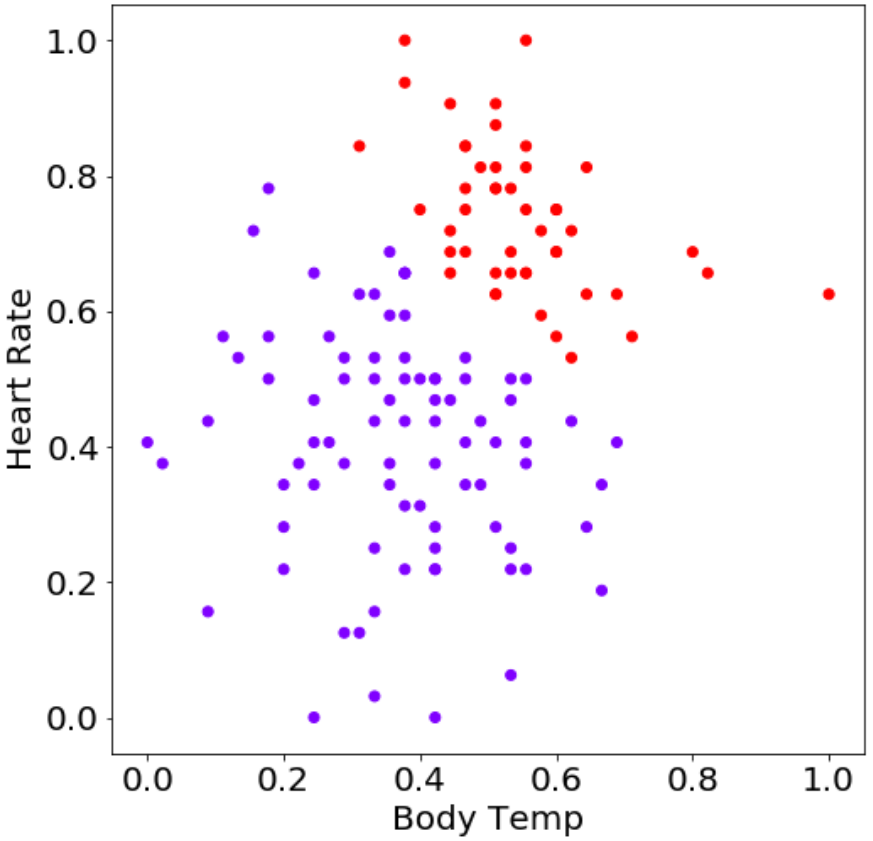
```
print(temp_heart_clusters.labels_)
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 1 0 0 0 1 1 0 0 1 1 1 0 0 1 0 1 1 1 1 1 1 1 1 0 1 0 0 1 0 0 0 0 0 0 0 1 0
 0 0 0 0 0 0 0 1 0 0 1 1 0 0 0 0 0 0 1 1 1 1 1 0 0 1 1 1 1 1 1 0 1 0 0 0 1
 0 0 1 1 0 0 1 1 1 1 1 1 1 0 0 1 1 1 1]
```

Assign data points to clusters

- Can we see **patterns** or **groups** within the data?
- Can we cluster data better with this method than with k-means, do we see anything different?

```
# Plot the clusters.
plt.figure(figsize = (12, 10))
plt.xlabel('Body Temp')
plt.ylabel('Heart Rate')
plt.scatter(temp_heart_hier['Body Temp'],
temp_heart_hier['Heart Rate'], c =
temp_heart_clusters.labels_, cmap = 'rainbow')
```



Knowledge check 4

Exercise 3

Module completion checklist

Objective	Complete
Define what makes k-means an unsupervised method of machine learning	✔
Prepare the dataset to be clustered using k-means	✔
Run and visualize k-means with k=2	✔
Find the optimal number of k using elbow method and visualize	✔
Inspect the results of clustering the dataset using the optimal k	✔
Discuss common pitfalls of clustering	✔
Summarize hierarchical clustering and its two types	✔
Understanding agglomerative clustering and types of linkage methods	✔
Implement hierarchical clustering on the data and visualize its dendrogram	✔

Workshop!

- Workshops are to be completed in the afternoon either with a dataset for a capstone project or with another dataset of your choosing
- Make sure to annotate and comment your code so that it is easy for others to understand what you are doing
- This is an exploratory exercise to get you comfortable with the content we discussed today
- Today, you will:
  - Prepare your dataset and scale to implement the clustering algorithms
  - Implement k means clustering
  - Implement hierarchical clustering
  - Find interesting patters in the clusters

Congratulations on completing this module!