DATA SOCIETY:

Machine learning - Part 1

One should look for what is and not what he thinks should be. (Albert Einstein)

Welcome

Who we are

- Data Society's mission is to **integrate Big Data and machine learning best practices across entire teams** and empower professionals to identify new insights
- We provide:
    - High-quality data science training programs
    - Customized executive workshops
    - Custom software solutions and consulting services
- Since 2014, we've worked with thousands of professionals to make their data work for them

Course structure

- **Lecture slides:**
  - 2.5 hours

- **Knowledge checks and exercises:**
  - 30 minutes

Why study machine learning?

Explore this article about how machine learning is used across industries:

https://builtin.com/artificial-intelligence/machine-learning-examples-applications

Using the chat box, answer at least one of the following questions:

Did any of the examples surprise you?
Which examples did you find the most interesting?
Can you think of how you would use machine learning?

List of files we will be using today

- Throughout our learning session today, we will be using the files below. Please make sure to download these files before you start working!
- **Slides**: day1-machine-learning-in-python.pdf
- **In class code file for slides**: day1_machine_learning_in_python_code.ipynb
- **Knowledge checks**: Day 1 Machine learning in Python - Knowledge Checks.pdf
- **Exercises**: day1-machine-learning-in-python-exercises.ipynb
- **Exercises with answers**: day1-machine-learning-in-python-exercises-with-answers.ipynb

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	

Loading the packages

- Let's make sure we have the packages we will need for the data cleaning, plotting, and clustering:

```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import sklearn
from sklearn import cluster
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
```

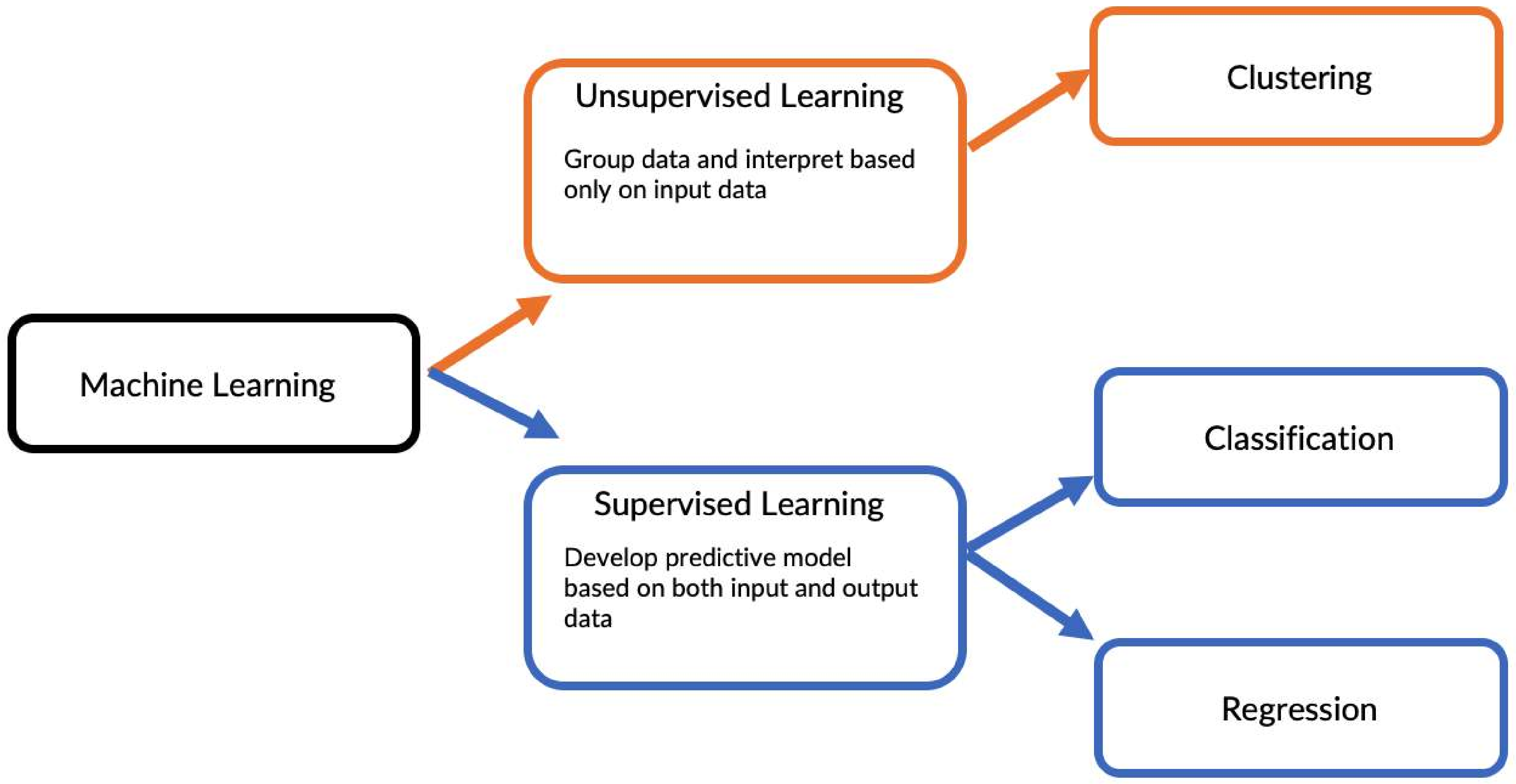
Directory settings

- In order to maximize the efficiency of your workflow, you should encode your directory structure into `variables`
- We will use the `pathlib` library
- Let the `main_dir` be the variable corresponding to your `skillsoft-machine-learning-2021` folder
- `data_dir` be the variable corresponding to your `data` folder

```
# Set 'main_dir' to location of the project folder
from pathlib import Path
home_dir = Path(".").resolve()
main_dir = home_dir.parent
print(main_dir)
```

```
data_dir = str(main_dir) + "/data"
print(data_dir)
```

Supervised vs. unsupervised learning


Machine Learning
Unsupervised Learning
Group data and interpret based only on input data
Clustering
Supervised Learning
Develop predictive model based on both input and output data
Classification
Regression

DATASOCIETY: © 2021

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
4. It depends on **attributes** of data
5. Usually measured by a *distance metric*
6. 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

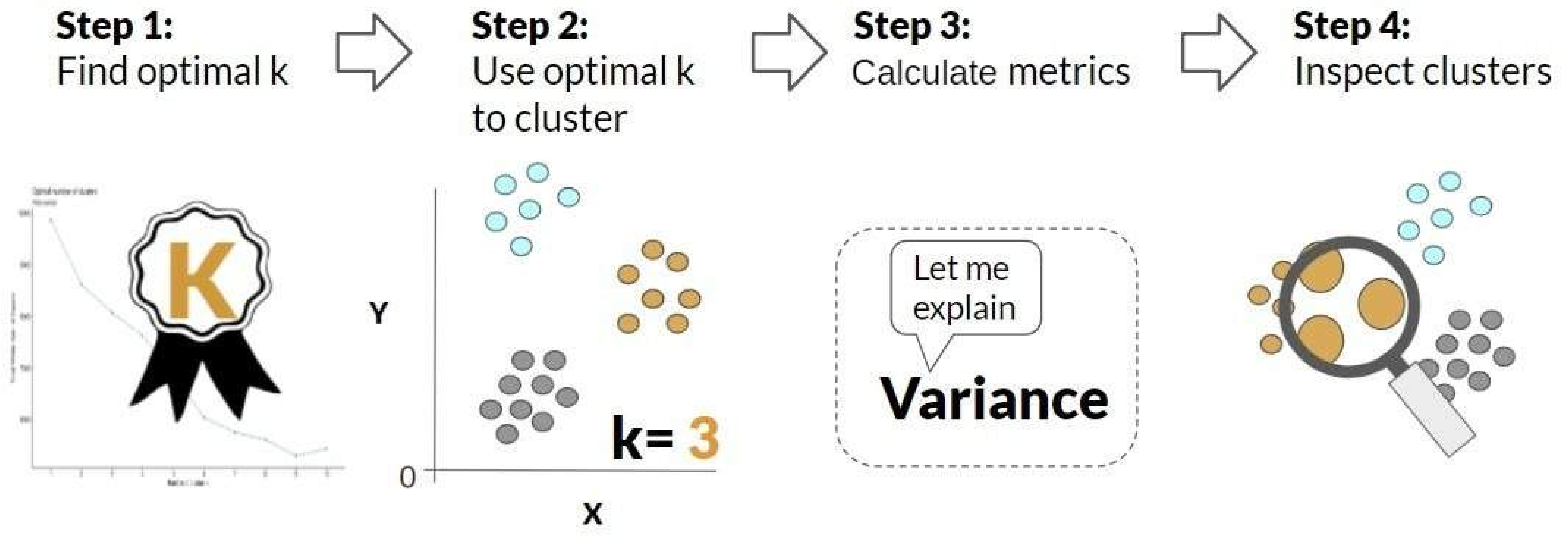
- You can ask questions like **“What patterns are in this data?”** and **“Which points are similar to each other?”**

DATASOCIETY: © 2021

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



DATASOCIETY: © 2021

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 and silhouette methods and visualize	
Inspect the results of clustering the dataset using the optimal k	
Discuss common pitfalls of clustering	

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) match 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:

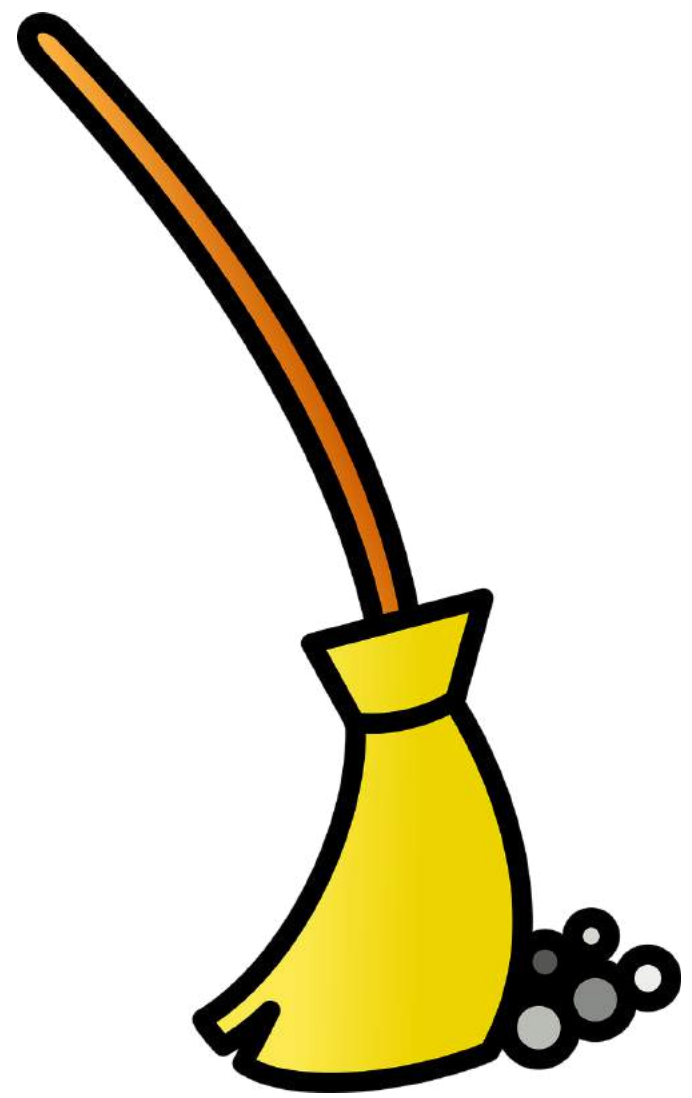
1. **Body temperature**
2. **Heart rate**

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)
```

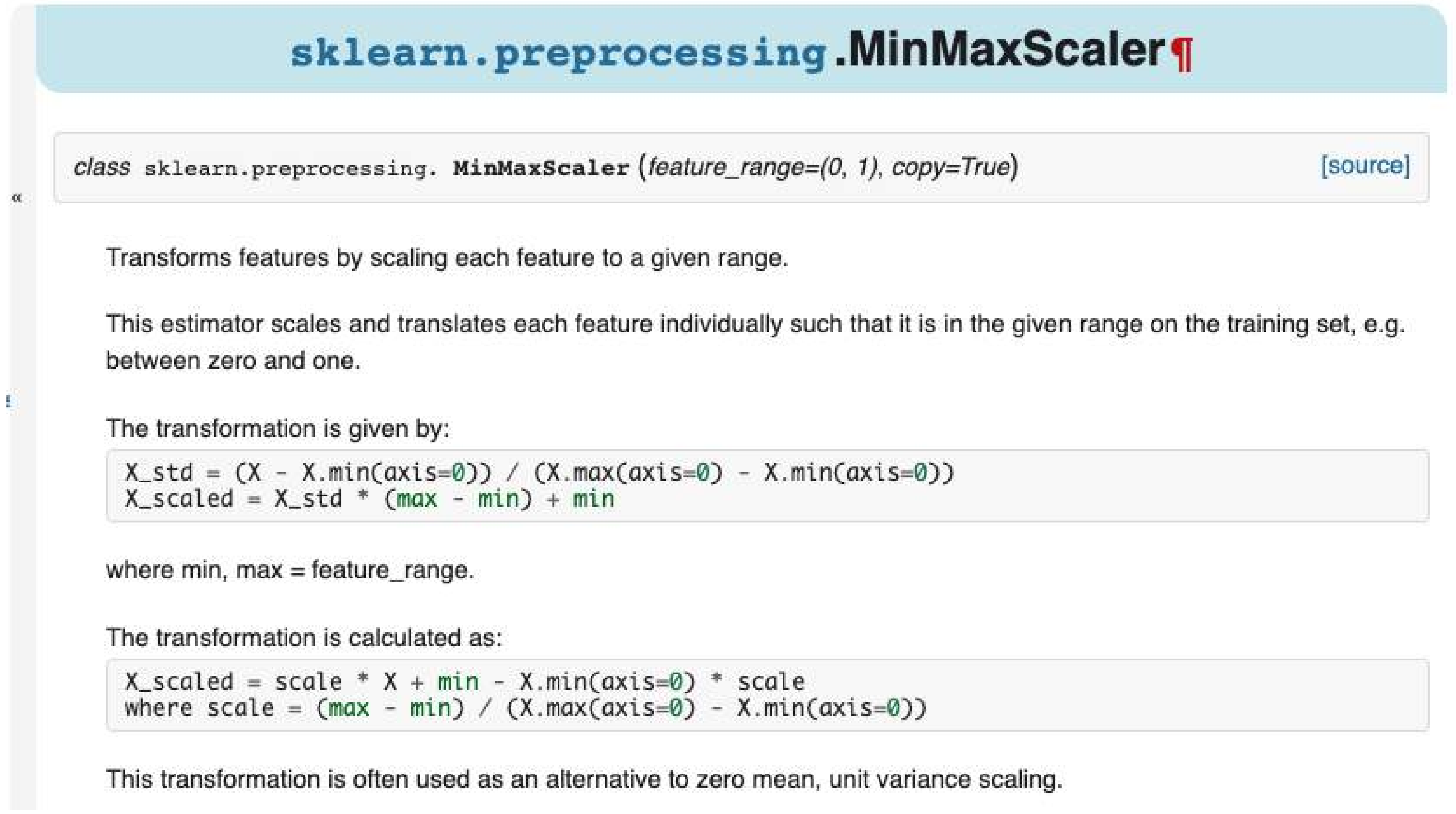
```
Body Temp      float64
Heart Rate       int64
dtype: object
```

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())
```

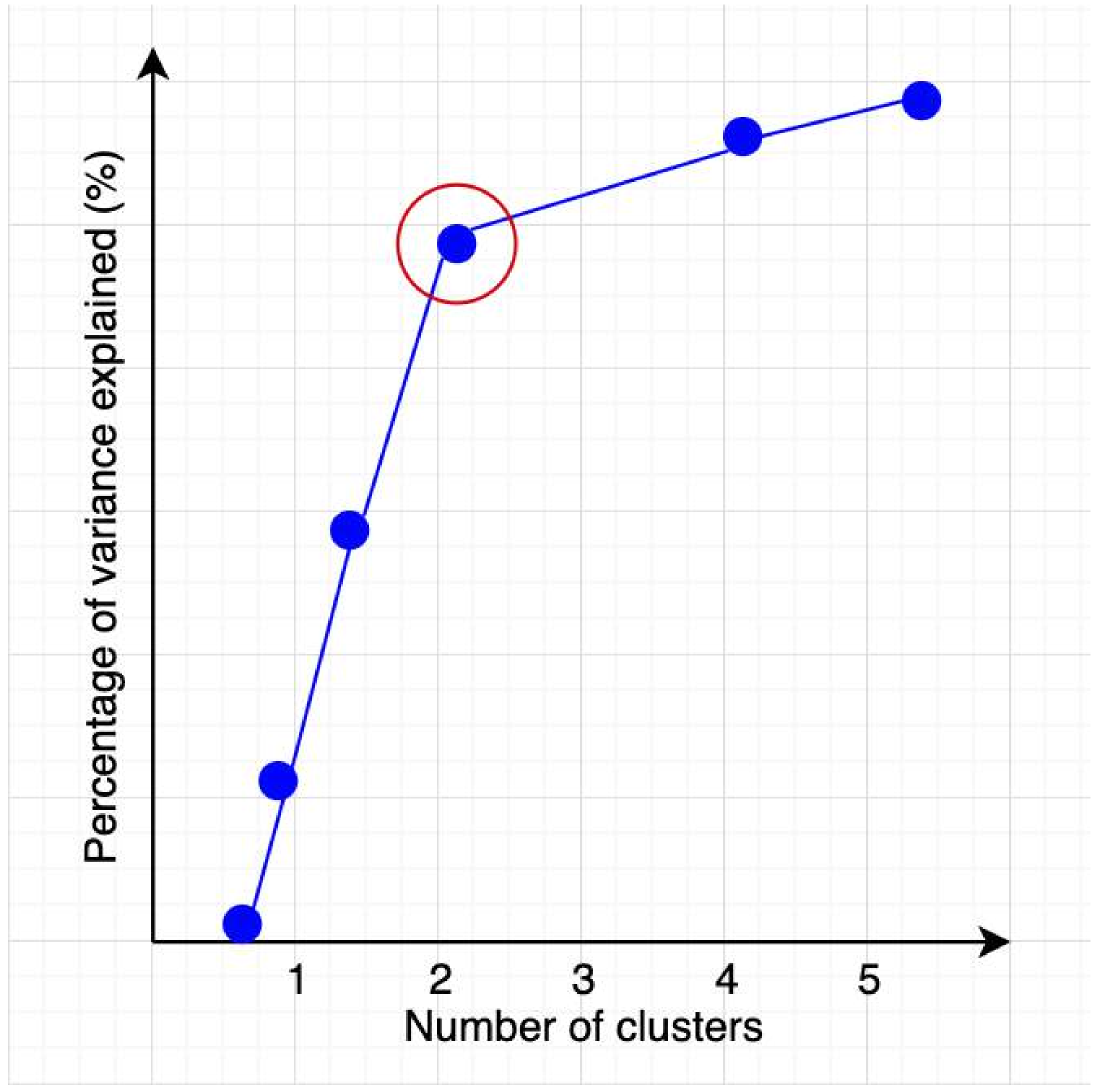
```
   Body Temp  Heart Rate
0   0.000000     0.40625
1   0.088889     0.43750
2   0.133333     0.53125
3   0.155556     0.71875
4   0.177778     0.50000
```

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 and silhouette methods and visualize	
Inspect the results of clustering the dataset using the optimal k	
Discuss common pitfalls of clustering	

Find the best k

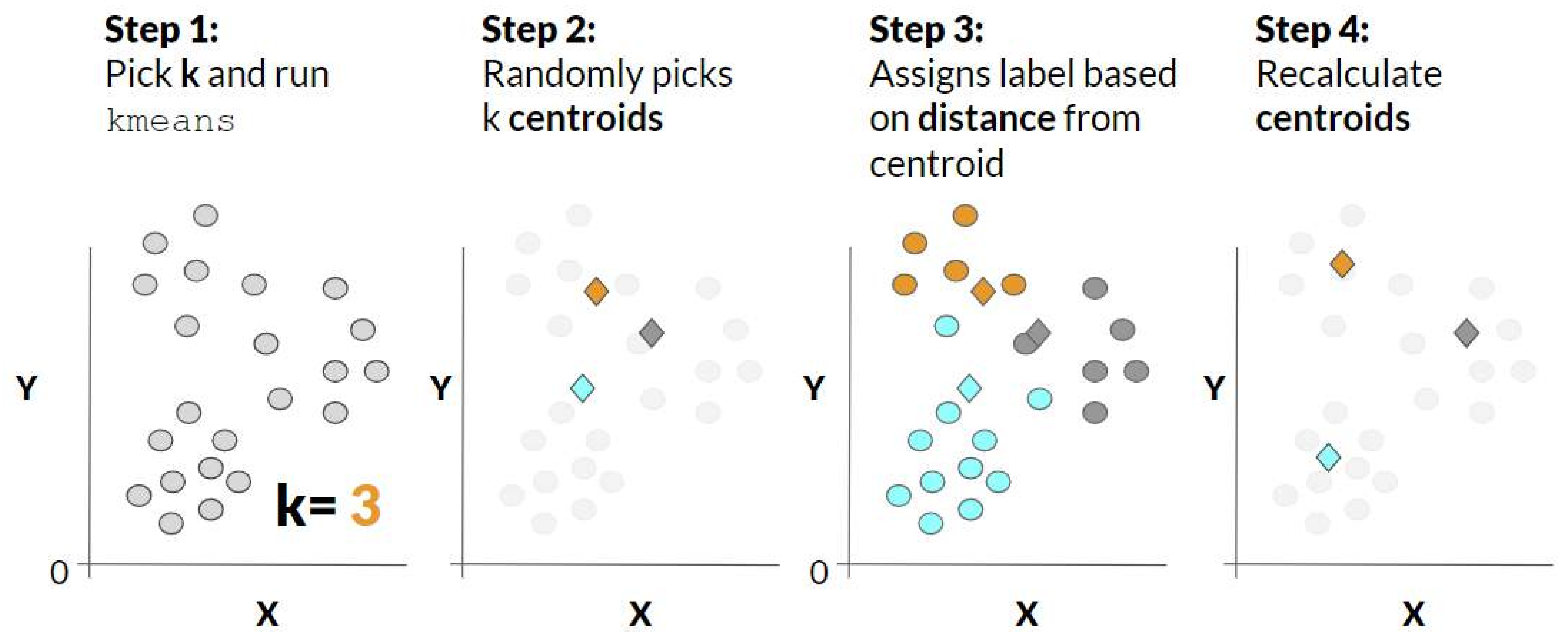
- First, we need to **run the k-means algorithm** on the dataset for `k = 1` through `k = 10`
- We will start with k = 2 since we hope to find more than one cluster in this dataset



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

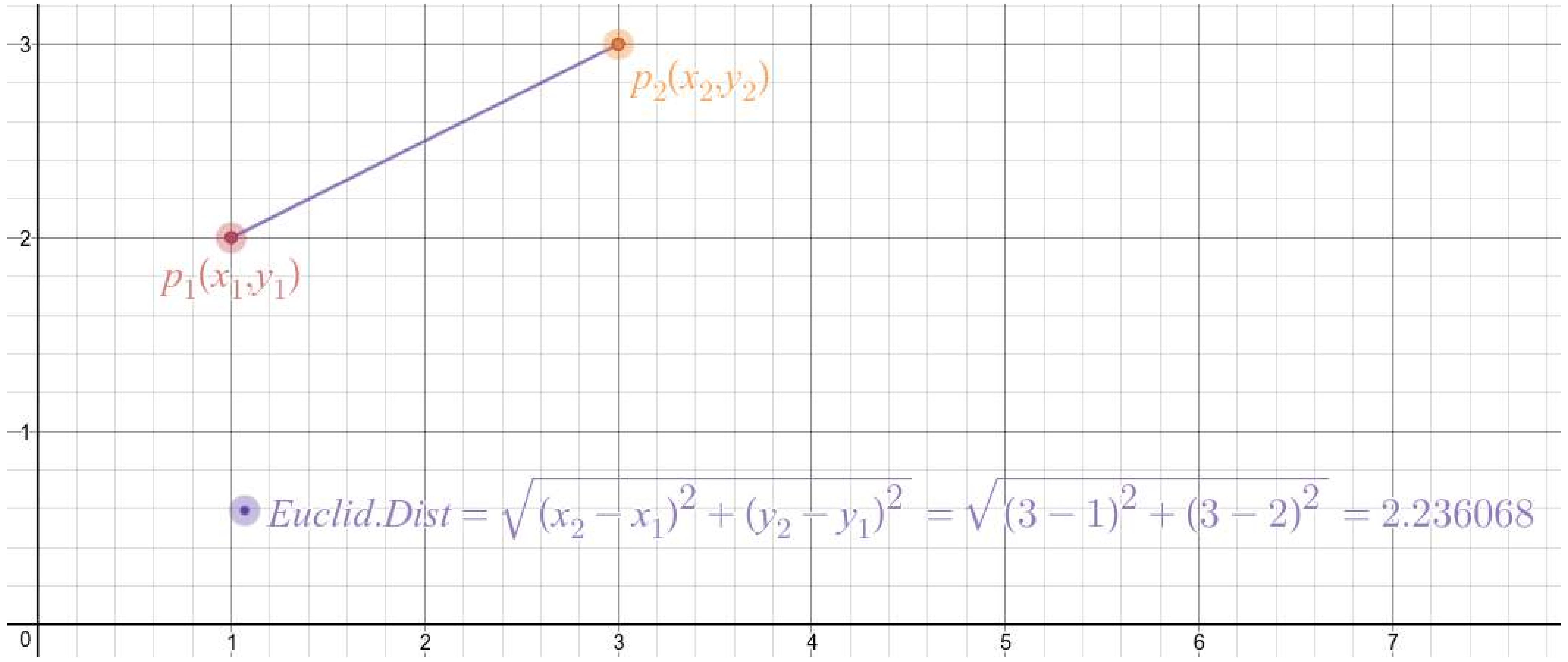


- A centroid is the center of a cluster. It's an n-dimentional vector where n is the number of variables (x), and each value is simply the mean of x1, x2,...,xn of all observations in the cluster
- **Repeats steps 3 and 4** again until **no more reassignments** are made, and **optimal** clusters are formed

DATASOCIETY: © 2021

K-means: uses Euclidean distance

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

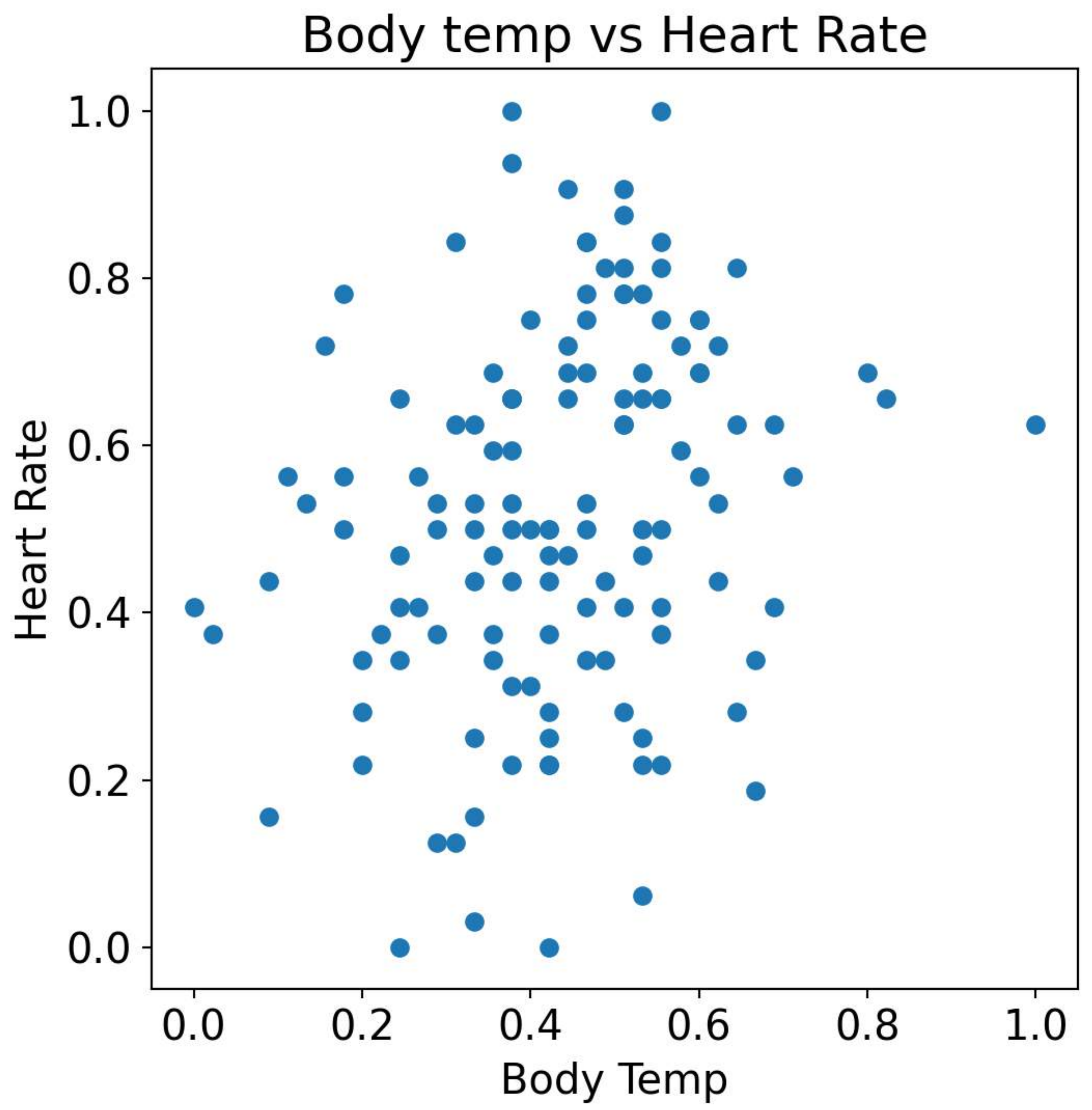


DATASOCIETY: © 2021

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

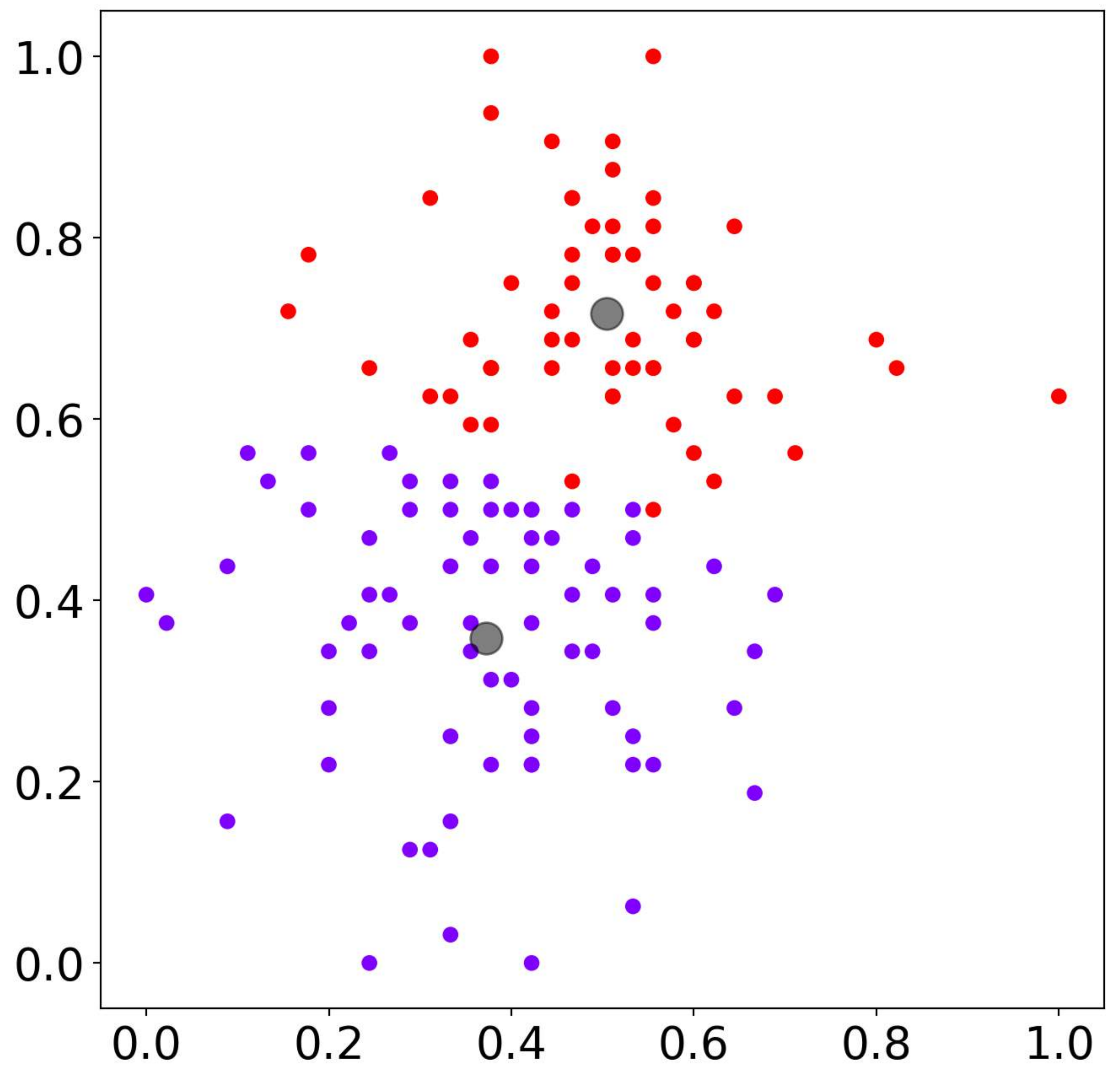
```
# K-means - start with 2 clusters.
# Initializing k-means.
kmeans_2 = KMeans(n_clusters=2)
# Fitting with inputs.
kmeans_2 = kmeans_2.fit(temp_heart_kmeans)
# Predicting the clusters.
labels = kmeans_2.predict(temp_heart_kmeans)
# Getting the cluster centers.
C_2 = kmeans_2.cluster_centers_
print(C_2)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

```
[[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)
```



DATASOCIETY: © 2021

Knowledge check 1

DATASOCIETY: © 2021

Exercise 1

DATASOCIETY: © 2021

Module completion checklist

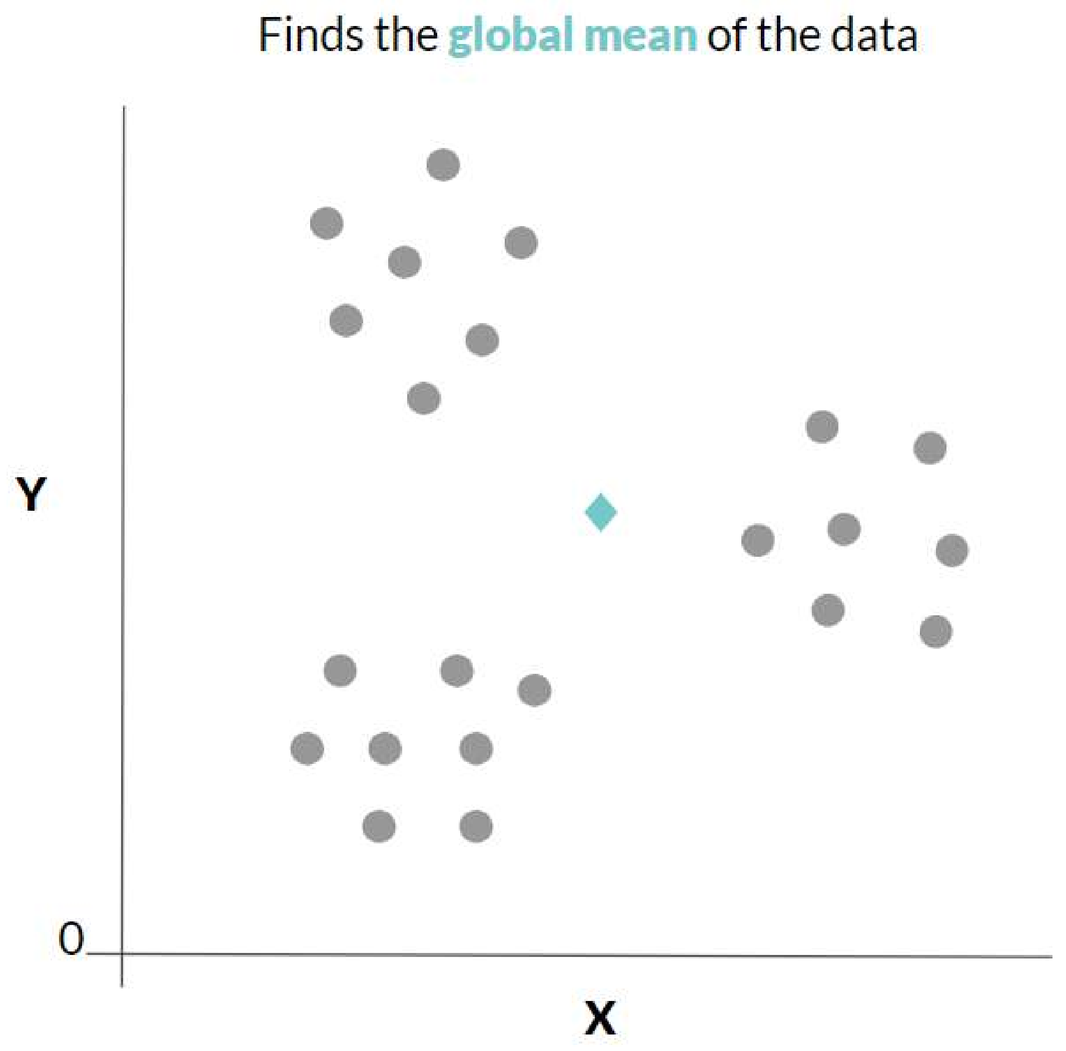
Objective	Complete
Define what makes k-means an unsupervised method of machine learning	✔
Prepare the dataset to be clustered using k-means	✔
Run and visualize k-means with k=2	✔
Find the optimal number of k using elbow and silhouette methods and visualize	
Inspect the results of clustering the dataset using the optimal k	
Discuss common pitfalls of clustering	

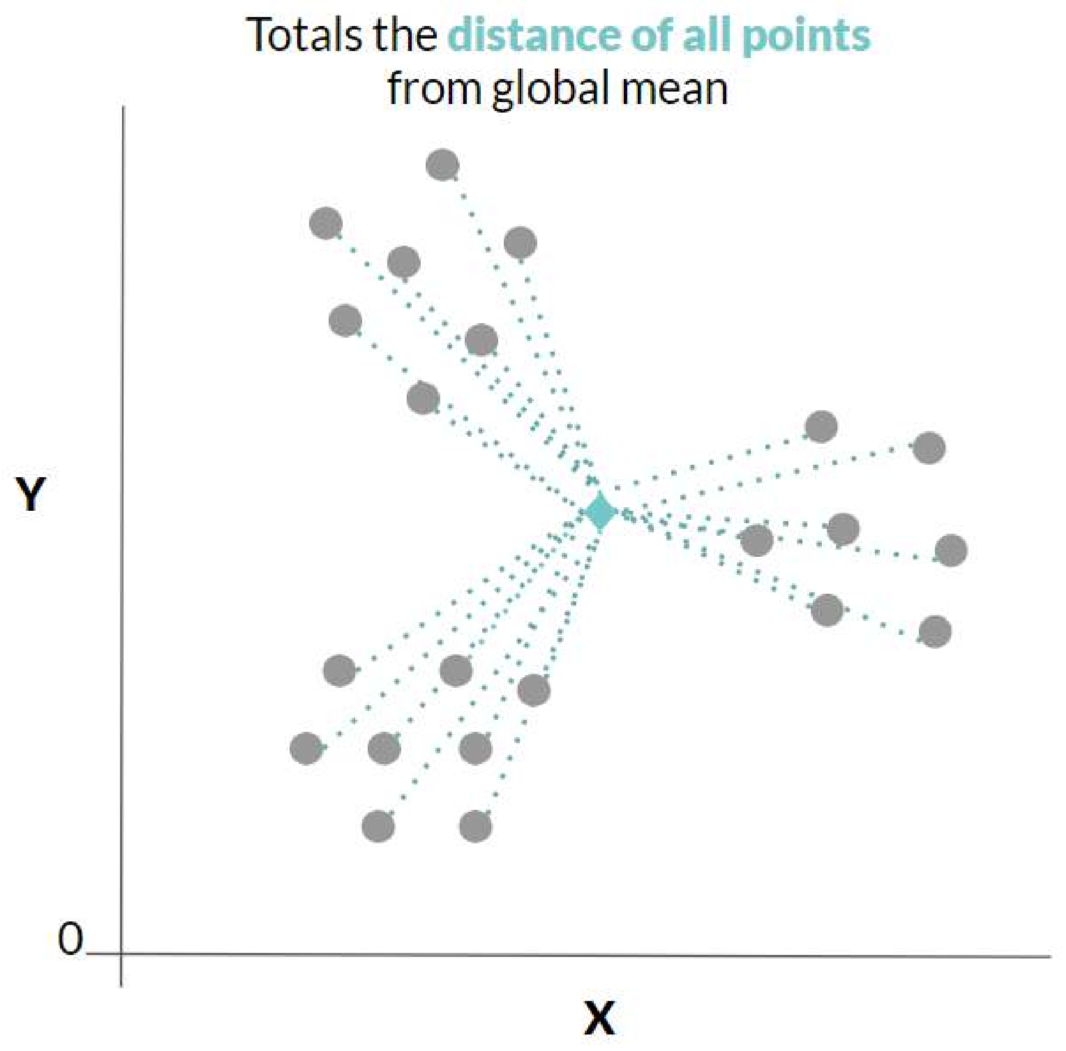
Evaluating your clusters

- 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 i.e global mean of the data
- The global mean is usually calculated by taking the average of all the data points in the feature space
- This is used as the denominator for explained variance

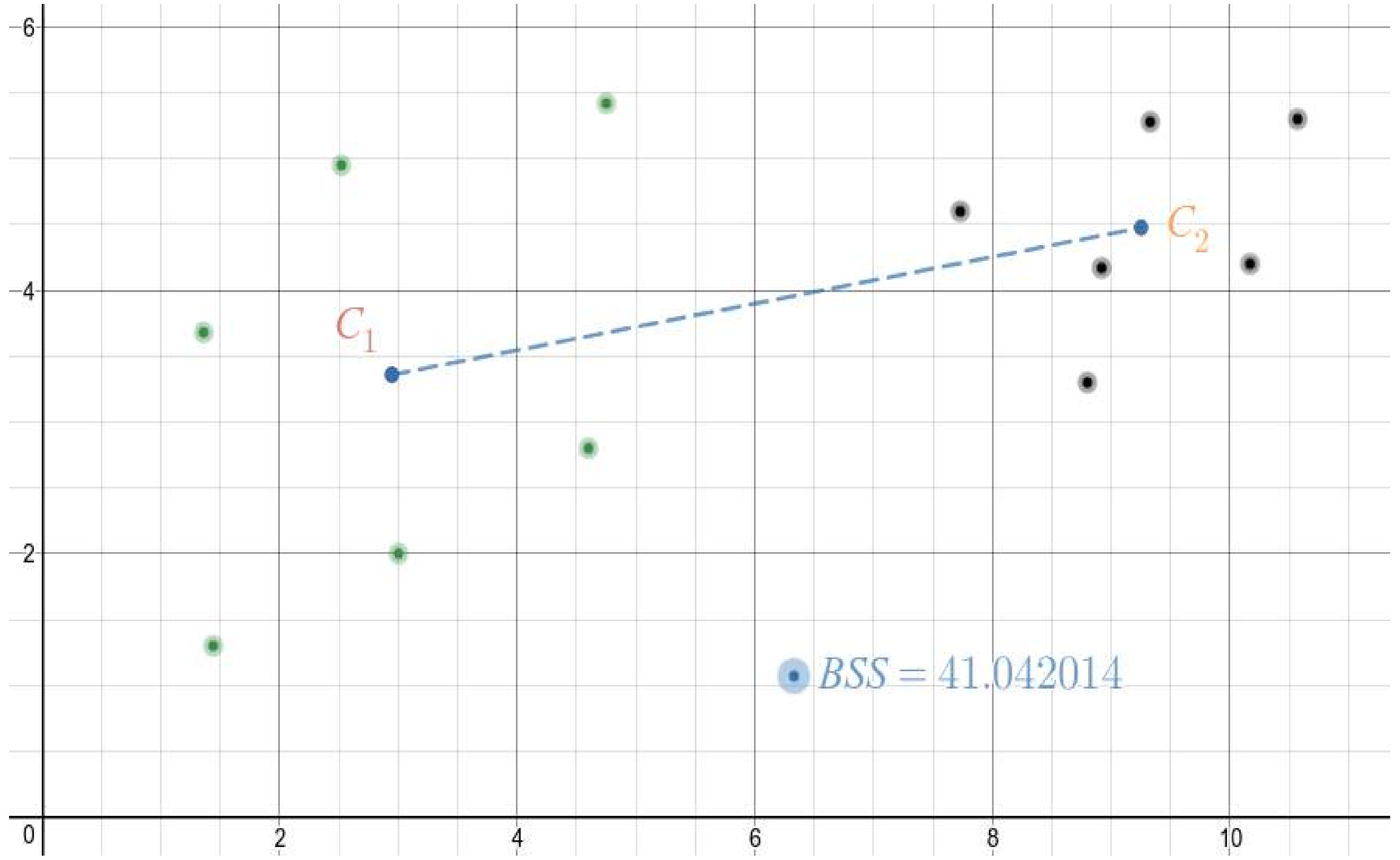


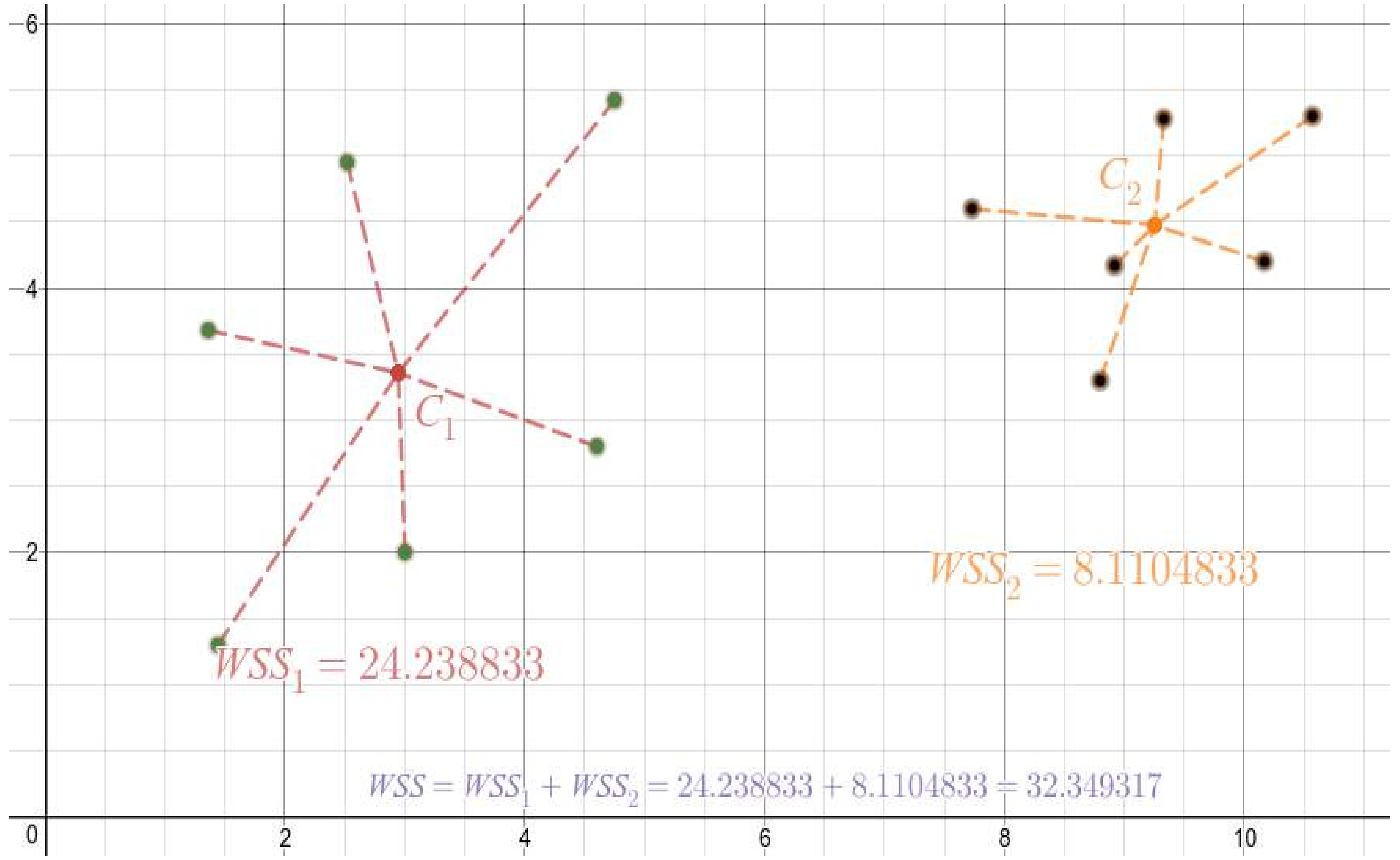


BSS & WSS

- **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**

- **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**



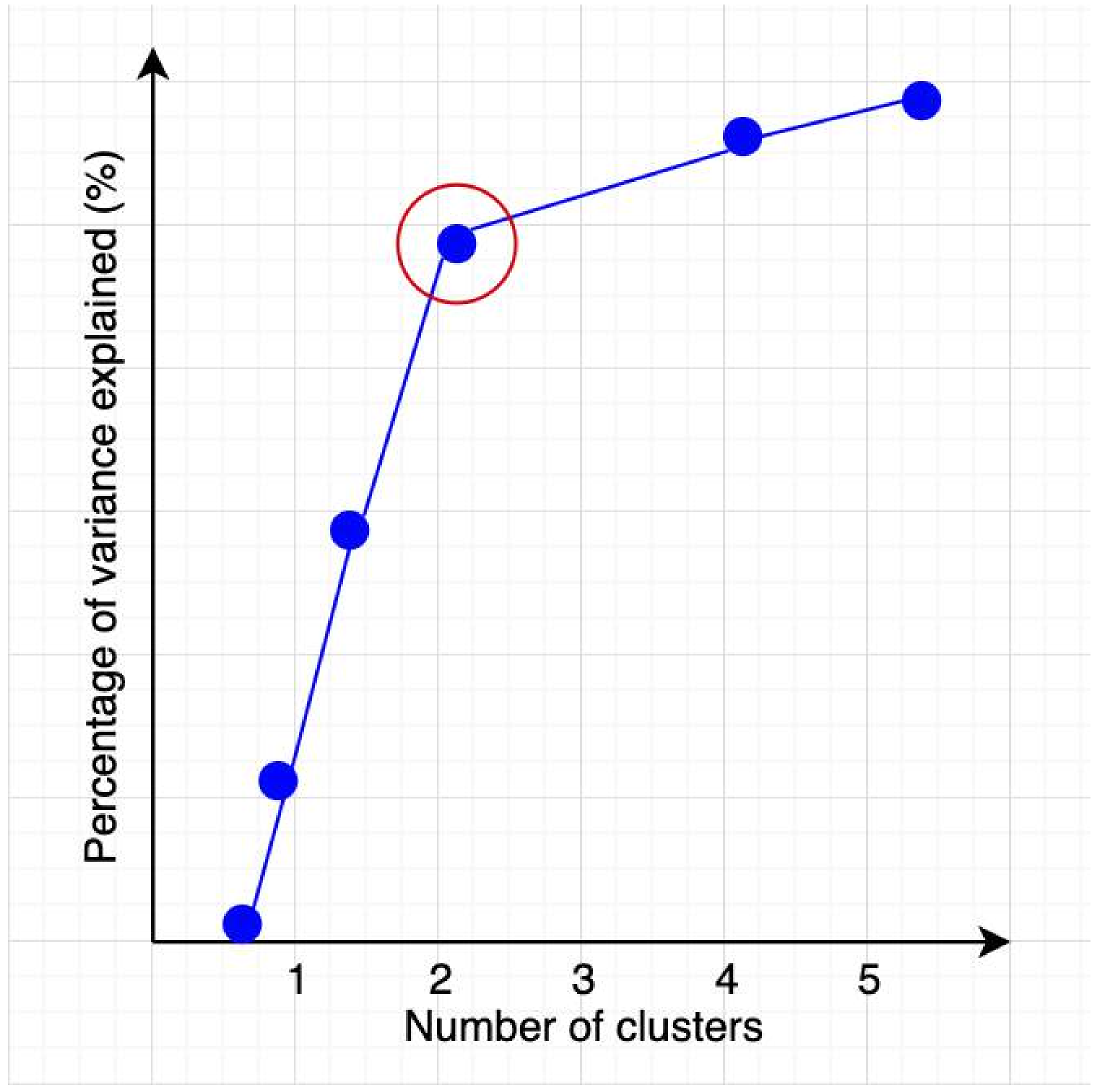


Evaluate explained variance

- `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** using the Elbow method and Silhouette method
  - Calculate these metrics for `k = 2` as well as the optimal number of clusters

K-means: methods to optimize k

- **Method 1: 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



K-means: methods to optimize k

- **Method 2: Silhouette plot**
- We could also use the **silhouette coefficient**
- The **silhouette coefficient** is calculated using:
    - the mean intra-cluster distance (`a`)
    - the mean nearest cluster distance for each sample (`b`)
    - the coefficient = **`(b-a) / max(a, b)`**
- The silhouette analysis measures **how well** an observation is clustered and it estimates the average **distance between clusters**
- The silhouette plot displays a measure of how close each point in one cluster is to points in the neighboring clusters

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

DATASOCIETY: © 2021

K-means: 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')
```

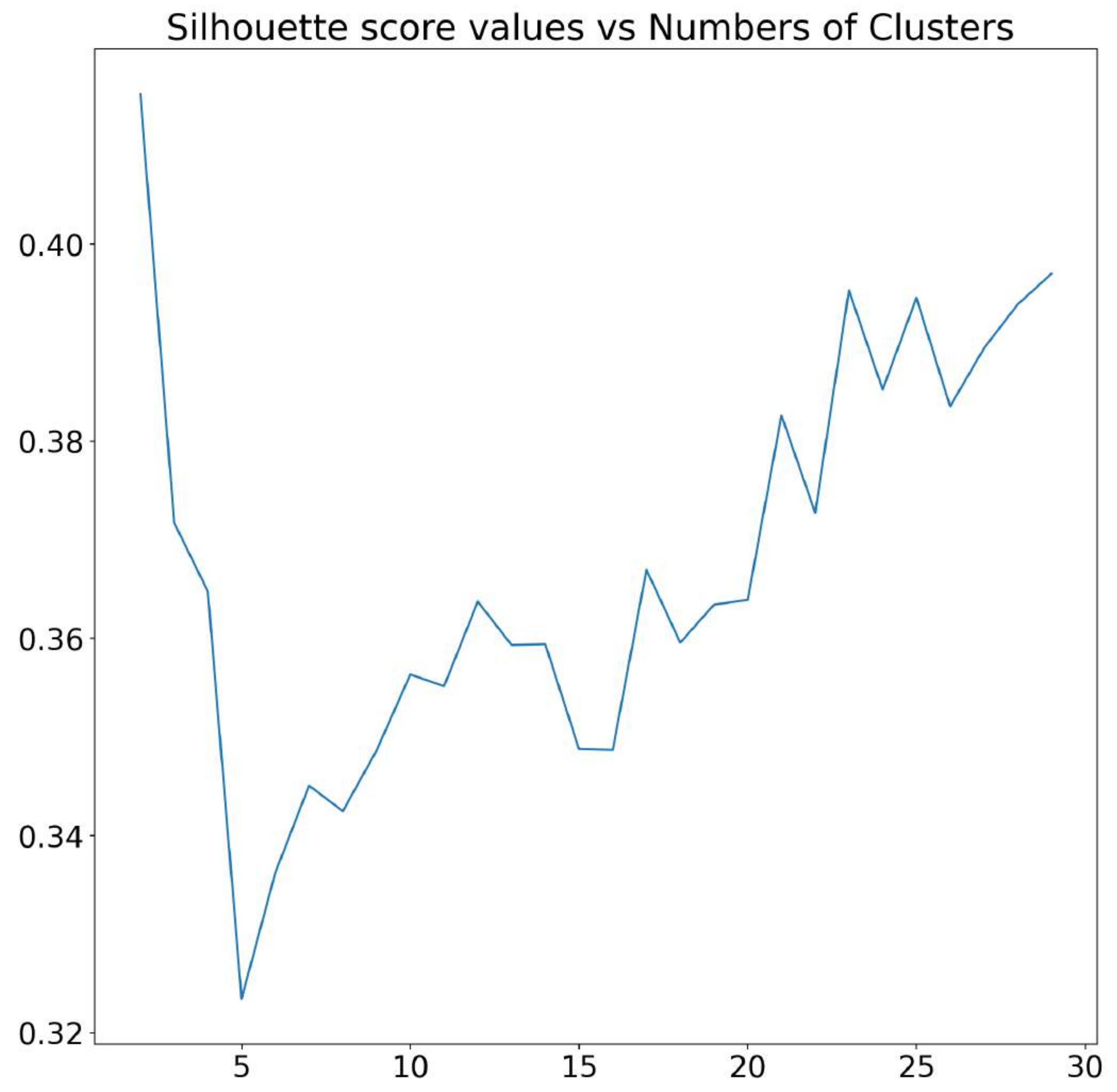
K-means: Elbow plot method - cont’d

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

K-means: silhouette method

```
obs = temp_heart_kmeans
silhouette_score_values=list()
NumberOfClusters = range(2,30)
for i in NumberOfClusters:
    classifier=cluster.KMeans(i,init='k-means++', n_init=10,
                                 max_iter=300,
                                 tol=0.0001,
                                 verbose=0,
                                 random_state=None,
                                 copy_x=True)
    classifier.fit(obs)
    labels= classifier.predict(obs)
    sklearn.metrics.silhouette_score(obs,labels ,metric='euclidean', sample_size=None,
random_state=None)
    silhouette_score_values.append(sklearn.metrics.silhouette_score(obs,labels ,metric='euclidean',
sample_size=None, random_state=None))
plt.plot(NumberOfClusters, silhouette_score_values)
plt.title("Silhouette score values vs Numbers of Clusters ")
```

K-means: silhouette method



```
Optimal_NumberOf_Components=NumberOfClusters[silho

print("Optimal number of components is:",
Optimal_NumberOf_Components)
```

```
Optimal number of components is: 2
```

K-means: silhouette method

- Silhouette method indicates **the optimal number of clusters as 2**
- We already ran k-means with 2 clusters, hence let's go ahead with 5 clusters and compare the two cases
- Choosing the number of clusters is **subjective**
    - **Depends on our goal and domain-specific knowledge**

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)
```

```
75.61283303653335
```

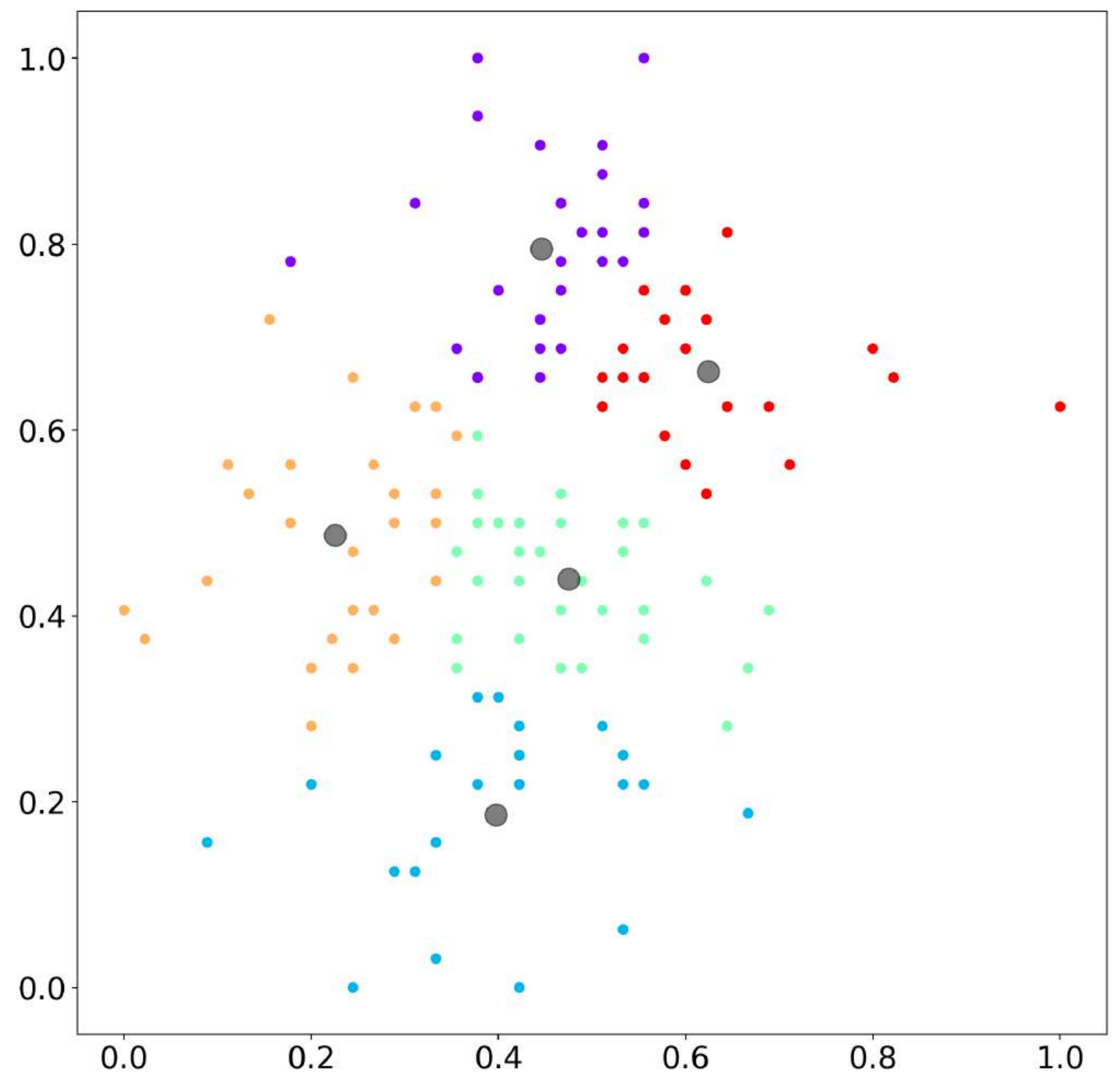
Run optimal k

- 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)
```



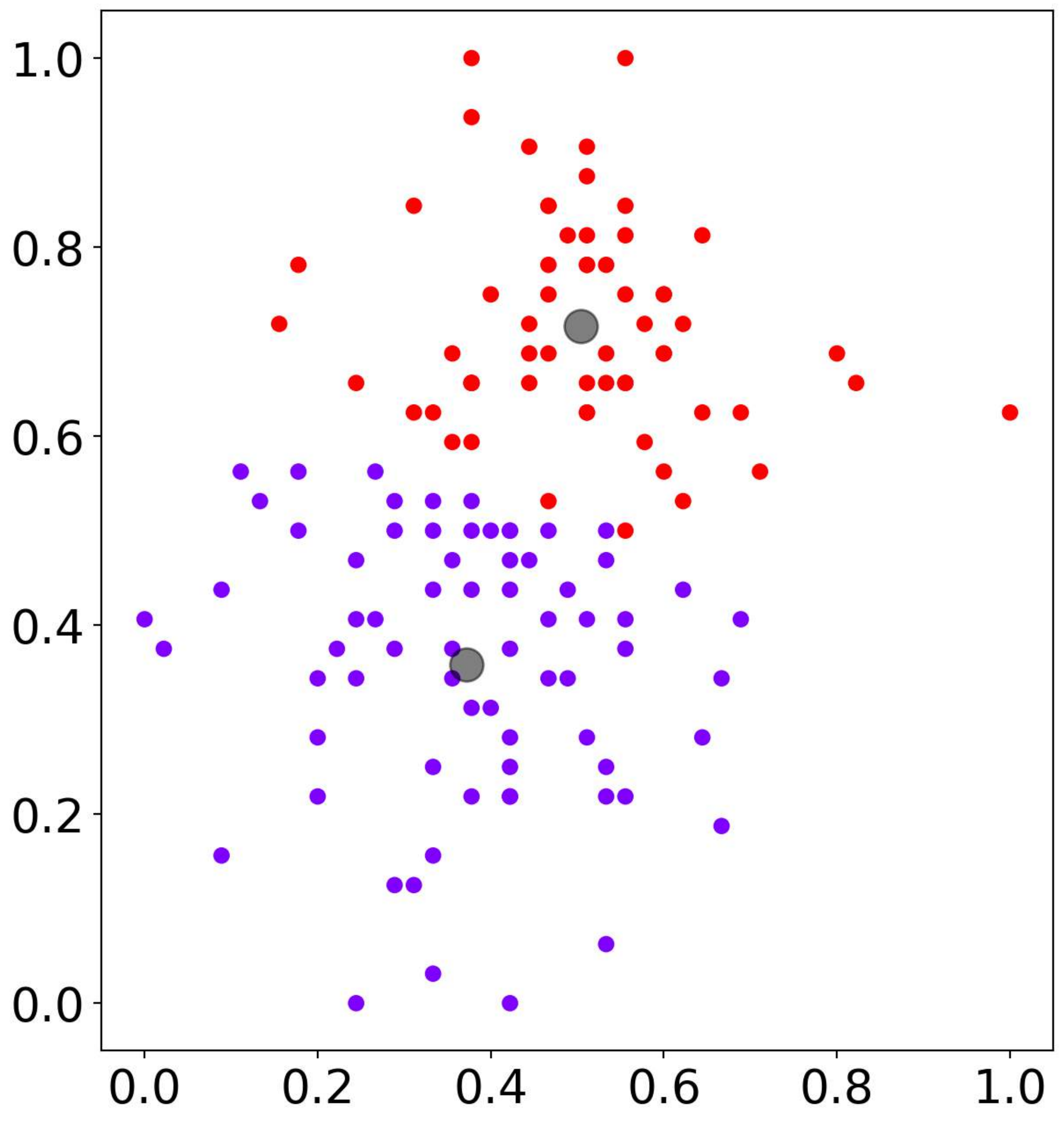
DATASOCIETY: © 2021

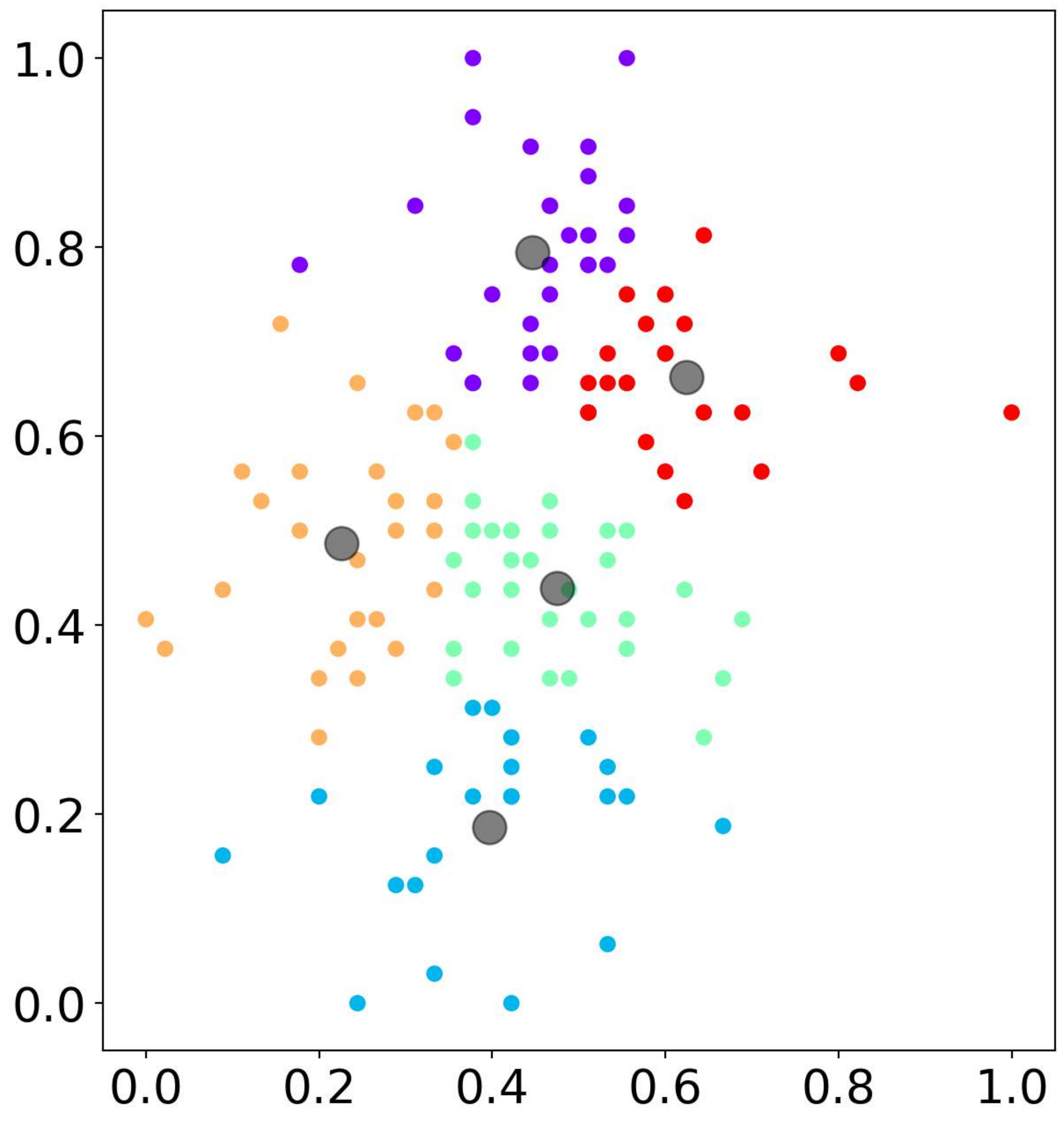
Plot k = 2 vs. k = 5

- 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





DATASOCIETY: © 2021

Inspect clusters

- We should always inspect our clusters when we work with `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         3
1       96.7          71         3
2       96.9          74         3
3       97.0          80         3
4       97.1          73         3
```

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         98.307143   82.428571
1         98.086364   62.954545
2         98.436667   71.066667
3         97.315385   72.576923
4         99.108333   78.208333
```

DATASOCIETY: © 2021

Inspect clusters

- Through these new found 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

In the chat box, answer the following questions:

- **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

DATASOCIETY: © 2021

Exercise 2

DATASOCIETY: © 2021

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 and silhouette methods and visualize	✔
Inspect the results of clustering the dataset using the optimal k	✔
Discuss common pitfalls of clustering	✔

DATASOCIETY: © 2021

Summary

Today we learned about

- the characteristics of supervised and unsupervised machine learning
    - clustering and its applications
    - using k-means for clustering

In the next class we will cover

- classification and its use cases
    - summary and applications of the knn algorithm
    - implementation of the knn algorithm on training data

Congratulations on completing this module!

DATASOCIETY: © 2021