DATA SOCIETY®

The premiere data science training for professionals

"If you can't explain it simply, you don't understand it well enough."

- Albert Einstein

Today's Objectives

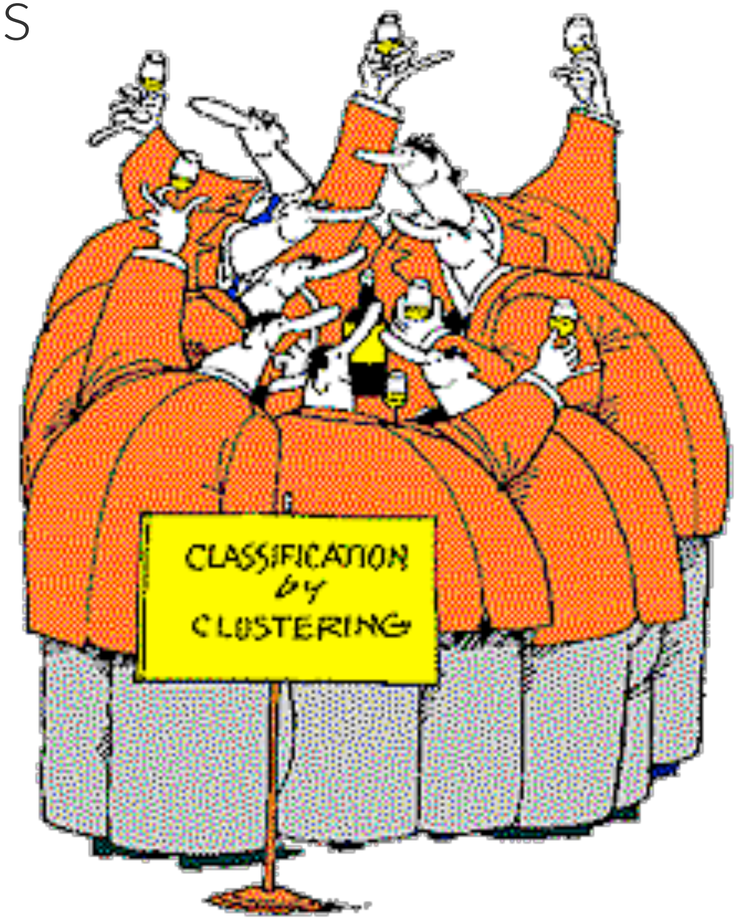
1. Understand what clustering methods can and can't achieve
2. Understand which clustering method is most appropriate for a given problem
3. Use Python's Scikit-learn library to implement clustering methods
4. Apply the data science method to clustering problems
5. Use LinkedIn to analyze a person's network
6. Learn how to get Twitter data and extract trends and sentiment

What is clustering?

1. Technique for finding similarity between groups

2. Synonymous with unsupervised learning
   - Not the only unsupervised learning algorithm

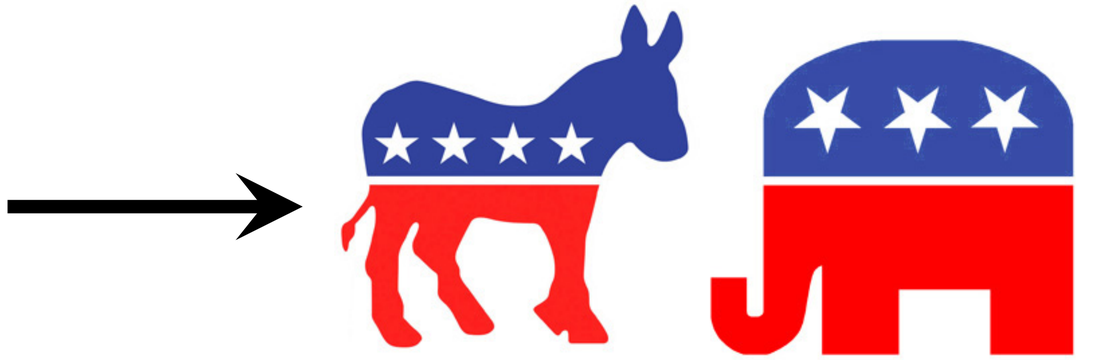
3. Similarity needs to be defined
   - Will depend on attributes of data
   - Usually a distance metric



Supervised machine learning

Pattern discovery when inputs (x) and outputs (y) are known

Input *x*:
Voter



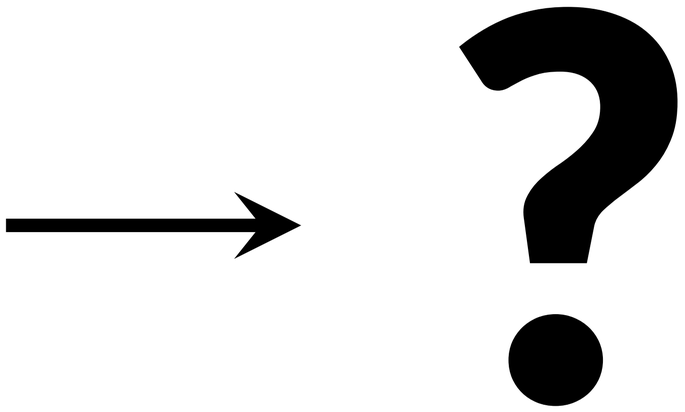
Output *y*:
Political affiliation

Examples: Classification and regression are supervised machine learning

Unsupervised machine learning

The data inputs *(x)* have no target outputs *(y)*

Input *x*:
Voter



Output *y*:
Not given
(to be discovered)

We want to impose structure on the inputs *(x)* to say something meaningful about the data

Clustering vs. classification

	Clustering	Classification
Other terminology	• Exploratory data analysis (EDA) • **Unsupervised** machine learning technique	• **Supervised** machine learning technique
Objective	• **Discover and form new groups** or categories of data	• **Assign data points to known groups** or categories
Methods	• Discover similarity among data points based on attributes • Can use distance between data points to find patterns and determine similarity • Can use distance between clusters to find patterns across groups of data points	• Probabilities of events occurring
Applications	• Discovering patterns in data	• Organizing new data based on known patterns

Clustering vs. classification

	Clustering	Classification
Real world applications	• Identify what types of people should receive credit cards • Create subgroups of social networks based on specific criteria • Predict voting patterns • More accurate real estate pricing • Analyze brand perception • Analyze political sentiment • Detect patterns in spread of disease	• Predict whether or not a customer is pregnant based on shopping patterns • Determine the likelihood of a customer's default on a loan based on social media and spending patterns • Identify an ailment based on symptoms • Predict buying habits based on shopping behavior

Discussion time

What attributes would you use to group your clients/stakeholders?

Examples of clustering

Customer Segmentation

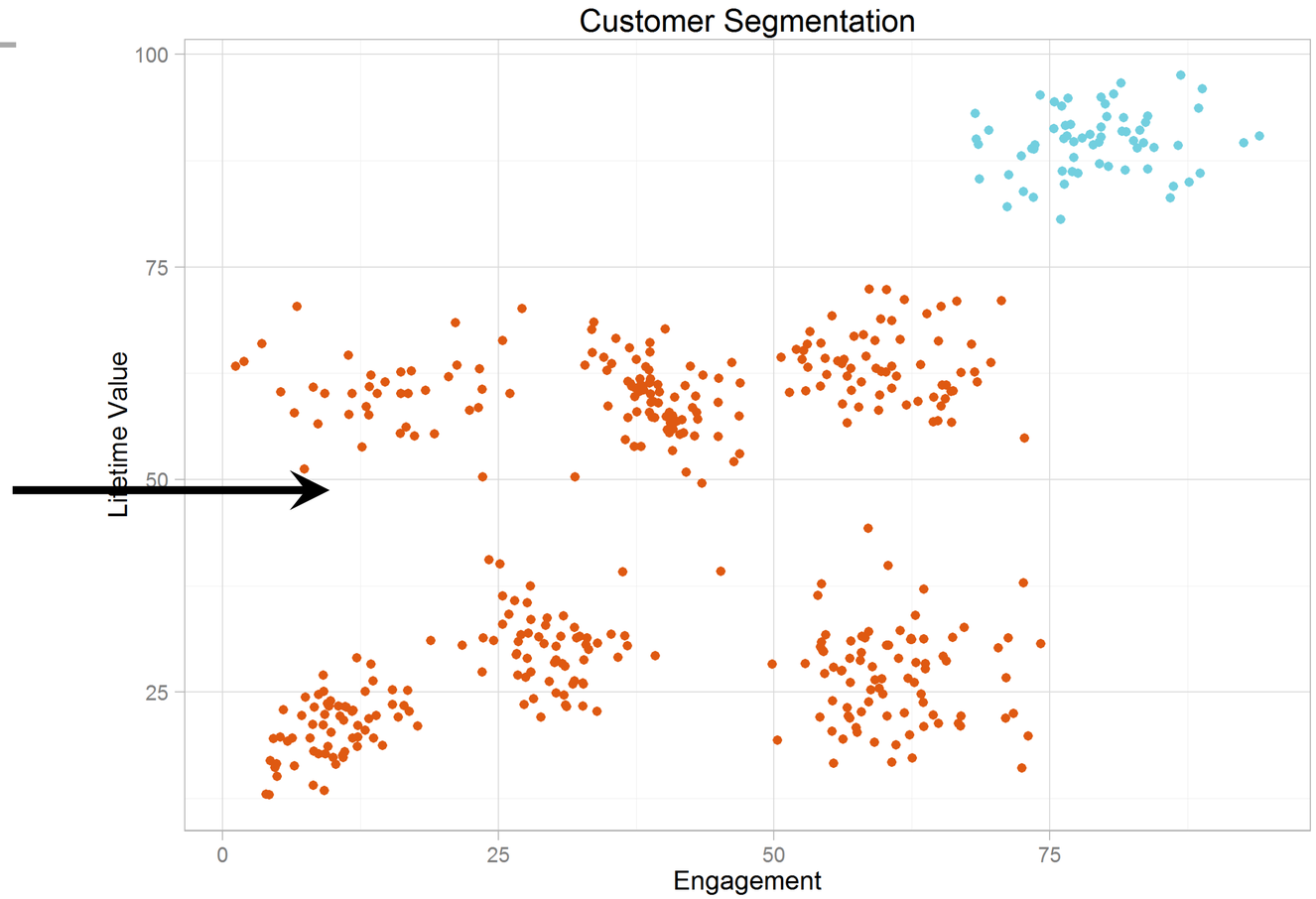
- Lifetime value vs. engagement
  - Engagement = social media, other interaction
  - Lifetime value = total $ spent
- Which customers should you target?
- Which are the most/least valuable?

Note: this data set is a sample set to illustrate the concept, not real data

What's the best clustering?

Cluster Formation

- Best value added?

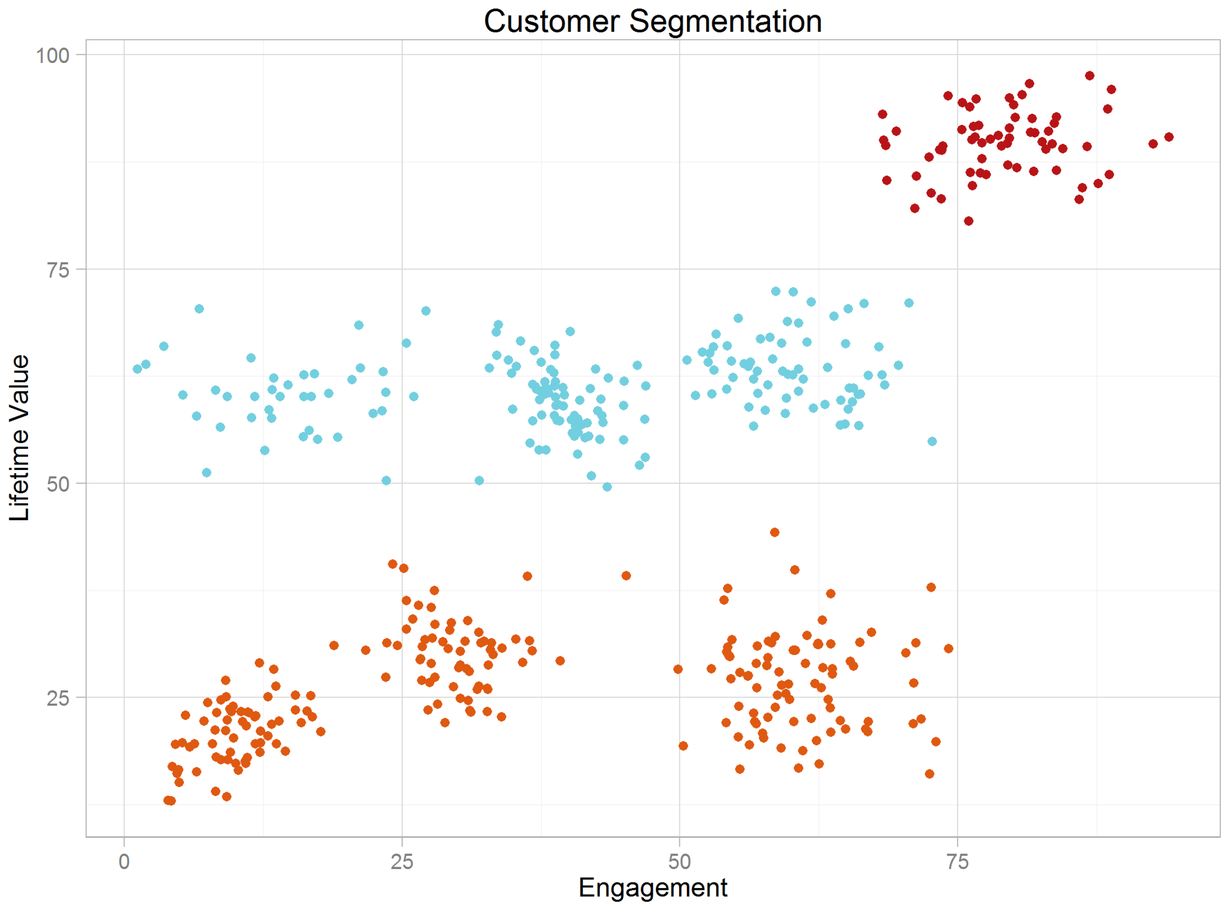


*Note: **Clusters are not color dependent** colors are simply for visualizing different ways of clustering*

What's the best clustering?

Cluster Formation

- Best value added?
- Clustering by **lifetime value**?



What's the best clustering?

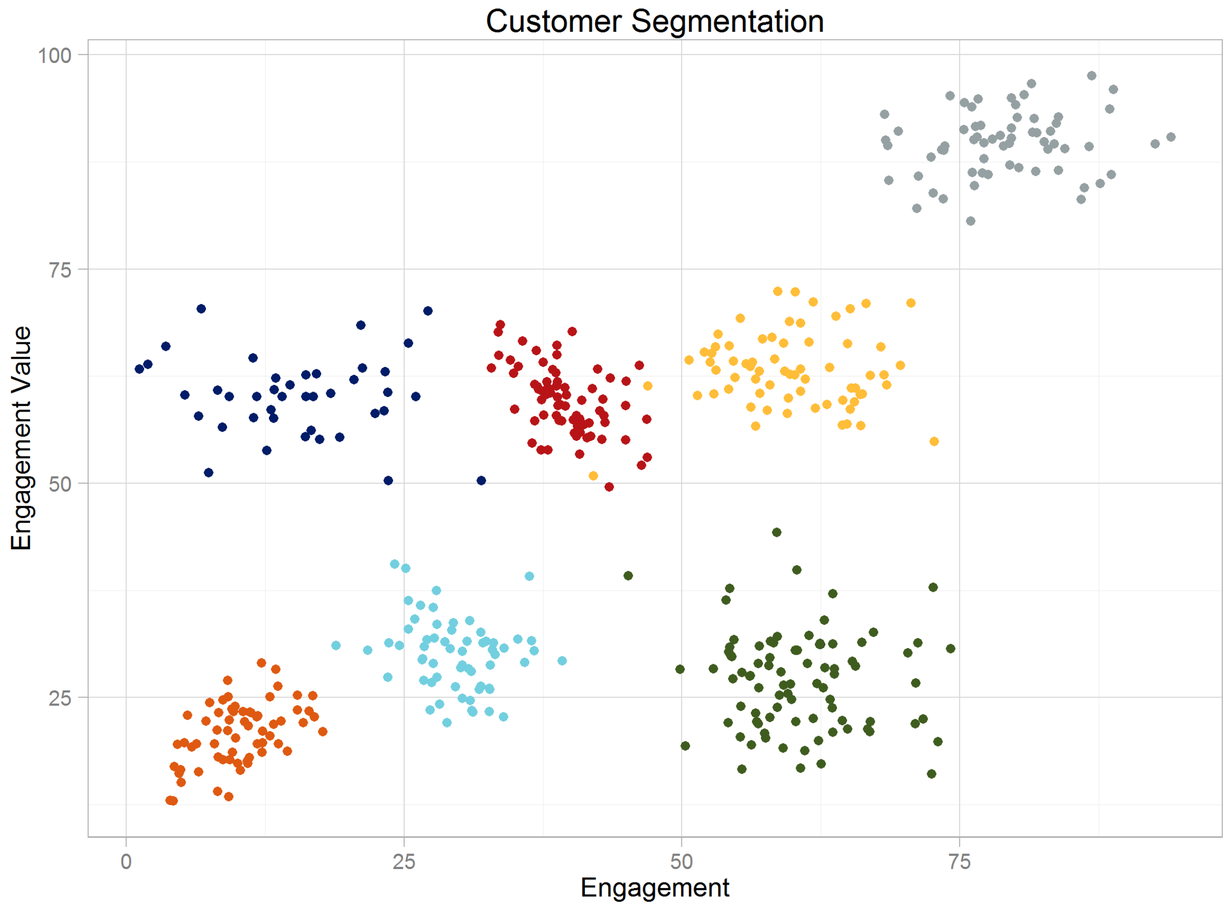
Cluster Formation

- Best value added?
- Clustering by lifetime value?
- Clustering by **engagement**?

What's the best clustering?

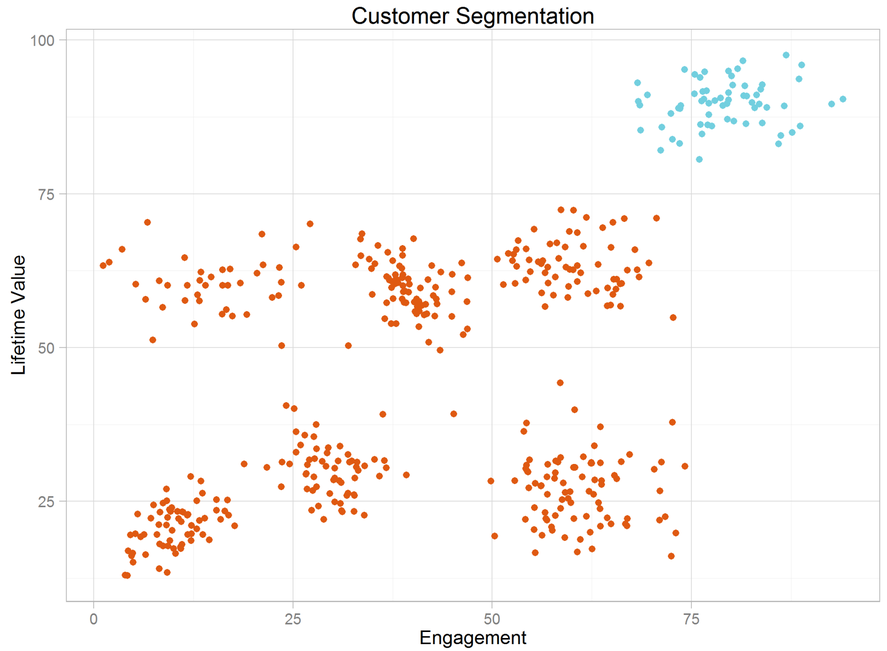
Cluster Formation

- Best value added?
- Clustering by lifetime value?
- Clustering by engagement?
- Clustering by **engagement x lifetime**?



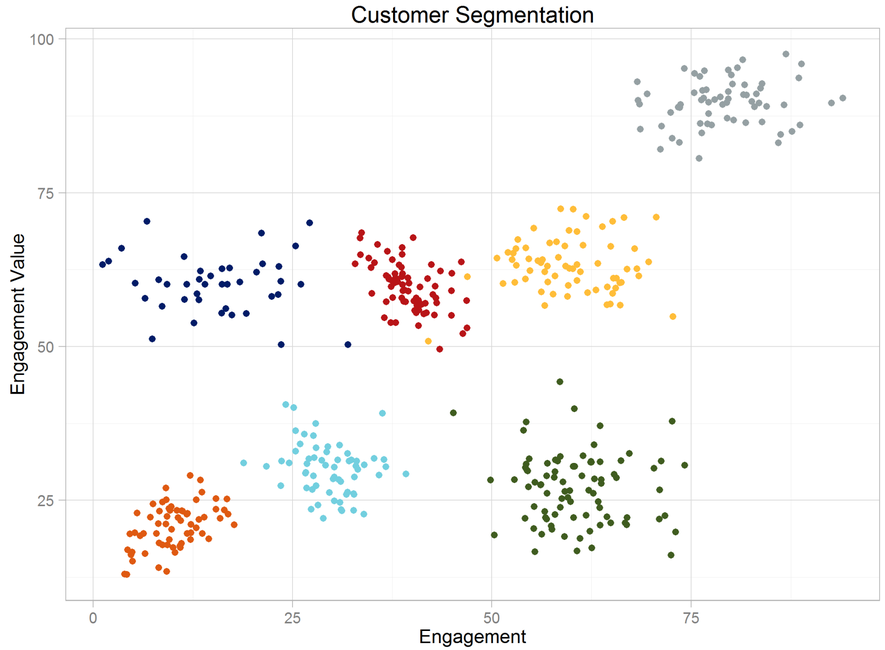
There is no “best” cluster

- Clustering requires user input
- What is “best” is subjective to what you want to achieve



Best value added

Vs.



Engagement x lifetime

Today's syllabus

- We will work with two distance-based clustering algorithms
  1. **k-means** algorithm
  2. Hierarchical algorithms:
     a. **Agglomerative** (AGNES): aggregates data points into groups to form increasingly large clusters
     b. Divisive Analysis (not covered today)

- Identifying when and when not to use clustering
  - If the data has the wrong structure, the clustering algorithm will not work
  - Attributes of data define the appropriate clustering algorithm

Key assumption: data points that are "closer" together are related or similar

Clustering Algorithms for today

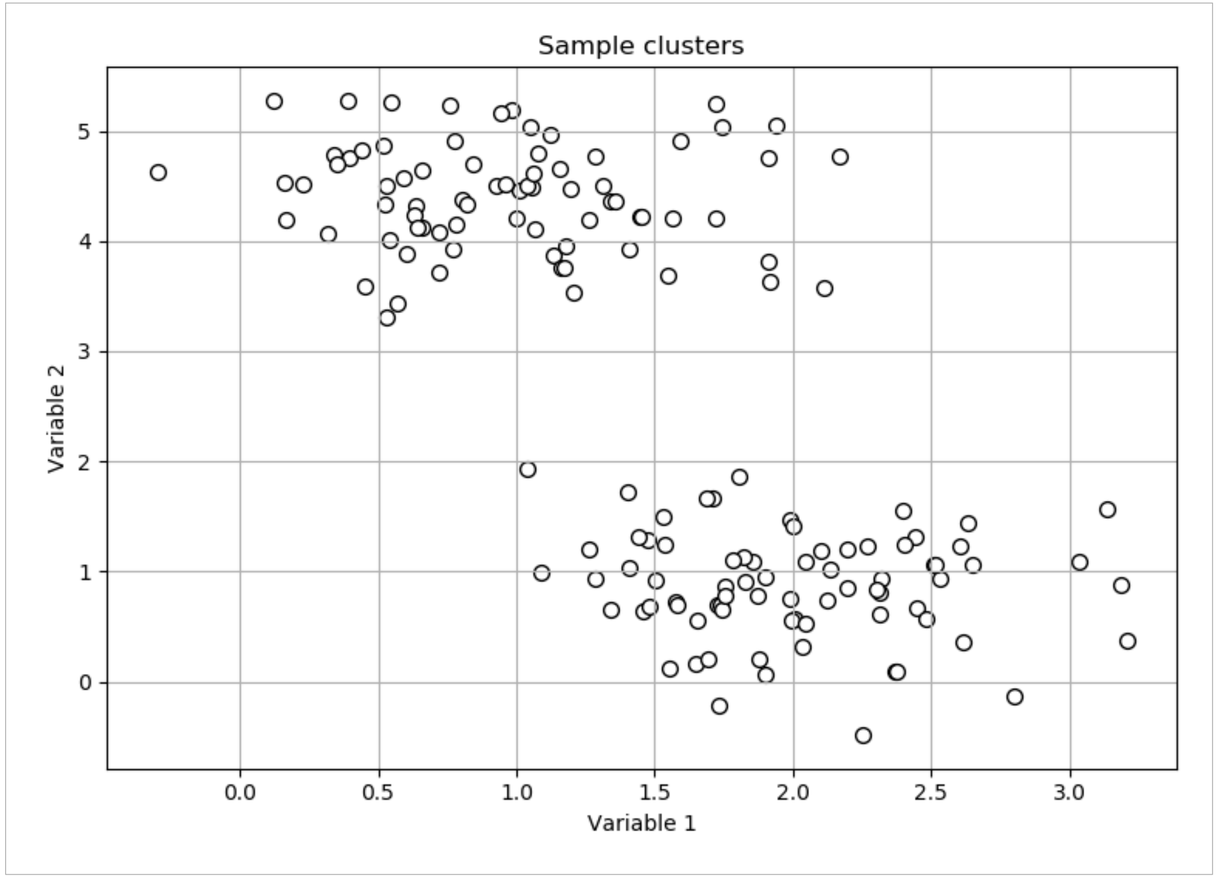
Tool / topic	Question to answer	Real world example
k-means clustering	Is there a pattern? How do we structure unstructured data?	Can we group similar chocolates at the local candy store so we can sell more?
Hierarchical clustering	Is there a pattern? How do we structure unstructured data?	What is the structure of a group of students and their learning habits?

k-means outline

1. What is a centroid?
2. What is intra vs. inter cluster distance?
3. k-means in 4 steps
4. How to determine good clustering
5. How to select the number of clusters
6. When can we use k-means?
7. Clustering Chocolates
8. Properties of clustering algorithms and 4 common problems

What does it mean to “cluster” points?

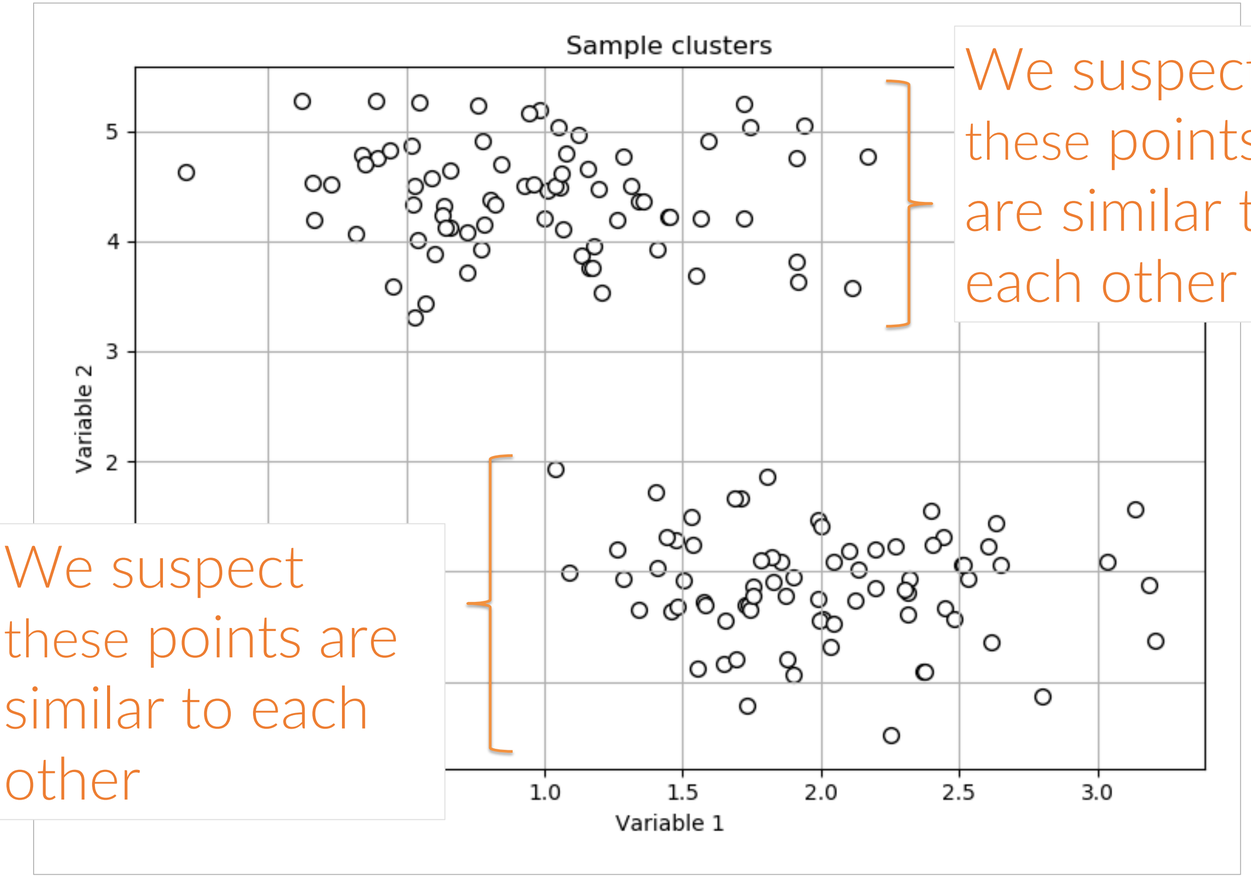
- Suppose we have a data set containing “Variable 1” and “Variable 2”
- By plotting Variable 2 against Variable 1, we may see a pattern
- We would like an algorithm to assign each data point to a group



What does it mean to “cluster” points?

- The underlying assumption is that points that are “close” to each other in the plot must be more similar in real life

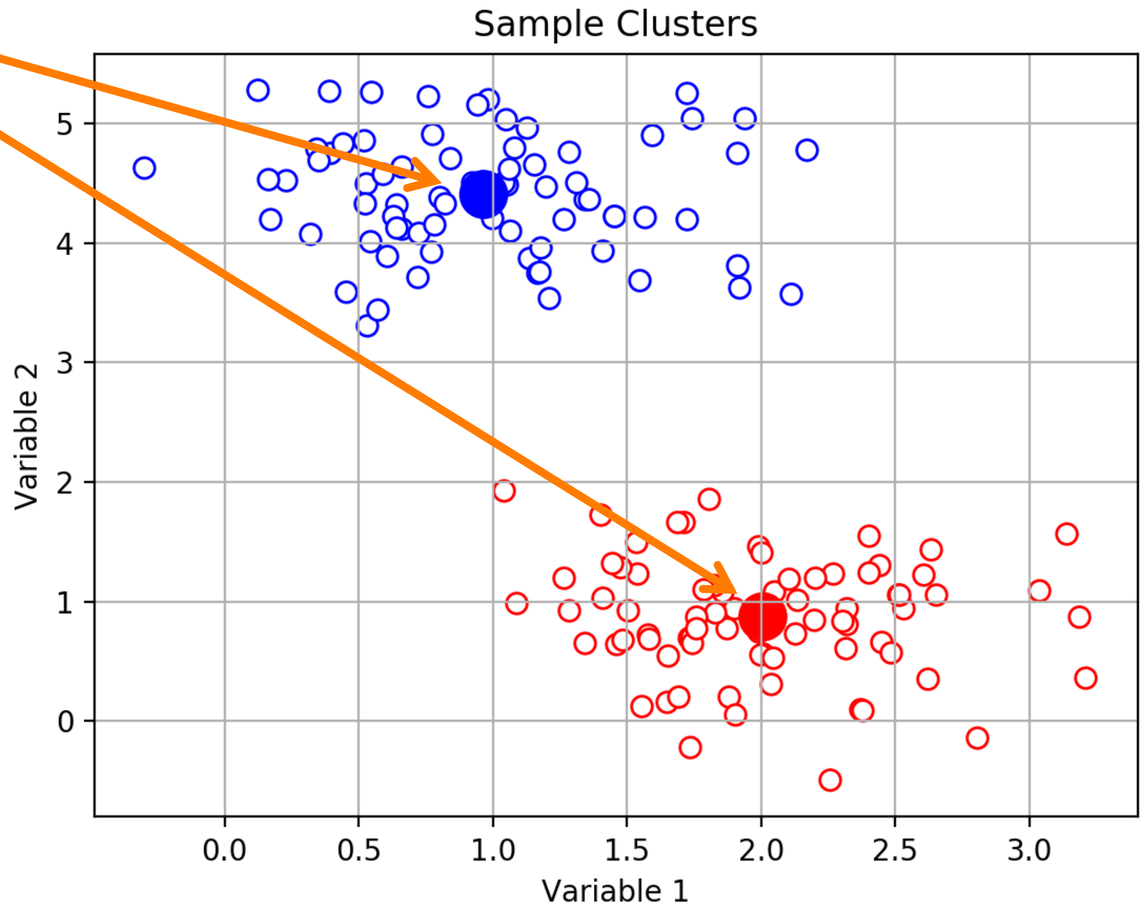
- A clustering algorithm should assign groups in which the members are *more similar to each other* than to members of other groups



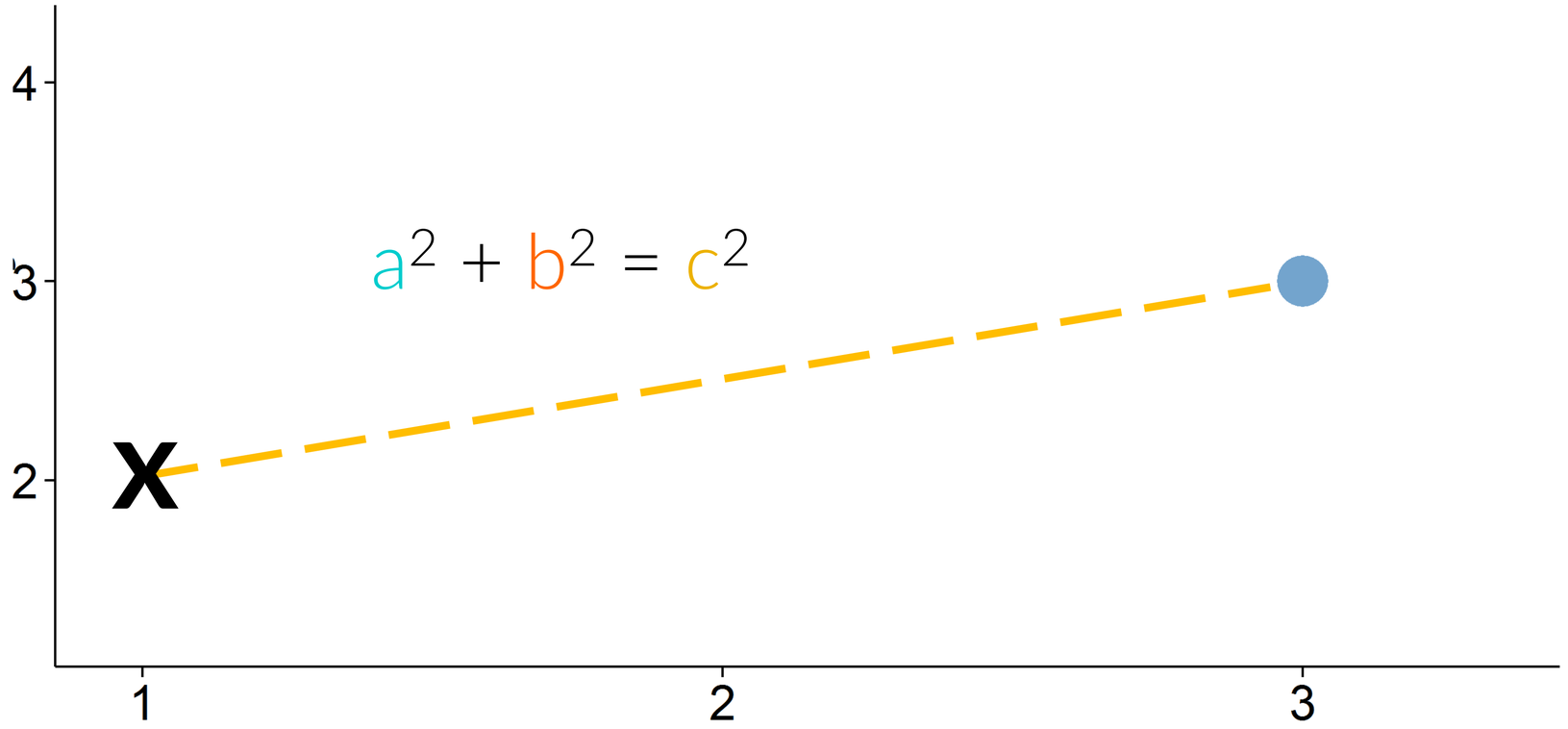
k-means clustering is based on centroids

- k-means seeks to calculate optimal locations for "cluster centroids"
- The centroid is *the average location of all points in the cluster*
- The optimal centroids **should minimize the distance between the centroid and all the data points in the cluster**

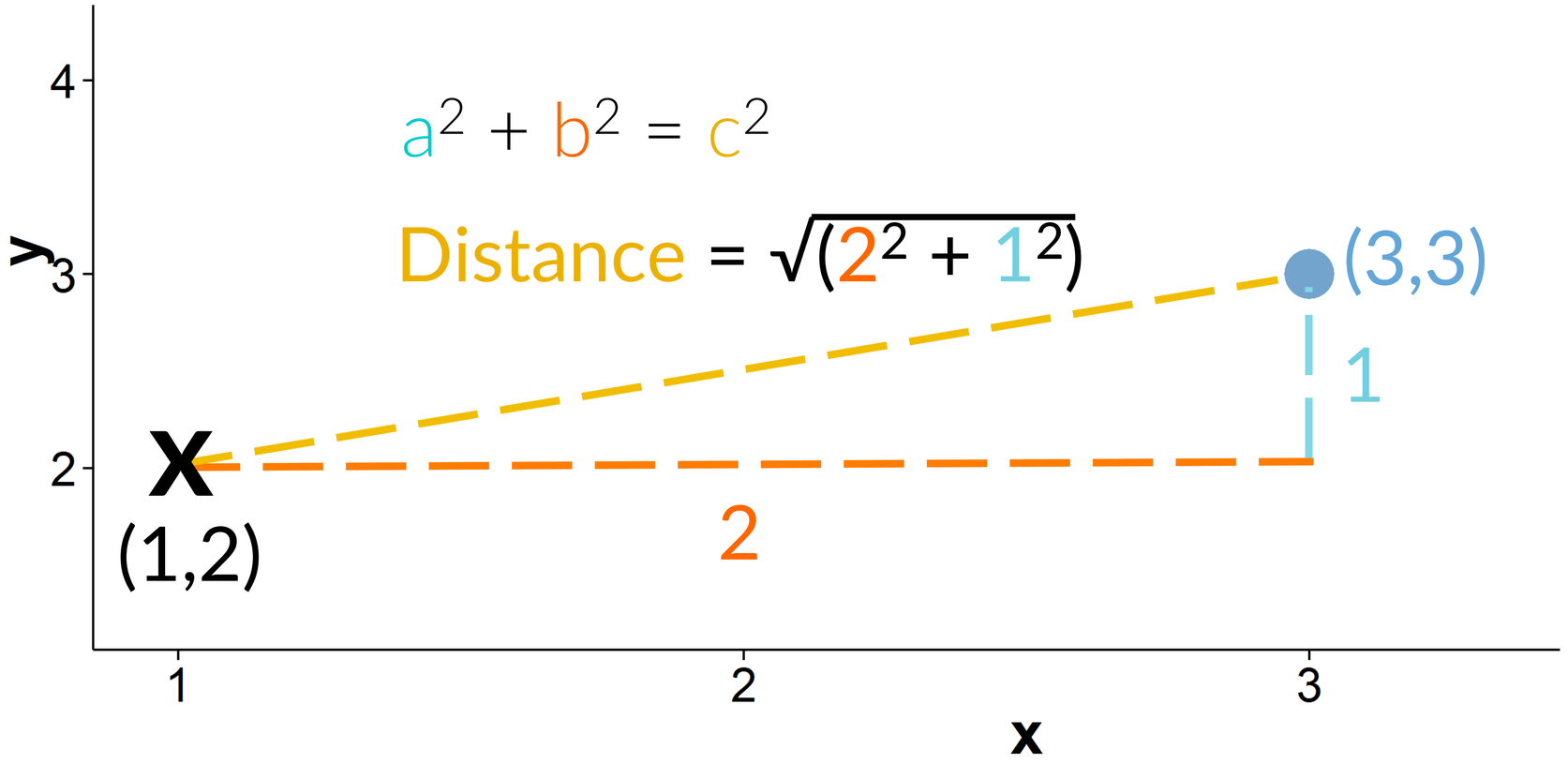
Note: Centroids are generally not existing data points, rather locations in space



Measure distance to assess similarity

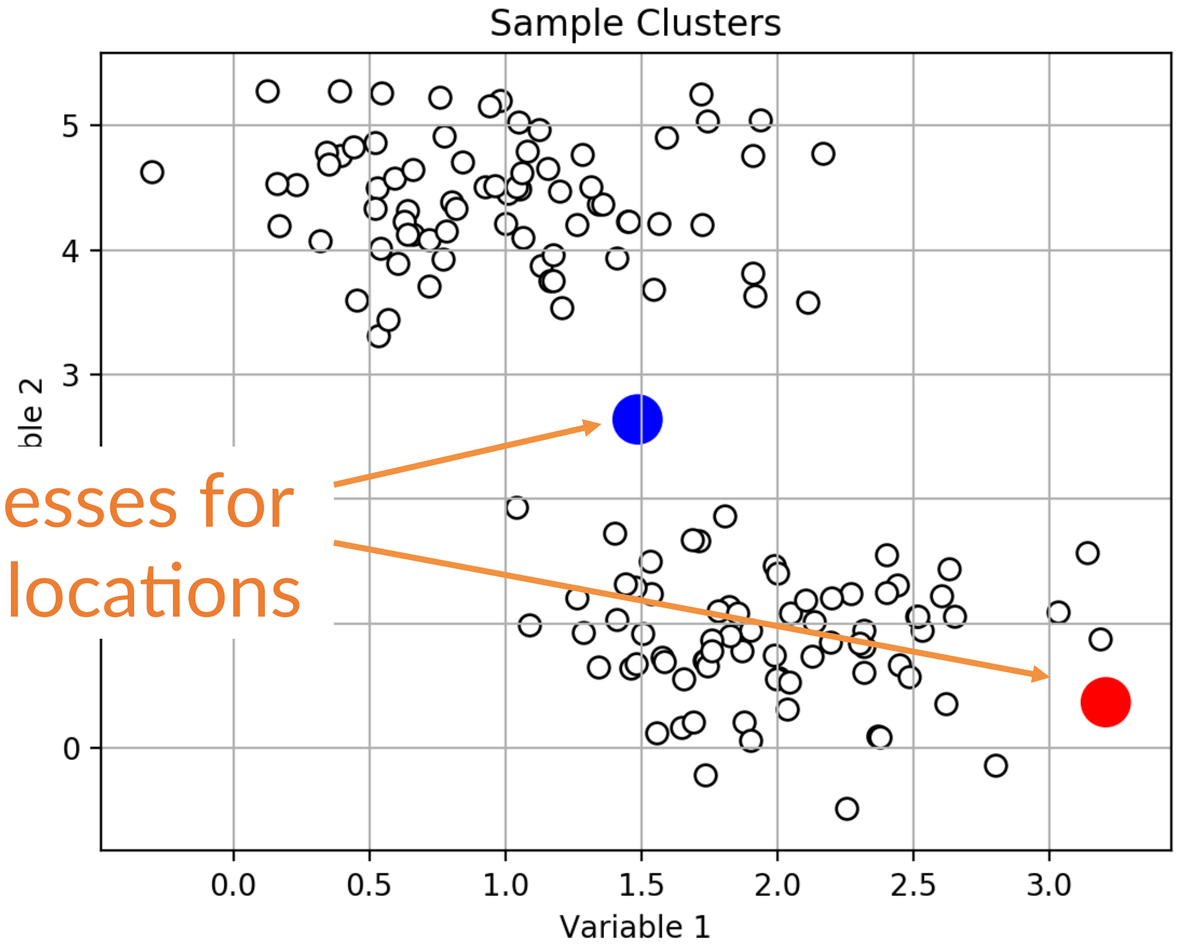


Measure distance to assess similarity



k-means in 4 steps

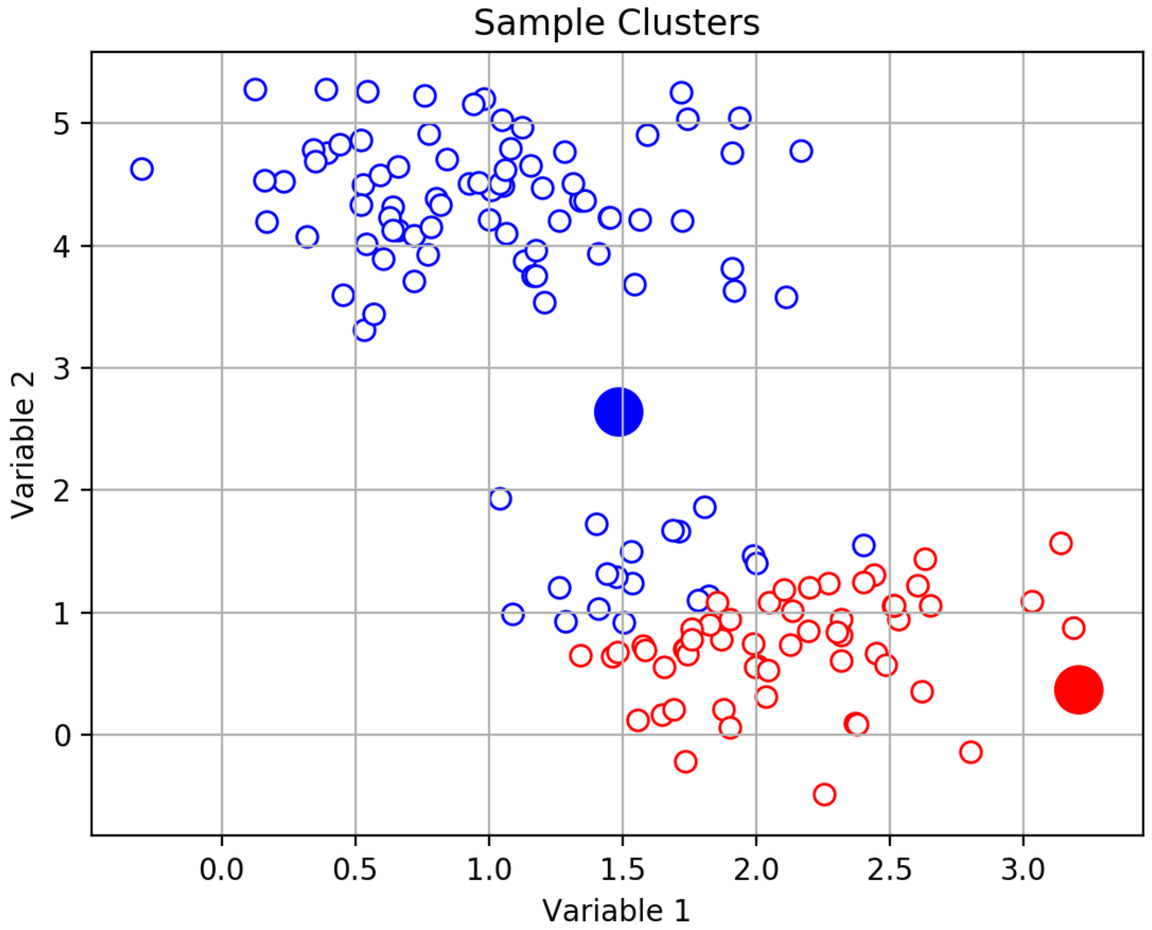
1. Randomly choose *k* data points to be centroids
   - We have chosen k = 2 in this example



k-means in 4 steps

1. Randomly choose k data points to be centroids

2. Assign each point to **closest centroid**

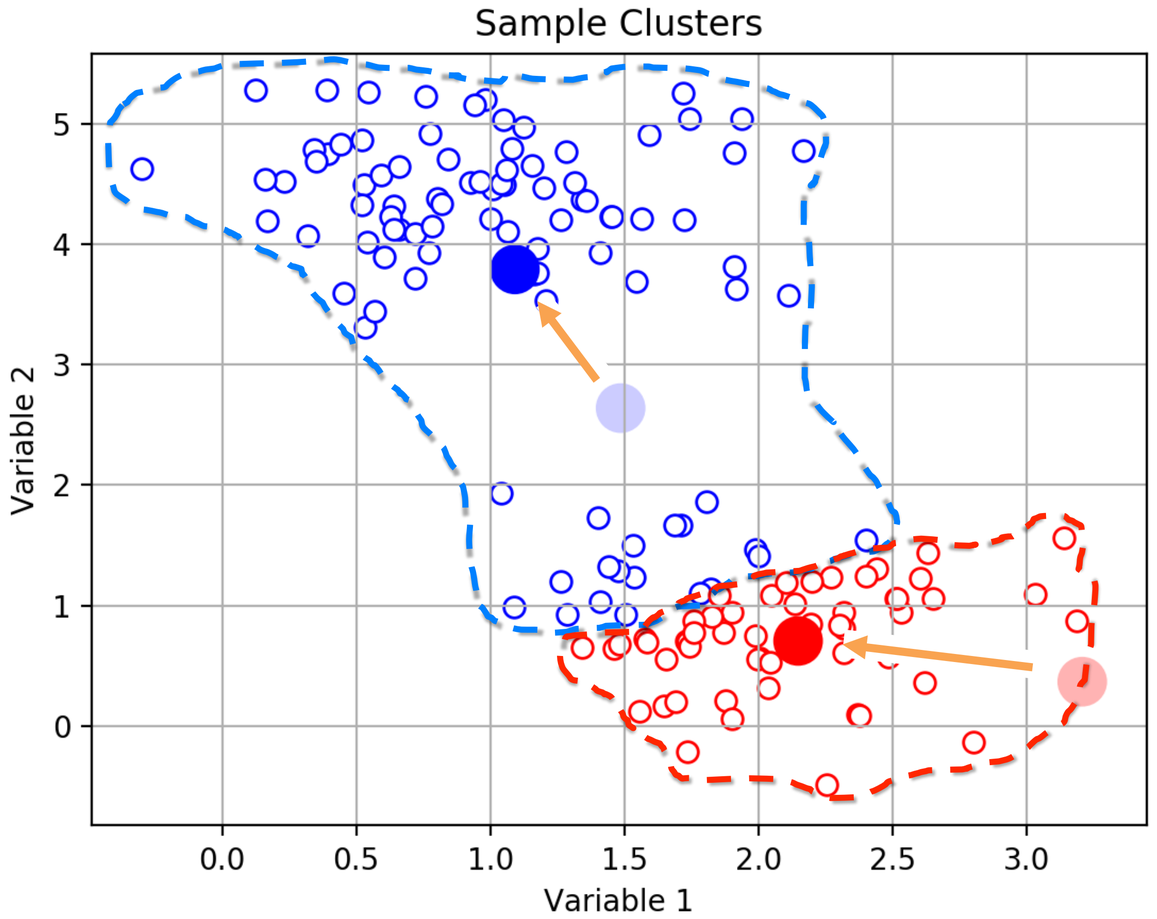


k-means in 4 steps

1. Randomly choose k data points to be centroids

2. Assign each point to closest centroid

3. **Recalculate centroids** based on current cluster membership

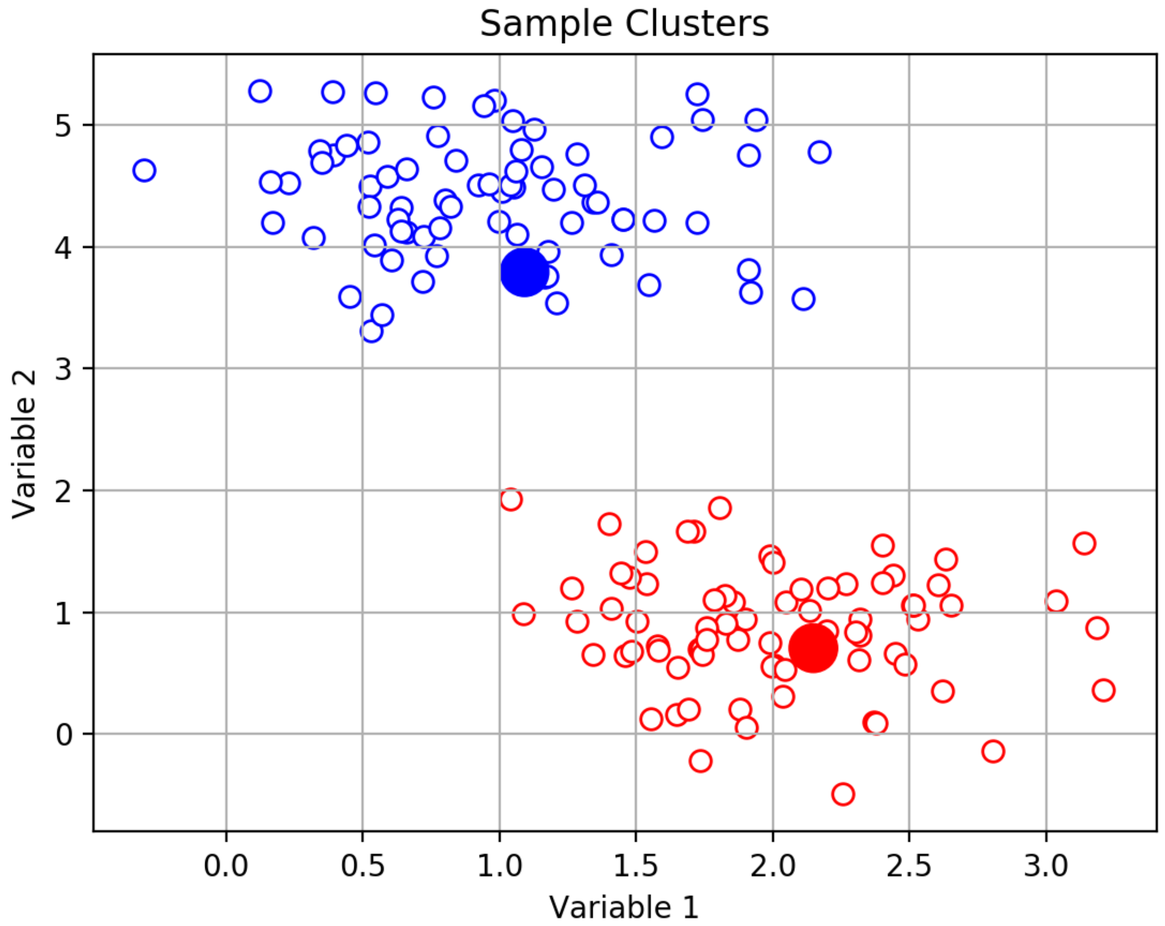


k-means in 4 steps

1. Randomly choose k data points to be centroids

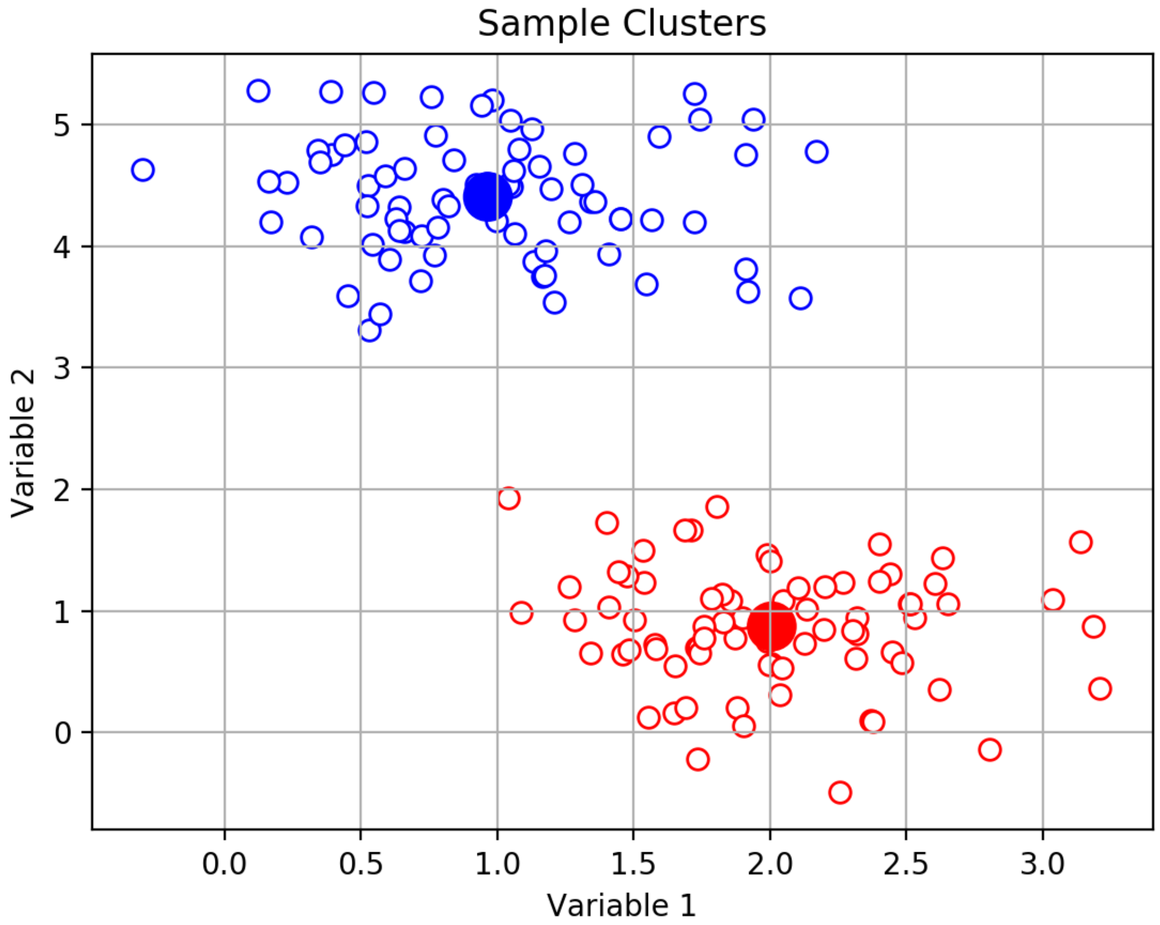
2. Assign each point to closest centroid

3. Recalculate centroids based on current cluster membership (then repeat step 2)



k-means in 4 steps

1. Randomly choose k data points to be centroids
2. Assign each point to closest centroid
3. Recalculate centroids based on current cluster membership
4. **Repeat steps 2-3** with the new centroids until the centroids **don't change anymore**



Step 1: load data

Script

```
# Load needed packages and modules
from sklearn import datasets
import numpy as np
import os
import matplotlib.pyplot as plt

# How to create dummy data for clustering
from sklearn.datasets import make_blobs
sample_data, y = make_blobs(n_samples = 150,
                            n_features = 2,
                            centers = 2,
                            cluster_std = 0.5,
                            shuffle = True,
                            random_state = 0)
```

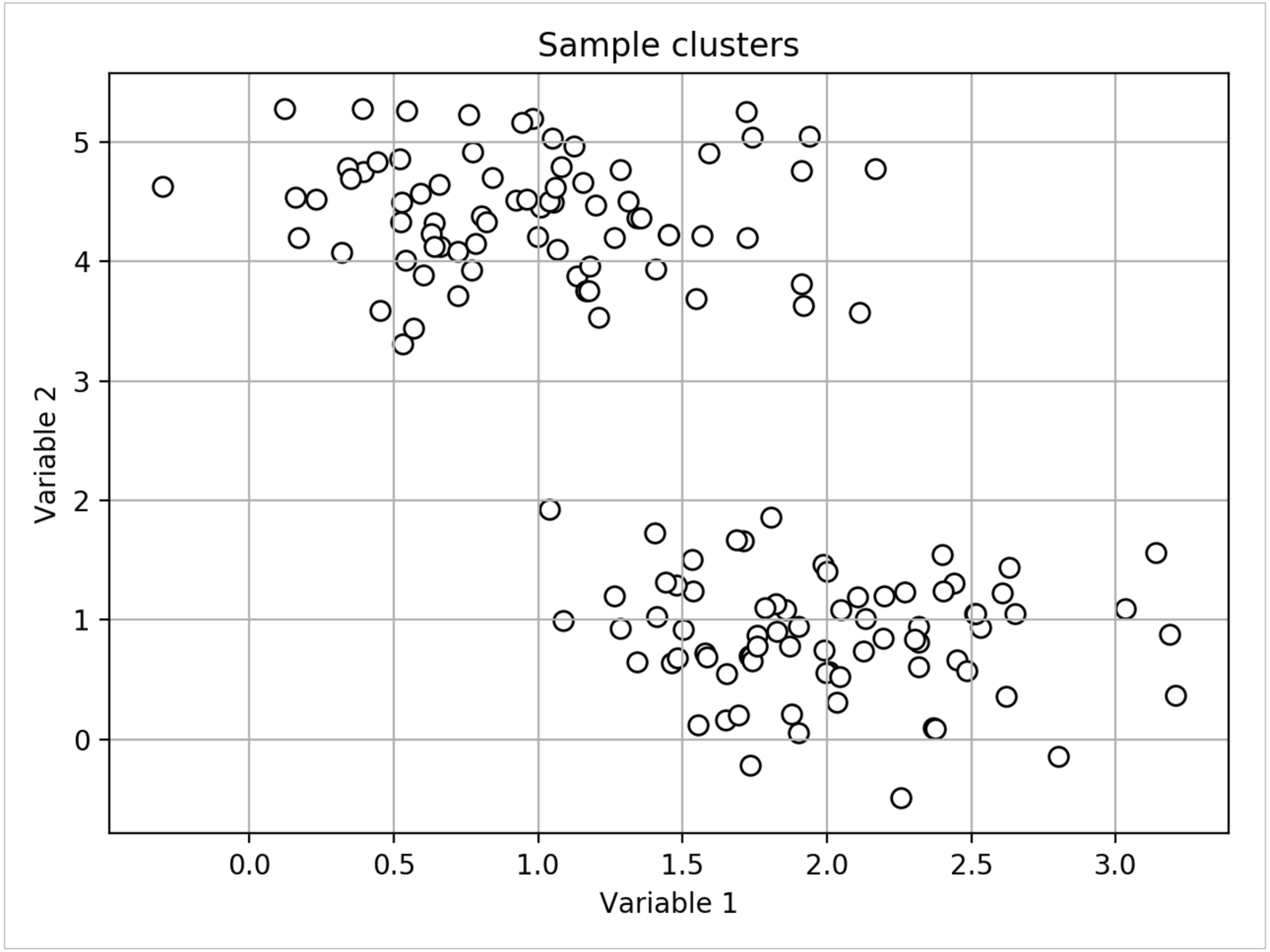
Step 2: plot data as is

Script

```
# Sample_data is a 2D array of the coordinates of our data
type(sample_data)
sample_data.shape
sample_data[0:5,:]

# Plot of sample clusters
plt.scatter(sample_data[:, 0], sample_data[:, 1],
            c = 'white', marker = 'o',
            edgecolor = 'black', s = 50)

plt.grid()
plt.title("Sample clusters")
plt.xlabel("Variable 1")
plt.ylabel("Variable 2")
plt.tight_layout()
plt.show()

plt.savefig('plots/sample_data/points_bw.png', dpi = 200)
```

Make sure to include edgecolor
otherwise the white circles won't appear

Step 2: plot data as is



Step 2: run k-means

- Import functions from package: `sklearn`
- Library: `from sklearn.cluster import KMeans`

```
KMeans(n_clusters, init, n_init, max_iter, tol, random_state)
```

KMeans Inputs:

`n_clusters:` specify the number of clusters

`n_init:` number of times we run the kmeans algorithm, choosing the final model with the least SSE

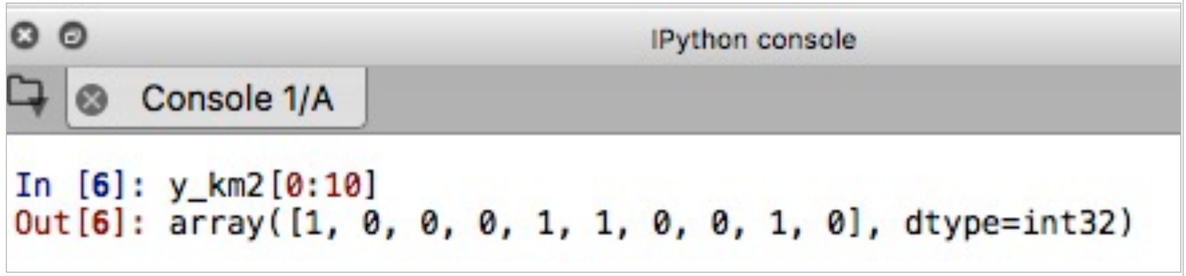
`max_iter:` specify the maximum number of iterations for each single run

`tol:` because the kmeans algorithm will continue iterating until the max_iter is reached, we set the tol parameter this is the parameter that controls the tolerance with regard to the changes in the WSS to declare convergence.

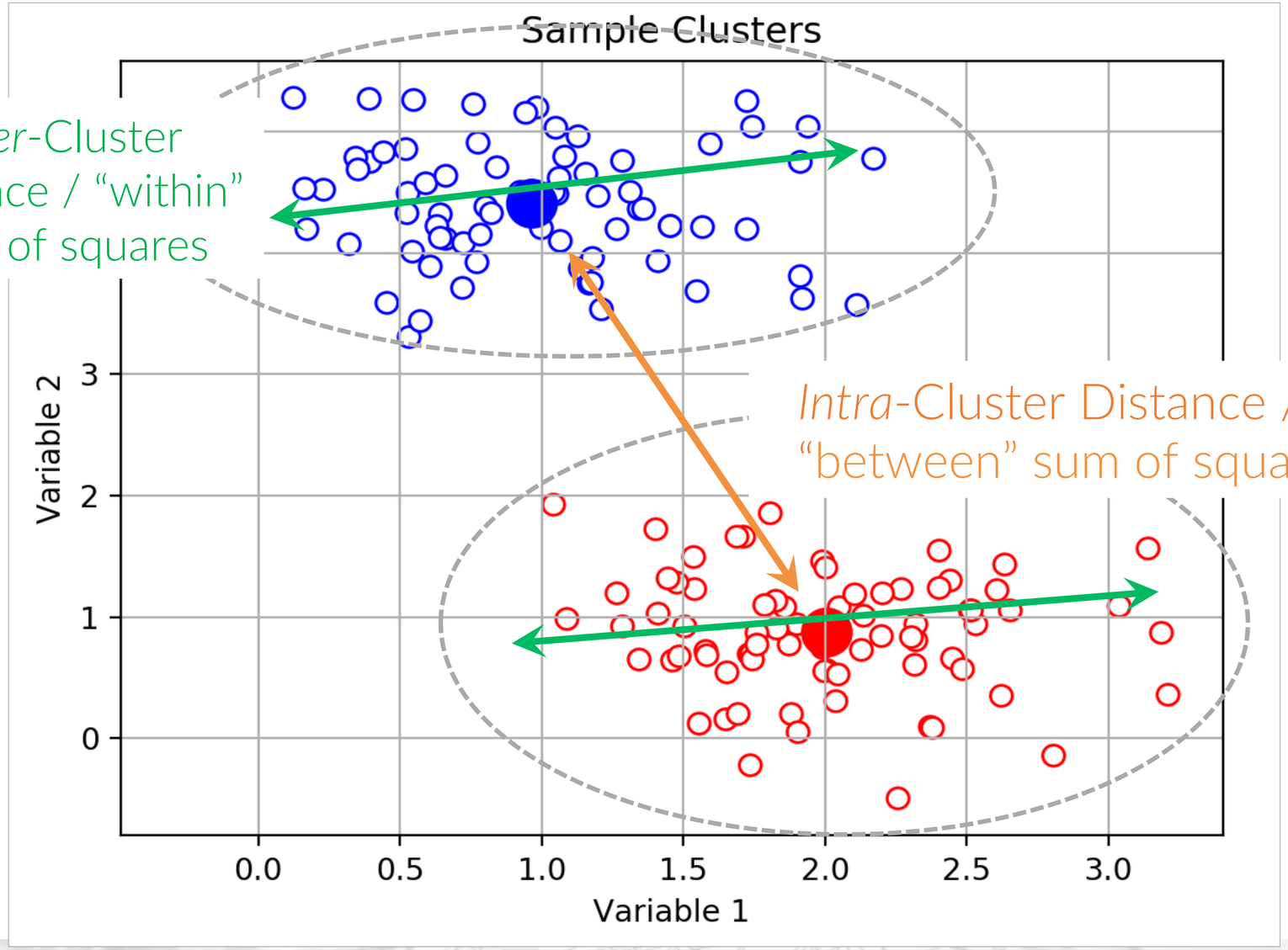
Step 2: run k-means

Script

```
# Import module KMeans from sklearn.cluster
from sklearn.cluster import KMeans

# Kmeans, with parameters already set
km = KMeans(n_clusters = 2,
            init = 'random',
            n_init = 10,
            max_iter = 300,
            tol = 1e - 04,
            random_state = 0)

# Run ("fit") kmeans on data set sample_data, and categorize
# ("predict") a cluster for each point
y_km = km.fit_predict(sample_data)
type(y_km2)
y_km2.shape
y_km2[0:10]
```



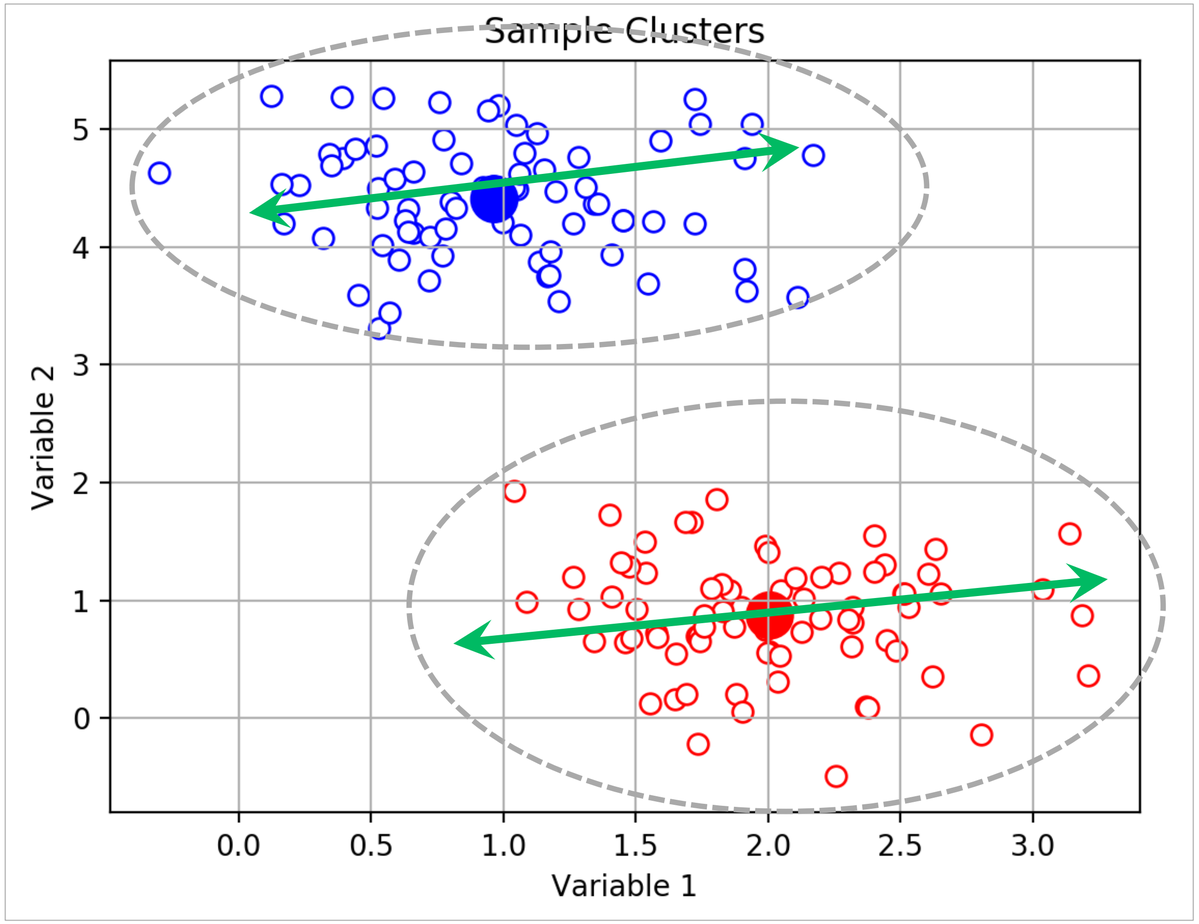
Intra vs. inter-cluster distance



Intra vs. inter-cluster distance

- The goal of the k-means algorithm is to **minimize** the total sum of the *Inter*-Cluster Distance / "within" sum of squares.
- This is also called the *inertia*
- We can get this value from the kmeans variable:

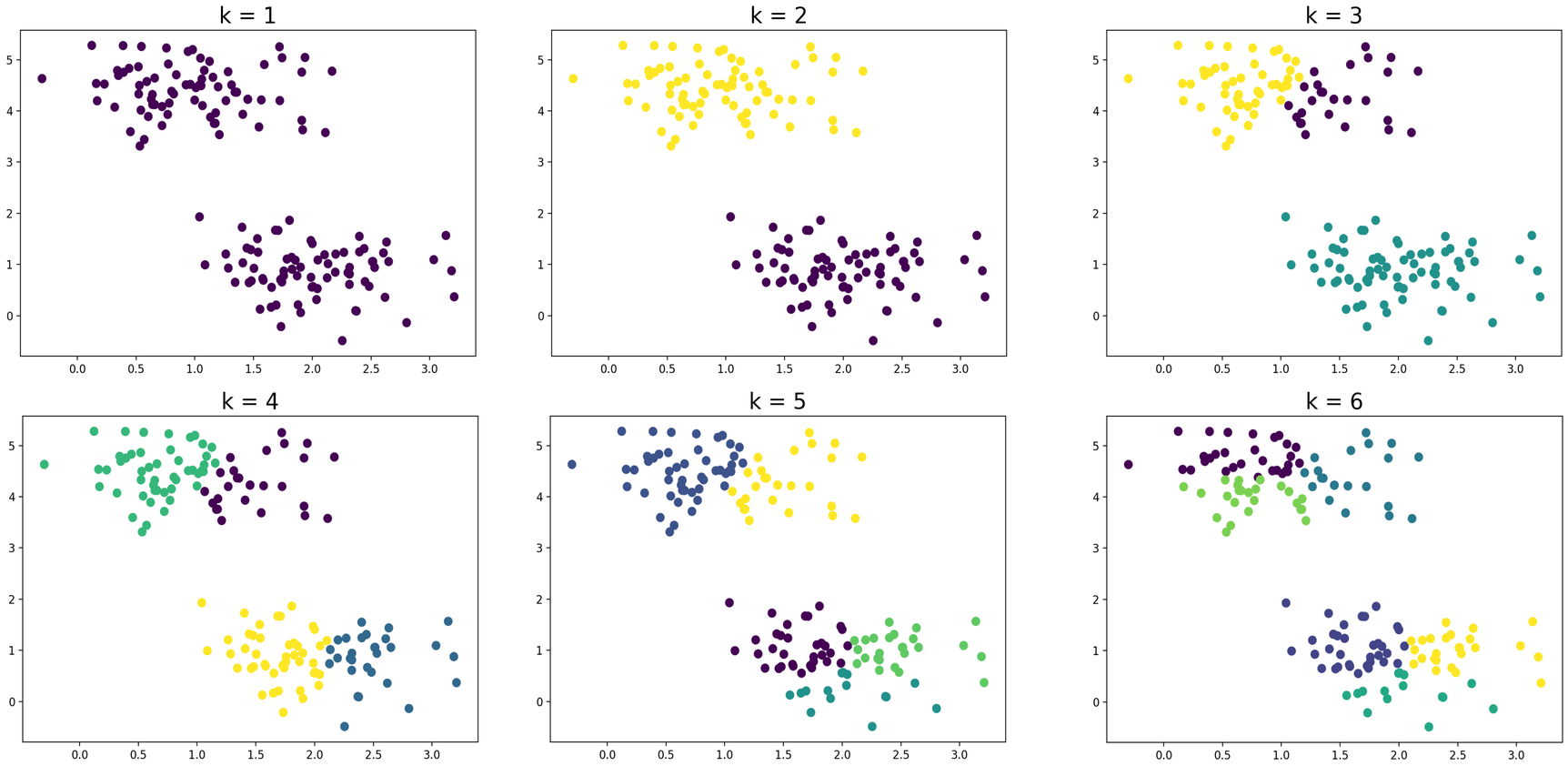
```
In [16]: print('Inertia: %.2f' % km2.inertia_)
Inertia: 73.10
```



How many clusters to make?

- Recall that we must specify a value for k, the number of clusters.
  - How do we know we've chosen k correctly?

Too few clusters



Too many clusters (overfitting)

Step 3: measure distortion

- To measure/quantify the quality of the clustering, we use within cluster SSE (inertia) also referred to as Distortion.
  - SSE is the same as the total sum of squares. We reviewed this earlier, it is one way to measure the goodness of the clusters.
  - One of the values returned by the Kmeans function is `inertia_`, so we can use this
  - To optimize the way we choose the number of k, we can use a plot that illustrates the total SSE or the '% Distortion' by # of clusters.
- Before plotting the clusters, let's make sure we have chosen the right number. We usually would perform this step before running k-means.

Step 3: get best *k* value

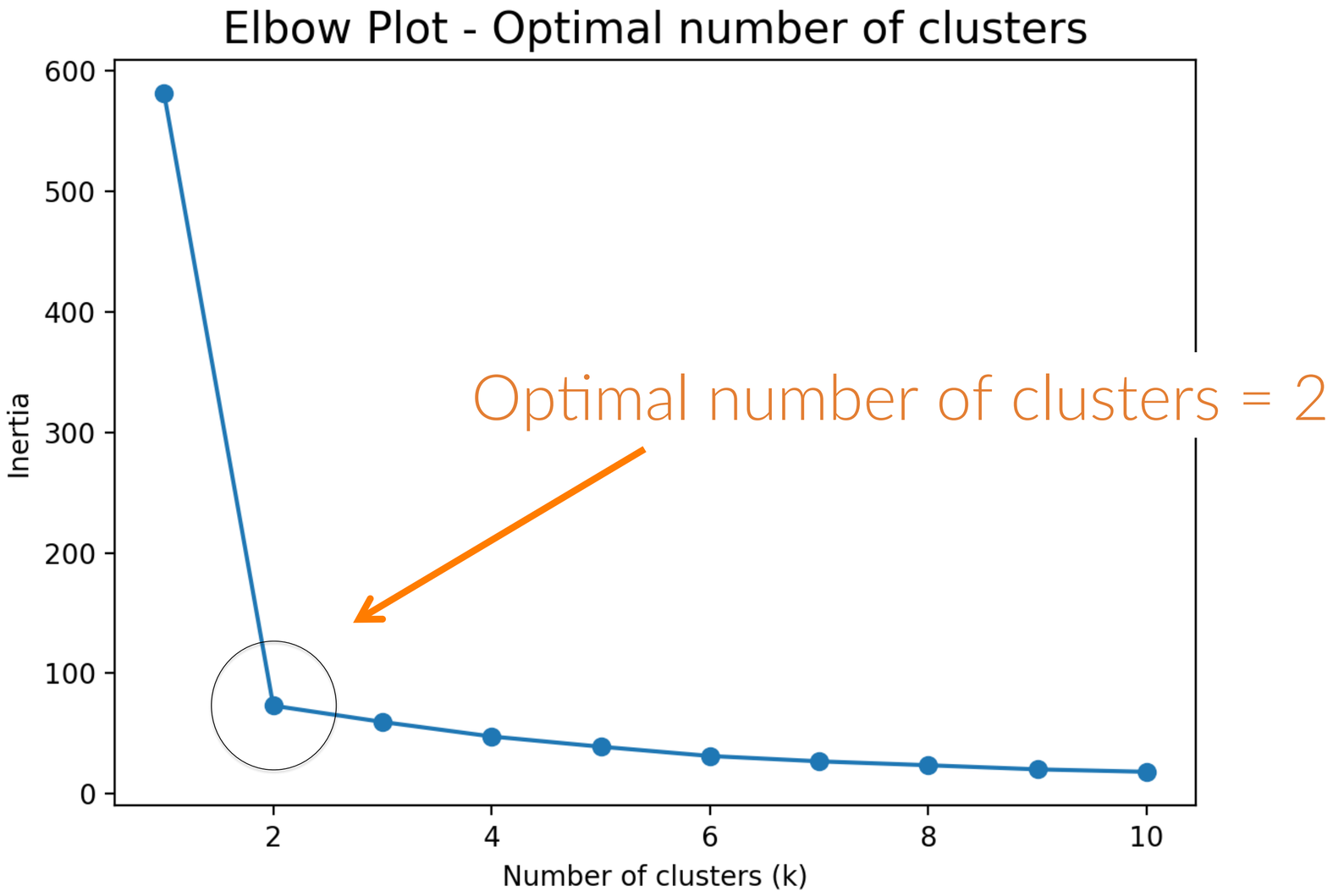
Script

```
inertias = []
for k in range(1, 11):
    km = KMeans(n_clusters = k,
        init = 'k-means++',
        n_init = 10,
        max_iter = 300,
        random_state = 0)
     km.fit(sample_data)
     inertias.append(km.inertia_)

plt.plot(range(1, 11), inertias, marker = 'o')
plt.xlabel('Number of clusters (k)')
plt.ylabel('Inertia')
plt.title('Elbow Plot - Find optimal number of clusters', fontsize = 16)
plt.tight_layout
plt.show()

plt.savefig('plots/sample_data/elbow.png', dpi = 200)
```

Here, we are plotting the % inertia by the number of clusters

Step 3: plot distortion

Plot the inertia for each k and look for the "elbow"



Step 4: visualize plot

Script

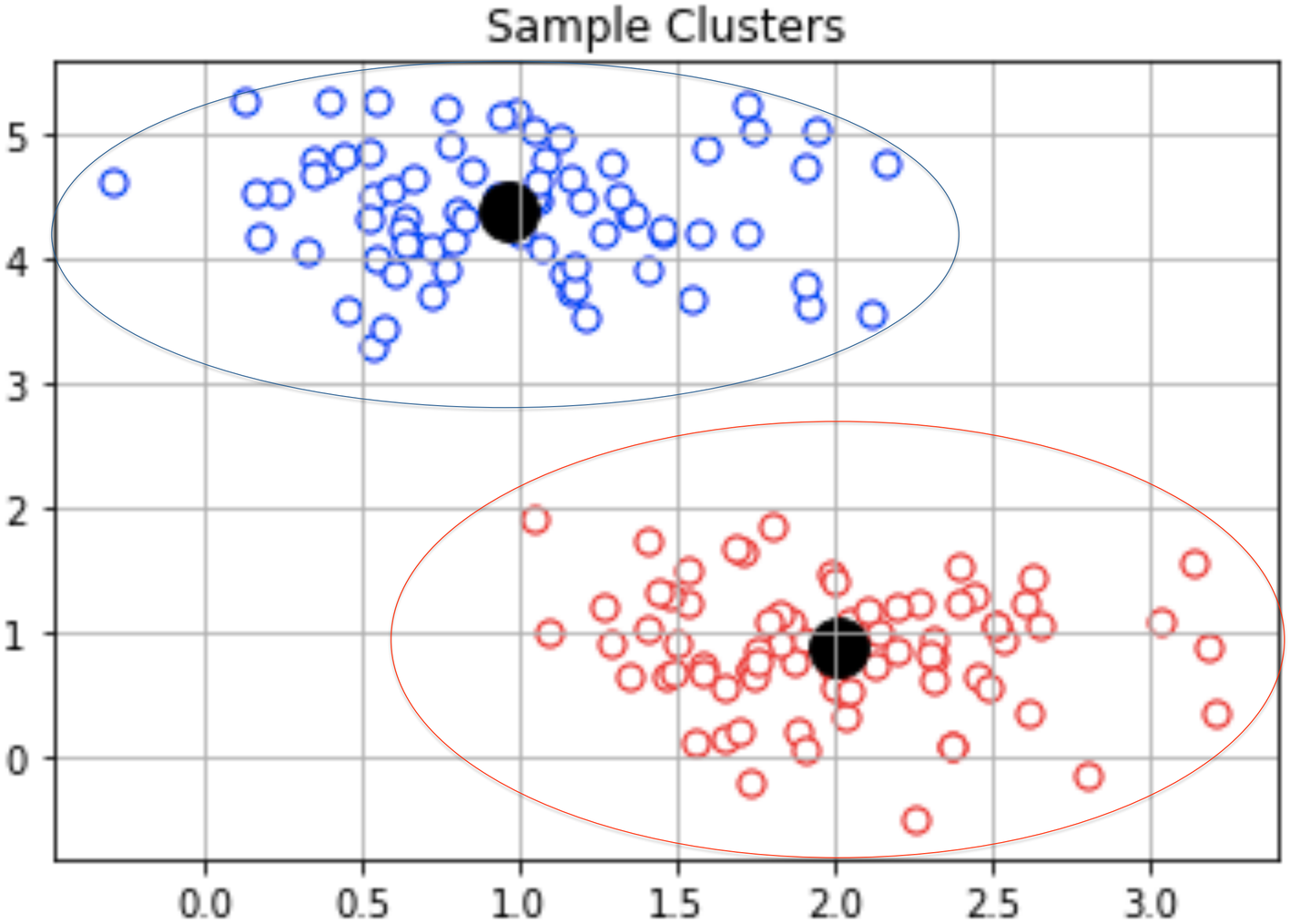
```
# Visualize the clusters 1 & 2, as they are predicted by y_km
cluster_0 = sample_data[y_km2 == 0, :]
cluster_1 = sample_data[y_km2 == 1, :]


plt.scatter(cluster_0[:, 0],
            cluster_0[:, 1],
            s = 50,
            c = 'white',
            marker = 'o',
            edgecolors = 'blue',
            label = 'cluster 1')
```

Step 4: visualize plot

Script

```
plt.scatter(cluster_0[:, 0],
            cluster_0[:, 1],
            s = 50,
            c = 'white',
            marker = 'o',
            edgecolors = 'red',
            label = 'cluster 2')
plt.scatter(km.cluster_centers_[:, 0],
             km.cluster_centers_[:, 1],
             s = 250,
             marker = 'o',
             c = 'black',
             label = 'centroids')
plt.grid()
plt.title("Sample Clusters")
plt.xlabel('Variable 1')
plt.ylabel('Variable 2')
plt.savefig('plots/sample_data/kmeans_2.png', dpi = 200)
```

Step 4: visualize plot



Exercise time!

Step 5: measure clustering quality

- Silhouette analysis is another metric that can be used to measure the quality of the clustering. It can also be applied to other clustering methods besides k-means.

- Silhouette analysis can be used as a graphical tool to plot a measure of how tightly grouped the samples in the clusters are.

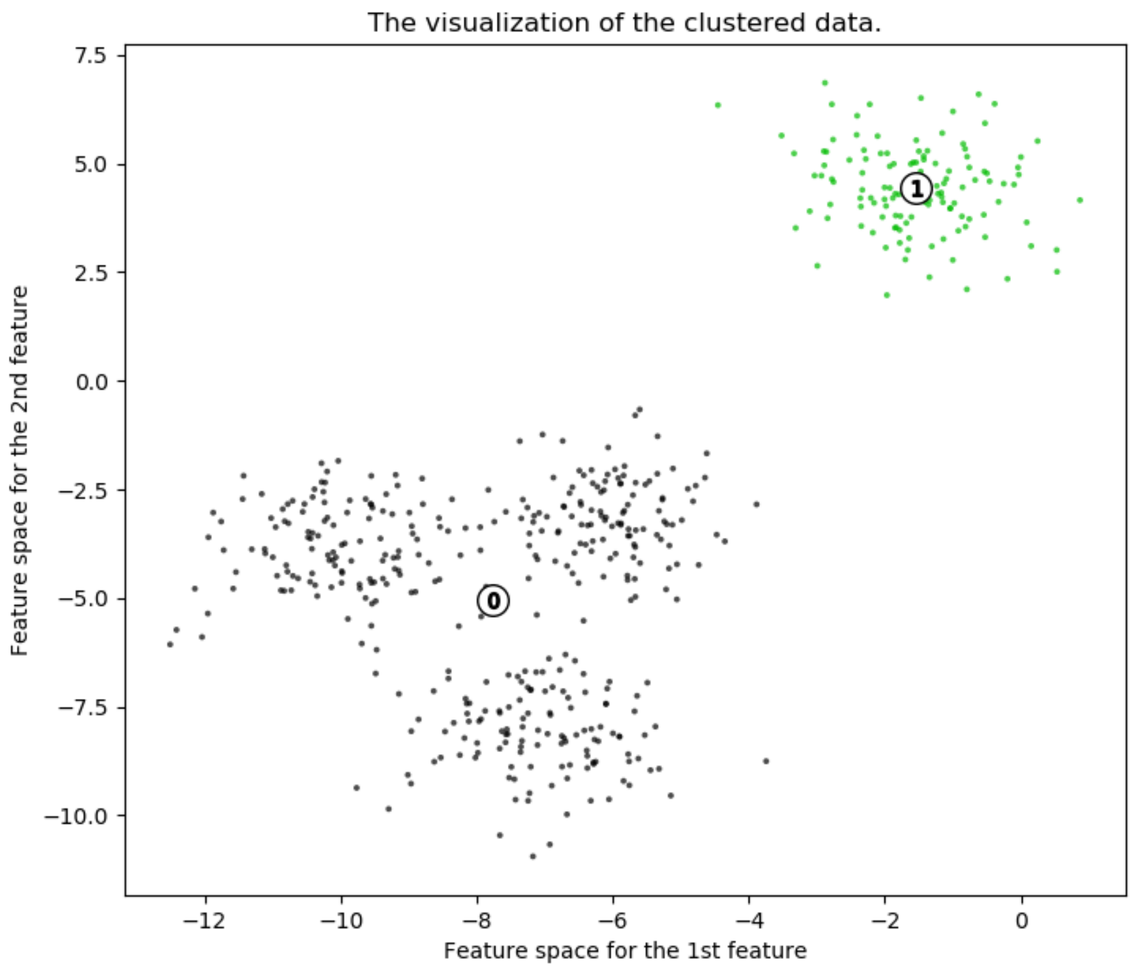
- To calculate the silhouette coefficient, apply the following three steps:

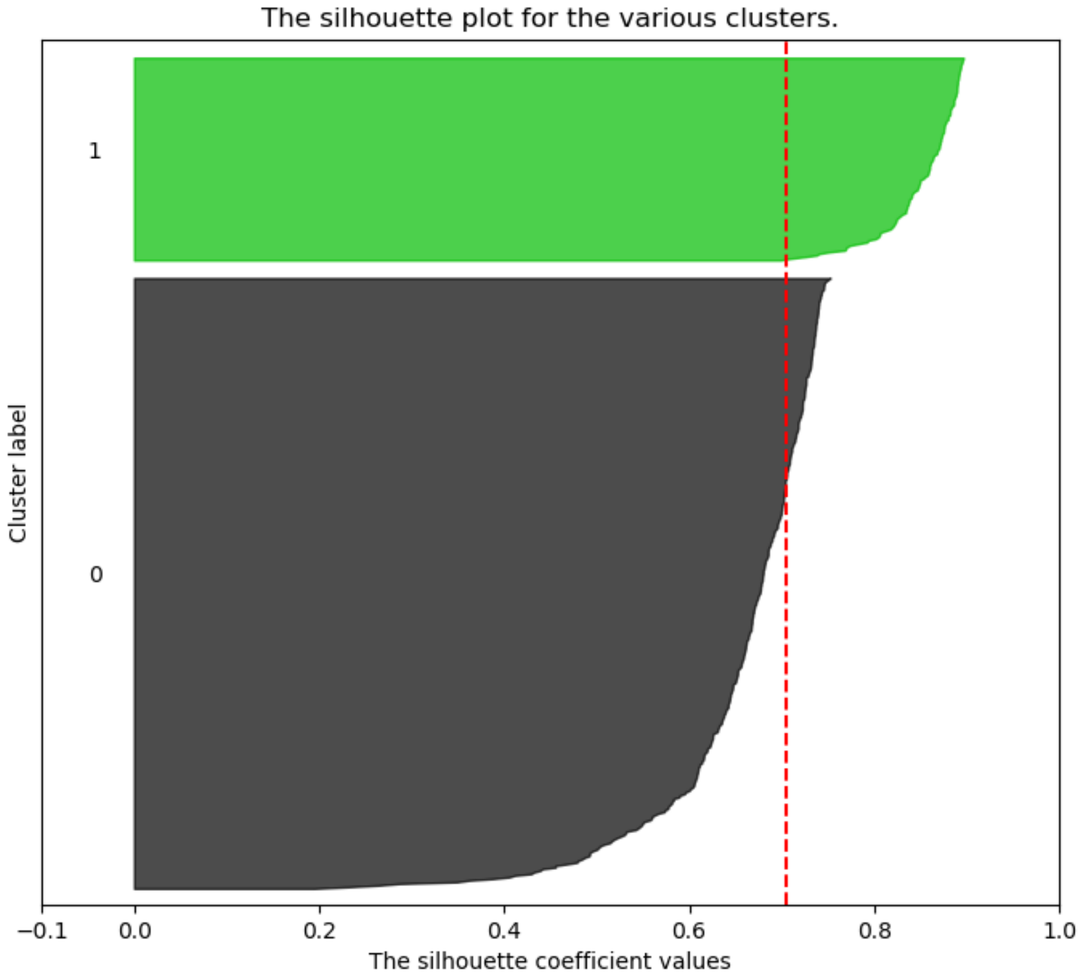
  1) Calculate the cluster "*cohesion*" as the average distance between a sample and all other points in the same cluster.

  2) Calculate the cluster separation from the next closest cluster as the average distance between the sample and all samples in the nearest cluster.

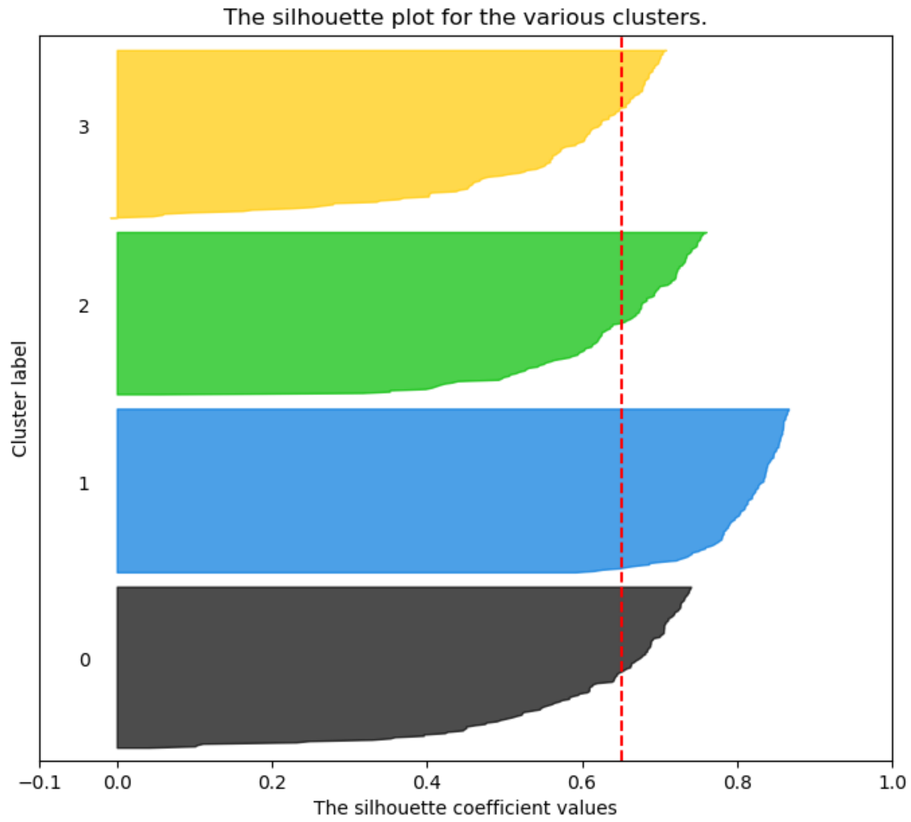
  3) Calculate the silhouette as the difference between cluster cohesion and separation divided by the greater of the two (whichever that may be)

Silhouette example – poor fit





Silhouette example – good fit



More silhouette examples:
http://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html

Step 5: measure clustering quality

Script

```
from matplotlib import cm
from sklearn.metrics import silhouette_samples

# We are now going to create a plot of the silhouette coefficient for a k-means
# clustering with 2 centers, as proven optimal from the elbow plot we just
# created in the previous slides.
# We first run k-means with 2 centers
km2 = KMeans(n_clusters = 2,
             init = 'k-means++',
             n_init = 10,
             max_iter = 300,
             tol = 1e-04,
             random_state = 0)

y_km2 = km2.fit_predict(sample_data)
```

Step 5: measure clustering quality

Script

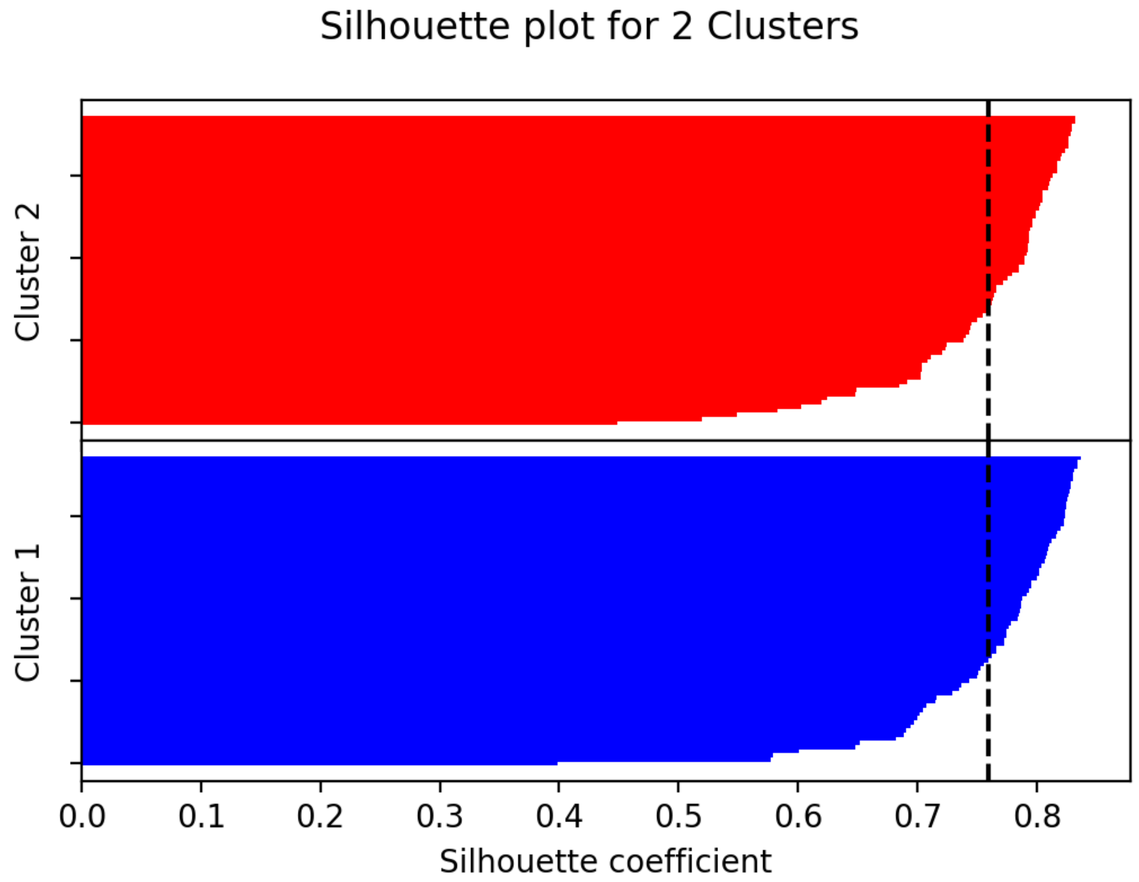
```
# Plot the silhouette for each cluster -
# we can use 2 subplots and create a horizontal barplot on each
fig, axes = plt.subplots(nrows = 2, ncols = 1, sharex = True)

cluster_0 = sample_data[y_km2 == 0,:] # the red one
cluster_1 = sample_data[y_km2 == 1,:] # the blue one

silhouette_vals = silhouette_samples(sample_data, y_km2, metric = 'euclidean')

silhouette_0 = sorted(silhouette_vals[y_km2 == 0])
silhouette_1 = sorted(silhouette_vals[y_km2 == 1])

axes[1].barh(range(0, len(cluster_0)), silhouette_0, height = 1.0,
             edgecolor = 'none', color = 'blue')

axes[1].set_ylabel('Cluster 1')
axes[1].set_yticklabels([])
```

Step 5: measure clustering quality

Script

```
axes[0].barh(range(0, len(cluster_1)), silhouette_1, height=1.0,
             edgecolor = 'none', color = 'red')
axes[0].set_ylabel('Cluster 2')
axes[0].set_yticklabels([])

# Add a vertical line for average silhouette value
silhouette_avg = np.mean(silhouette_vals)
axes[0].axvline(silhouette_avg, color = "black", linestyle = "--")
axes[1].axvline(silhouette_avg, color = "black", linestyle = "--")

# Eliminate the gap between the two subplots
plt.subplots_adjust(hspace=0.0)

plt.xlabel('Silhouette coefficient')
fig.suptitle("Silhouette plot for 2 Clusters")

fig.savefig('plots/sample_data/silhouette_k2.png', dpi = 200)
```

Step 5: measure clustering quality



We see that both clusters are past the 'average silhouette value' and therefore can be considered pretty good.

Step 5: measure clustering quality

Script

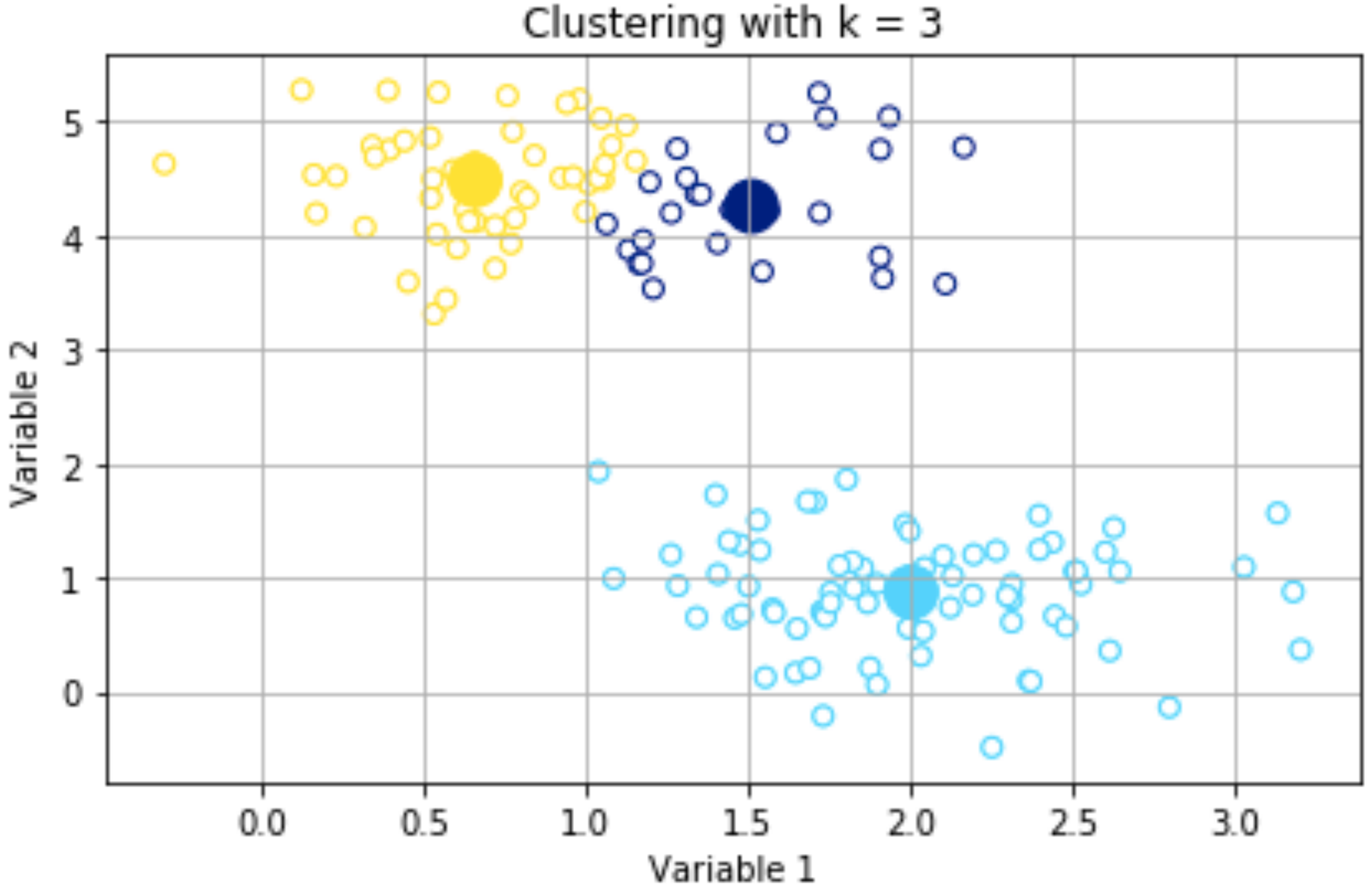
```
# Now, let's run clustering with 3 centers.
km3 = KMeans(n_clusters = 3,
             init = 'k-means++',
             n_init = 10,
             max_iter = 300,
             tol = 1e-04,
             random_state = 0)

y_km3 = km3.fit_predict(sample_data)

colors = [cm.jet(float(i)/3) for i in y_km3]
plt.scatter(sample_data[:, 0], sample_data[:, 1],
            marker = 'o',
            c = 'white', edgecolor = colors)
```

Step 5: measure clustering quality

Script

```
centroid_colors = [cm.jet(float(i)/3) for i in range(3)]
plt.scatter(km3.cluster_centers_[:, 0],
            km3.cluster_centers_[:, 1],
            s = 250,
            marker = 'o',
            c = centroid_colors,
            label = 'centroids')

plt.grid()
plt.xlabel("Variable 1")
plt.ylabel("Variable 2")
plt.title("Clustering with k = 3")
plt.tight_layout()

plt.show()

plt.savefig('plots/sample_data/kmeans_3.png', dpi = 200)
```

Step 5: measure clustering quality



Step 5: measure clustering quality

Script

```
# Now we get the silhouette values for each cluster. The for loop below
# is looping through each of the cluster numbers, in this case i = 0, 1
# and 2 (clusters 1-3). Enumerate is a function native to Python that allows
# us to loop over something and have an automatic counter.
cluster_labels = np.unique(y_km3)
n_clusters = cluster_labels.shape[0]
silhouette_vals = silhouette_samples(sample_data, y_km3, metric = 'euclidean')
y_ax_lower, y_ax_upper = 0, 0
yticks = []
for i, c in enumerate(cluster_labels):
    c_silhouette_vals = silhouette_vals[y_km3 == c]
    c_silhouette_vals.sort()
    y_ax_upper += len(c_silhouette_vals)
    color = cm.jet(float(i) / n_clusters) # create a unique color based on i
    plt.barh(range(y_ax_lower, y_ax_upper), c_silhouette_vals, height = 1.0,
             edgecolor = 'none', color = color)
    yticks.append((y_ax_lower + y_ax_upper) / 2.)
    y_ax_lower += len(c_silhouette_vals)
```

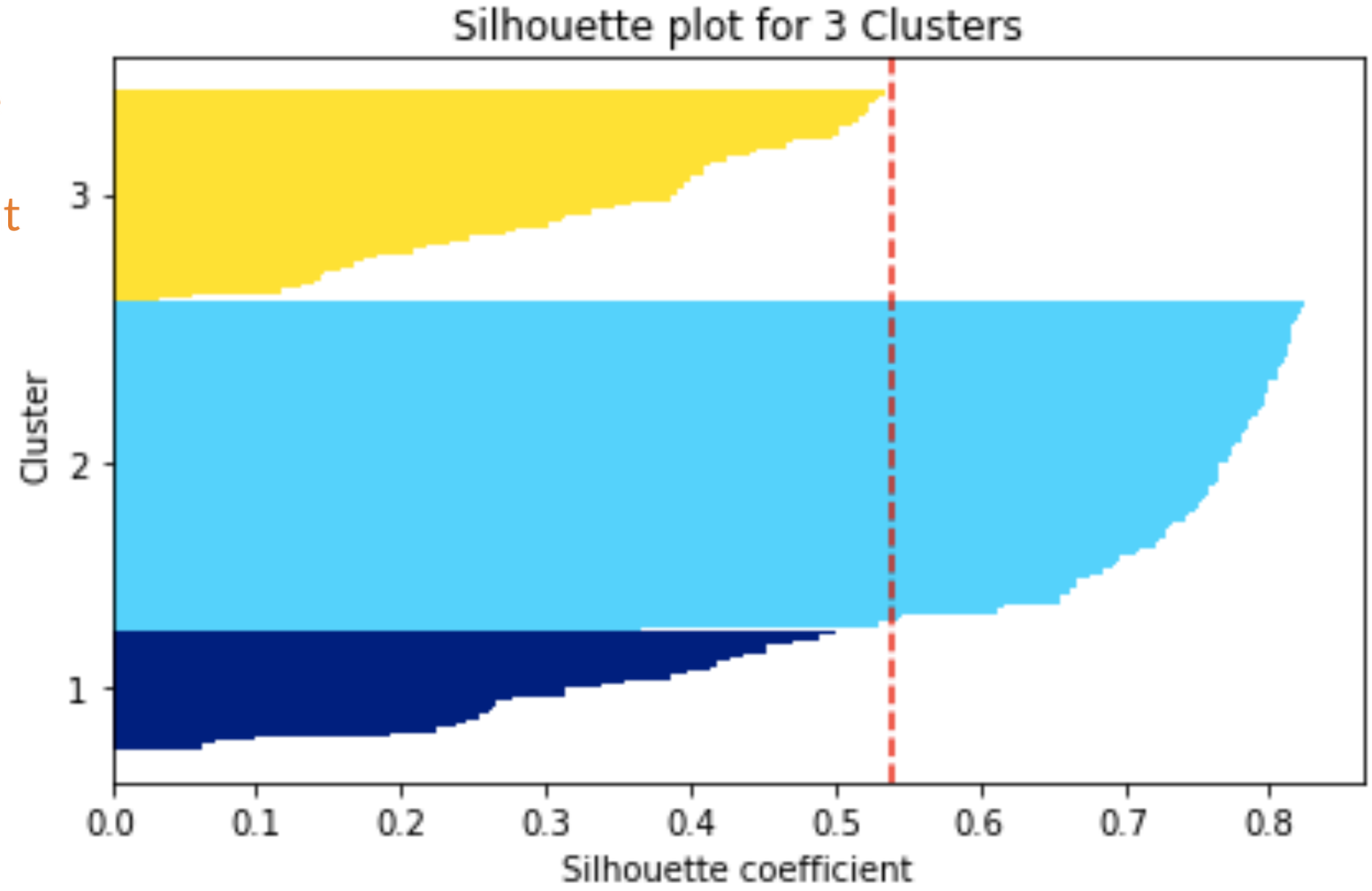
Step 5: measure clustering quality

Script

```
# To summarize the goodness of our clustering, we add the average silhouette
# coefficient line to the plot.
silhouette_avg = np.mean(silhouette_vals)
plt.axvline(silhouette_avg, color = "red", linestyle = "--")

plt.yticks(yticks, cluster_labels + 1)
plt.ylabel('Cluster')
plt.xlabel('Silhouette coefficient')



plt.title("Silhouette plot for 3 Clusters")
plt.tight_layout()
plt.show()

plt.savefig('plots/sample_data/silhouette_k3.png', dpi = 300)
# plt.close()
```

Step 5: measure clustering quality

We can see that the light blue cluster is more disperse – this doesn't seem to be as good a fit as 2 clusters.



Fry's General Store - Candy clustering

How do we place chocolates at the Fry's, the local grocery store?

1. Data set consists of 100 local candy and chocolate bars
2. Each has been rated by four categories:
   - Sweetness
   - Sourness
   - Nuttiness
   - Texture
3. Rated on a 0-1 scale

Four variables of ratings: Sweetness, Sourness, Nuttiness, Texture

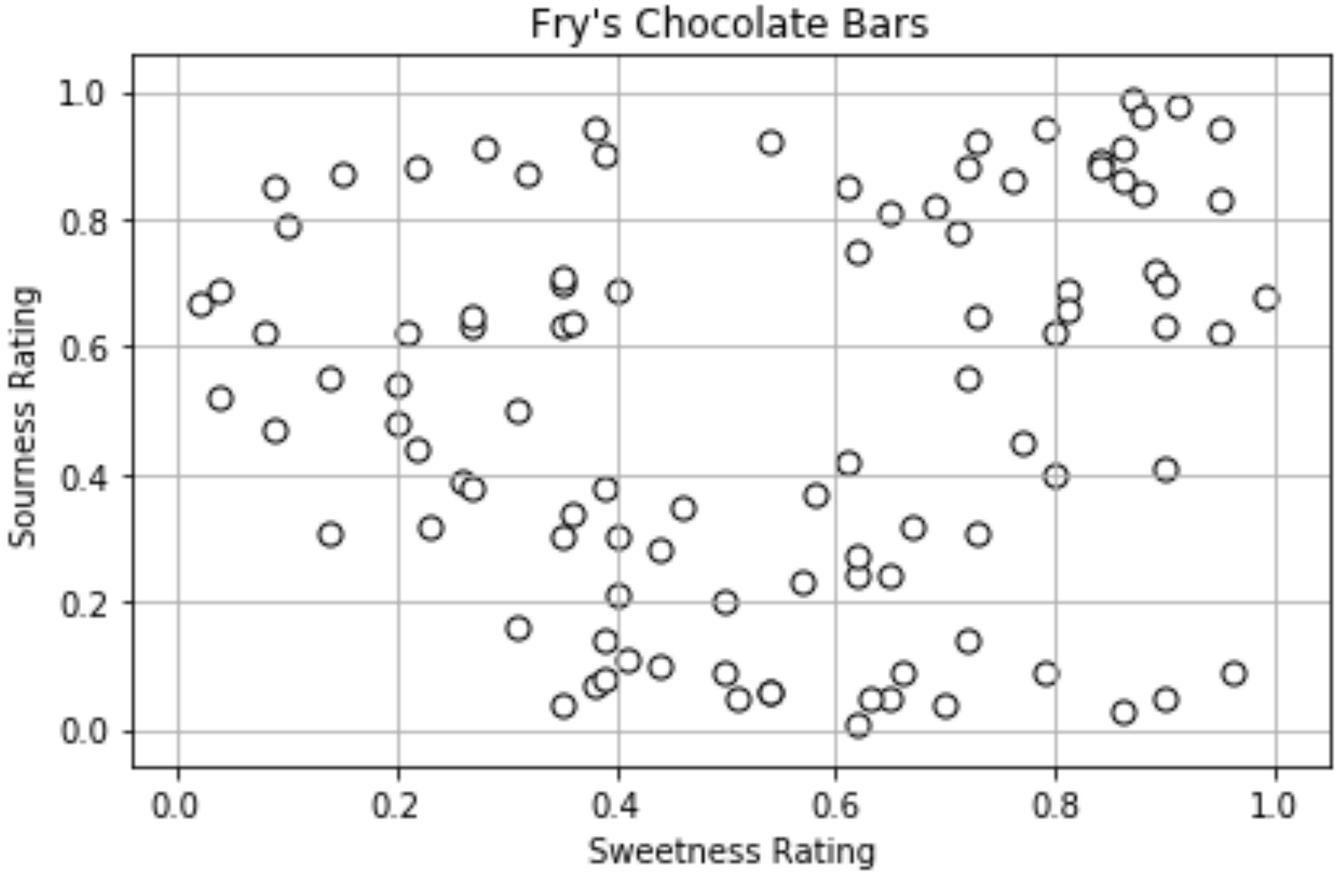
The local candy store that needs help selling their large stock of chocolate and candy!

Finding patterns between the sweets

Each data point represents a chocolate bar

- How do we identify similar chocolates?

- Assumption:
  - If we place chocolates that are similar, we will sell more.
  - Customers that come in will likely see more than what they just came to buy.



Step 1: read in data

Script

```
# Let's use a real dataset to apply clustering.
from sklearn import datasets
import numpy as np
import os
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.cluster import Kmeans

candies = pd.read_csv('small_candy.csv')

# Remove names but store them as a
# separate variable
candies = candies.set_index('Names')

# Summarize the numerical columns:
candies.describe()
```

IPython console

Console 1/A

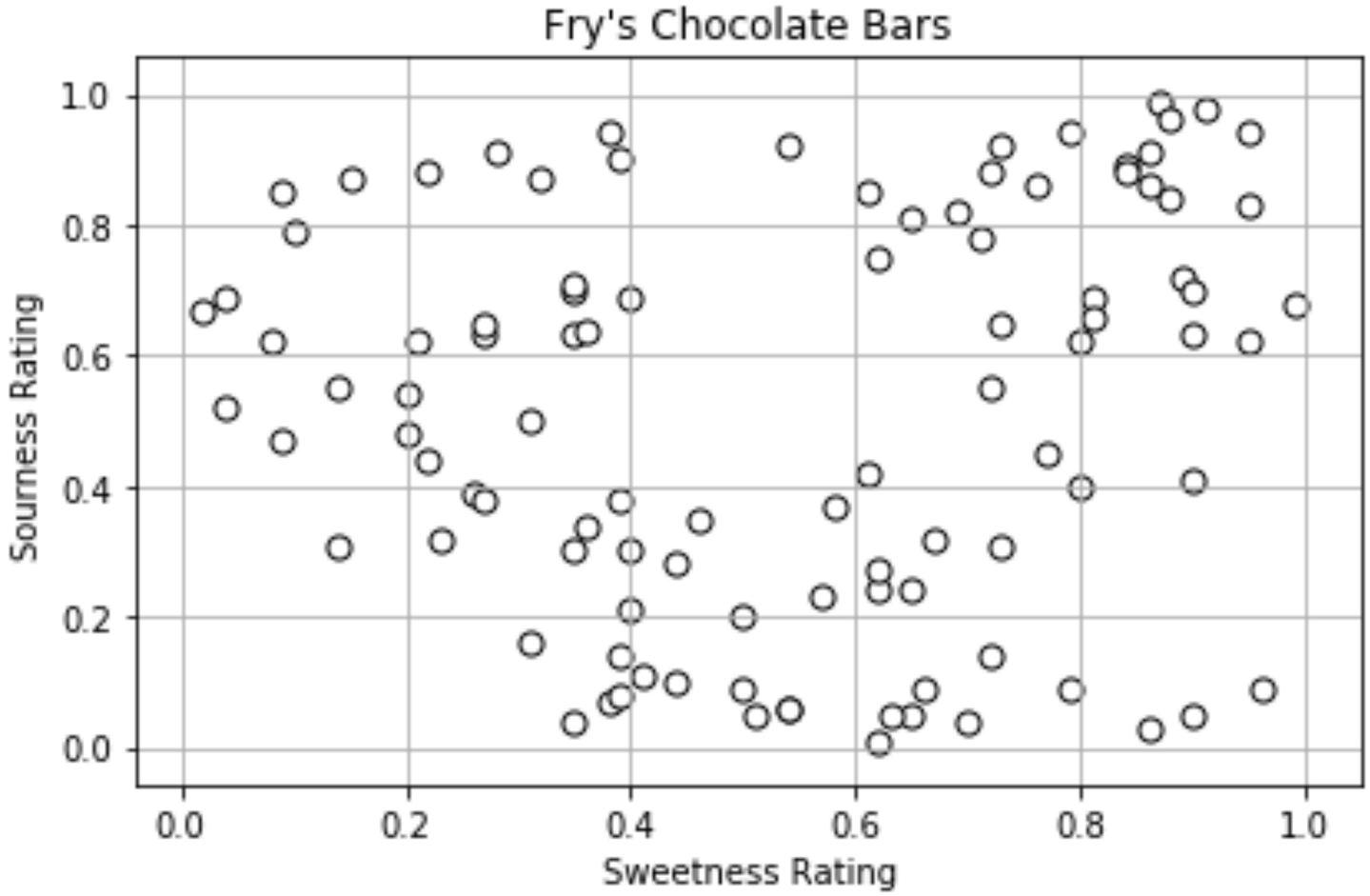
```
In [27]: candies.describe()
Out[27]:
       Sweetness    Sourness   Nuttiness     Texture
count  100.000000  100.000000  100.000000  100.000000
mean     0.539600    0.509100    0.608700    0.517100
std      0.267744    0.306542    0.257685    0.282787
min      0.020000    0.010000    0.010000    0.010000
25%      0.350000    0.240000    0.405000    0.300000
50%      0.555000    0.545000    0.665000    0.525000
75%      0.775000    0.795000    0.800000    0.755000
max      0.990000    0.990000    0.990000    0.990000
```

Exploratory data analysis

Script

```
# Let's do some exploratory data analysis (EDA) before we do the
# clustering to see if we can find any patterns.
# Plot variables against each other:

# Plot Sweetness vs Sourness
plt.scatter(candies['Sweetness'], candies['Sourness'],
            c = 'white', marker = 'o',
            edgecolor = 'black', s = 50)
plt.grid()
plt.xlabel('Sweetness Rating')
plt.ylabel('Sourness Rating')
plt.title("Fry's Chocolate Bars")
plt.tight_layout()
plt.show()

plt.savefig('plots/candy/eda.png', dpi = 200)
```

Exploratory data analysis



Are there any patterns we can pick out?

Step 2: select k

Script

```
# Notice that we run KMeans on all of the variables
# (Sweetness, Sourness, Nuttiness, and Texture)
# unlike in our example when we only used 2 variables.
# KMeans can be run on any number of variables, not just 2!

# Inertia plot to assess optimal number of clusters
inertias = []
try_k = range(1, 11)
for k in try_k:
    km = KMeans(n_clusters = k,
                init = 'k-means++',
                n_init = 10,
                max_iter = 300,
                random_state = 0)
    km.fit(candies) # this will use all variables in the candies dataframe
    inertias.append(km.inertia_)
```

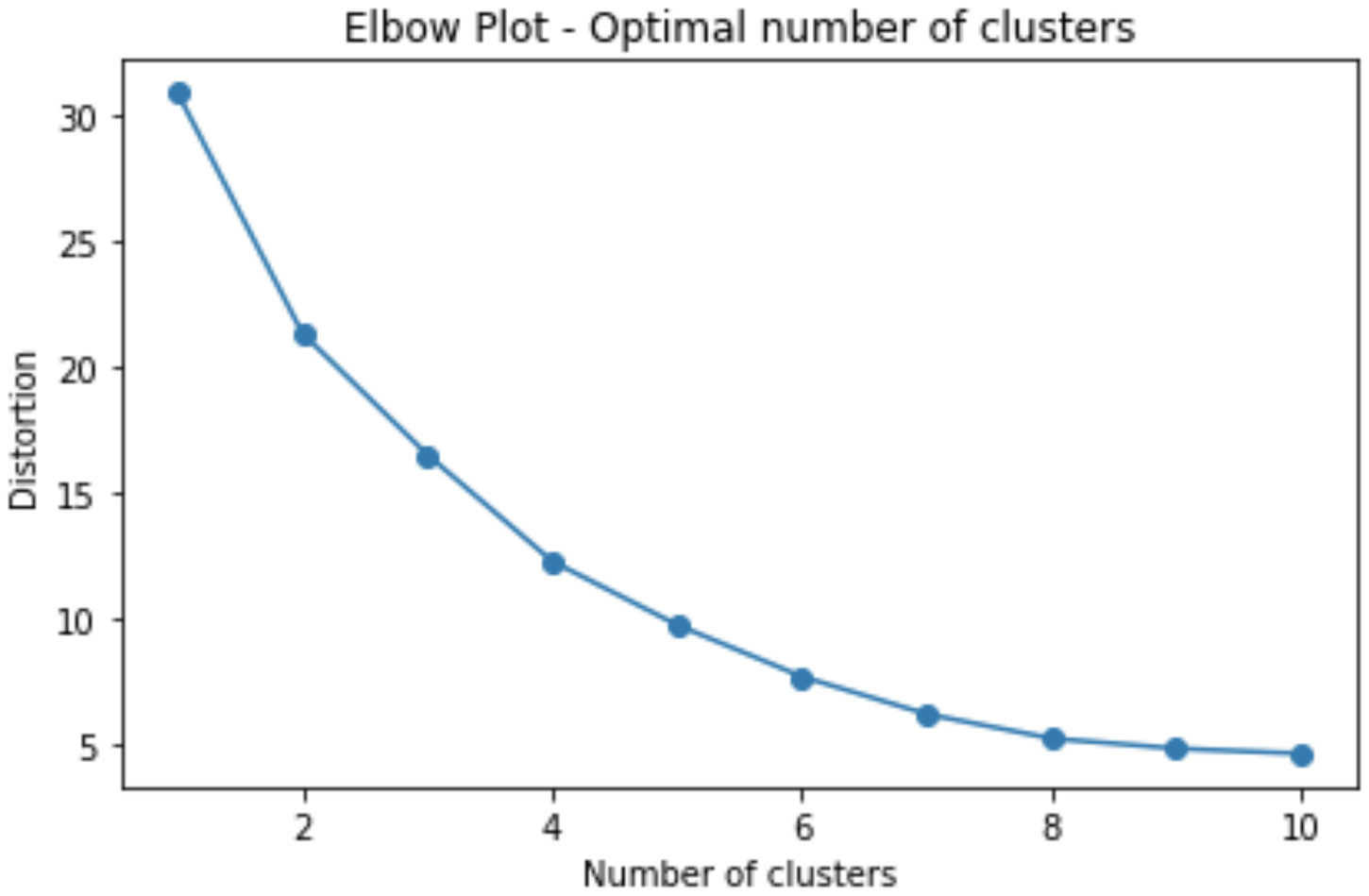
Step 2: select k

Script

```
del km

plt.plot(try_k, inertias, marker = 'o')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
plt.title('Elbow Plot - Optimal number of clusters')
plt.tight_layout()

plt.show()

plt.savefig('plots/candy/elbow.png', dpi = 200)
```

Elbow method: measure variance

It will not always be as clear cut as the example. However, here we look at the plot as well as the distortion percentages, and see k = 3 is the optimal number of clusters



Step 3: run k-means

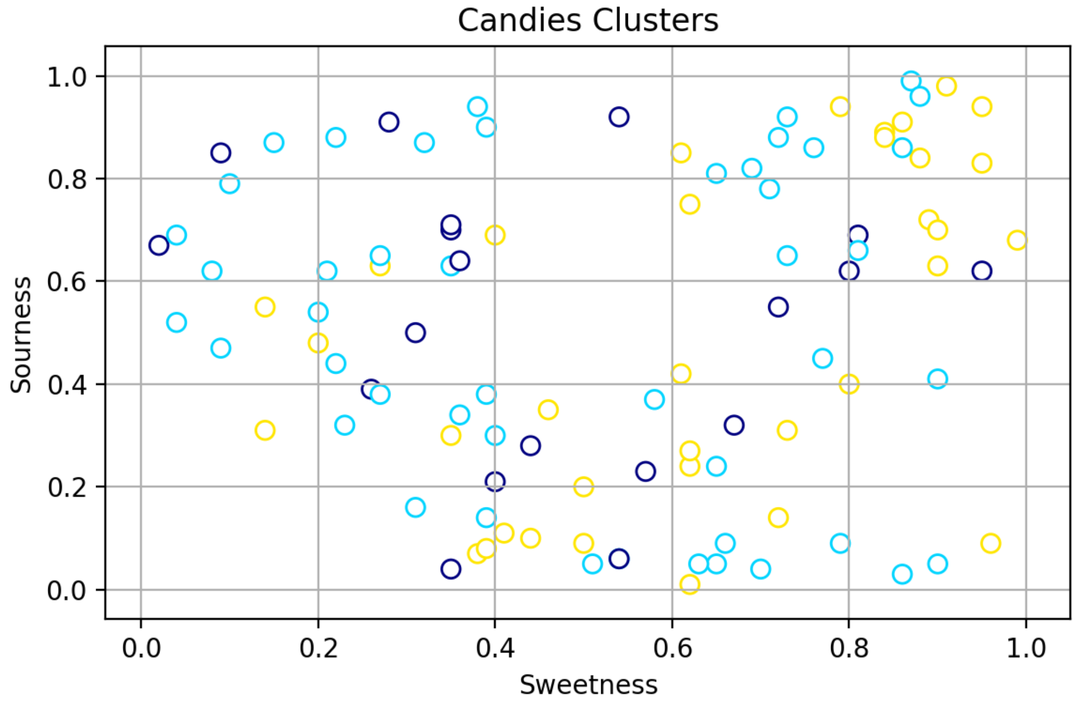
Script

```
# Kmeans, using 3 clusters, using k-means++ instead of
# random for centroids
km3 = KMeans(n_clusters = 3,
             init = 'k-means++',
             n_init = 10,
             max_iter = 300,
             tol = 1e-04,
             random_state = 0)

# Predict which cluster each point will fall into.
y_km3 = km3.fit_predict(candies)
```

Step 4: visualize clustering results

Script

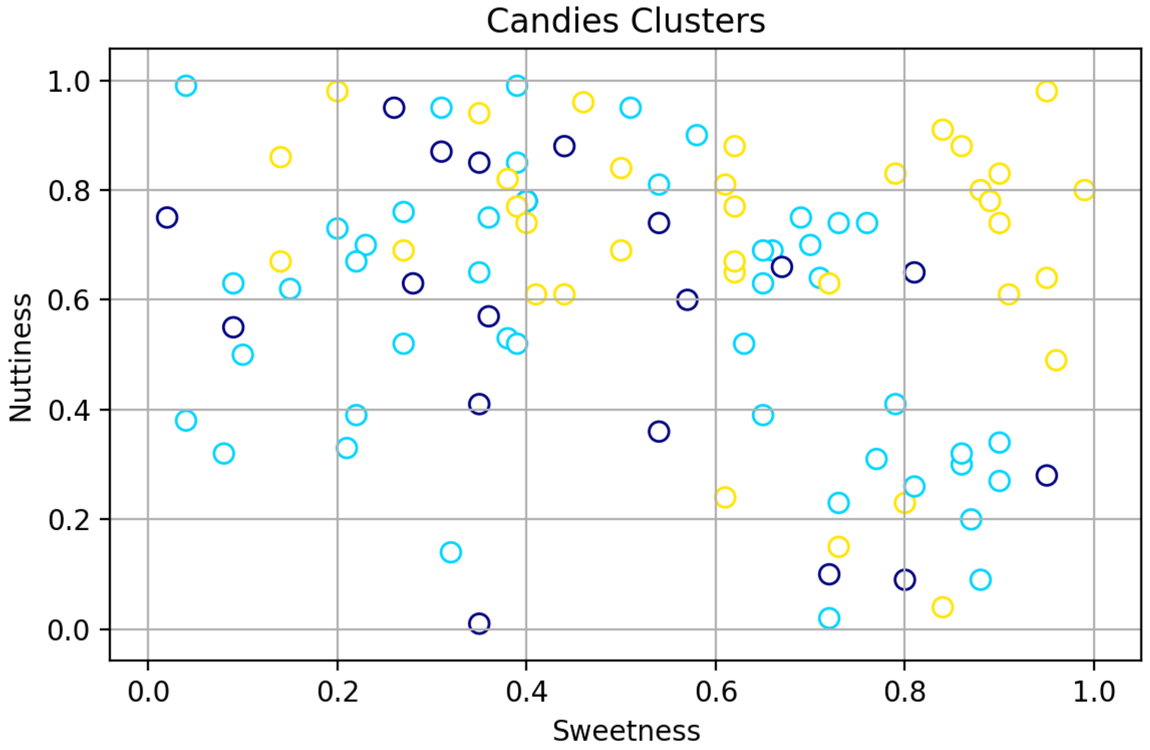
```
n_clusters = 3
colors = [cm.jet(float(i) / n_clusters) for i in y_km3]

# Plot Sweetness vs Sourness
plt.scatter(candies['Sweetness'], candies['Sourness'],
            s = 50,
            c = 'white',
            edgecolor = colors)
plt.grid()
plt.xlabel('Sweetness')
plt.ylabel('Sourness')
plt.title("Candies Clusters")
plt.tight_layout()

plt.savefig('plots/candy/
            kmeans_k3_sweet_sour.png',
            dpi = 200)
plt.close()
```



Step 4: visualize clustering results

Script

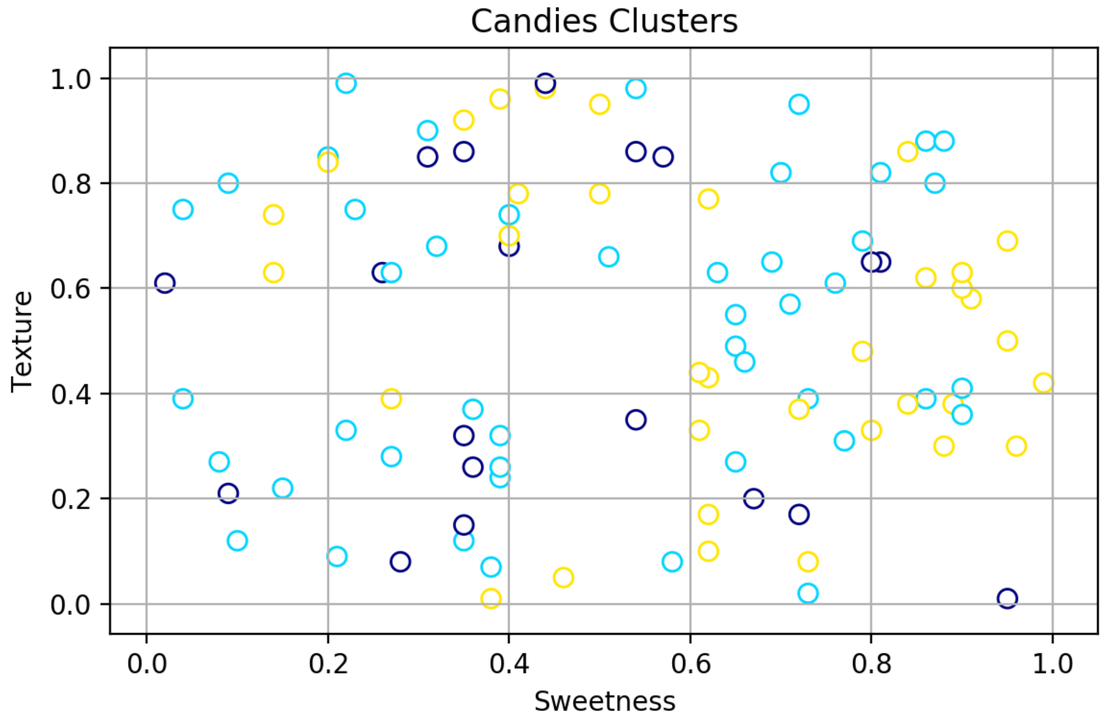
```
# Plot sweetness vs nuttiness
plt.scatter(candies['Sweetness'], candies['Nuttiness'],
            s = 50,
            c = 'white',
            edgecolor = colors)
plt.grid()
plt.xlabel('Sweetness')
plt.ylabel('Nuttiness')
plt.title("Candies Clusters")
plt.tight_layout()

plt.savefig('plots/candy/
             kmeans_k3_sweet_nutty.png',
            dpi = 200)
plt.close()
```



Step 4: visualize clustering results

Script

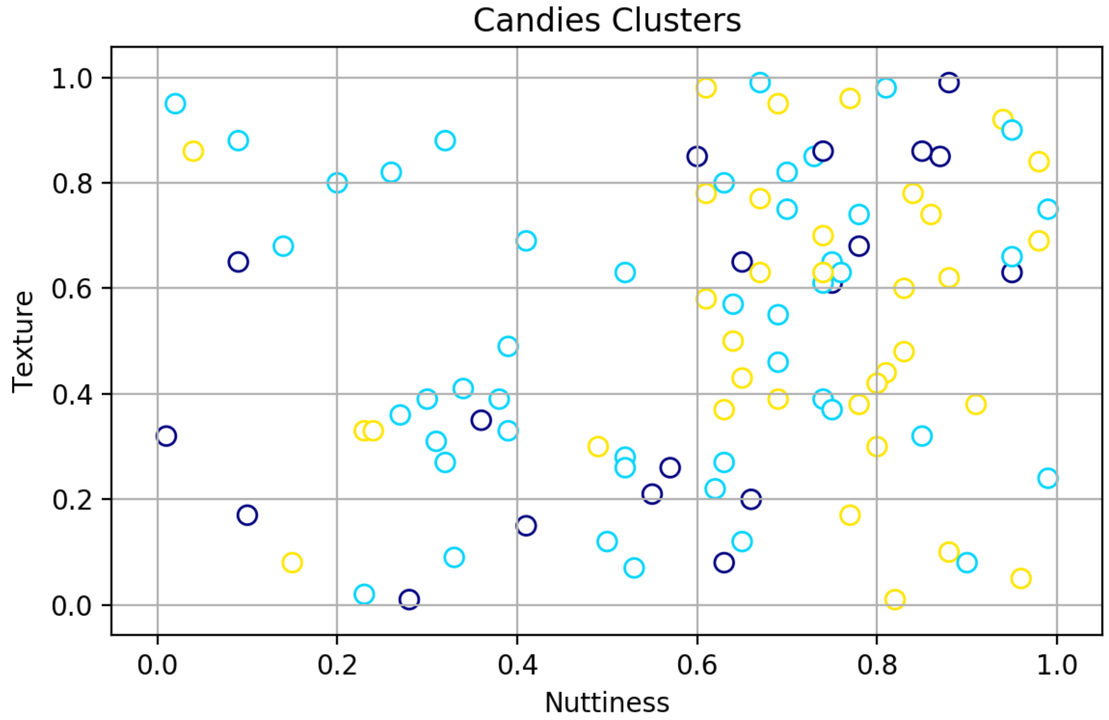
```
# Plot sweetness vs texture
plt.scatter(candies['Sweetness'], candies['Texture'],
            s = 50,
            c = 'white',
            edgecolor = colors)
plt.grid()
plt.xlabel('Sweetness')
plt.ylabel('Texture')
plt.title("Candies Clusters")
plt.tight_layout()

plt.savefig('plots/candy/
            kmeans_k3_sweet_texture.png',
            dpi = 200)
plt.close()
```



Step 4: visualize clustering results

Script

```
# Plot nutiness vs texture
plt.scatter(candies['Nuttiness'], candies['Texture'],
            s = 50,
            c = 'white',
            edgecolor = colors)
plt.grid()
plt.xlabel('Nuttiness')
plt.ylabel('Texture')
plt.title("Candies Clusters")
plt.tight_layout()

plt.savefig('plots/candy/
            kmeans_k3_nutty_texture.png',
            dpi = 200)
plt.close()
```

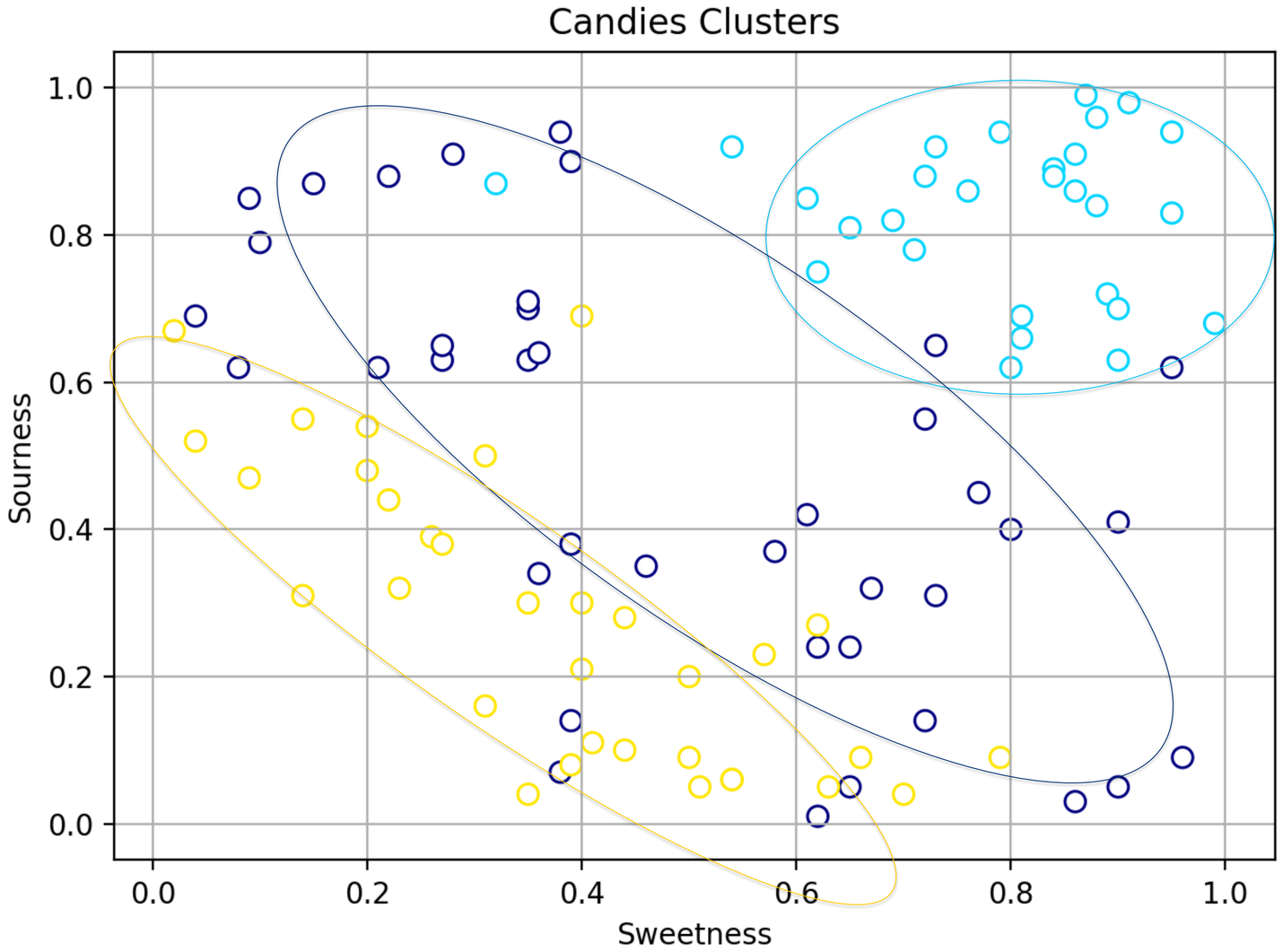


Step 4: visualize plot

As you can see, the clusters are not cleanly separated. This is because we used all 4 variables (sweetness, sourness, nuttiness and texture) to do the clustering.

Therefore candies in the same clusters are *as similar as possible across all 4 variables*.

If we only care about the sweetness and sourness variables, we could repeat the clustering with just these variables.



Exercise time!

Step 5: measure the goodness

Script

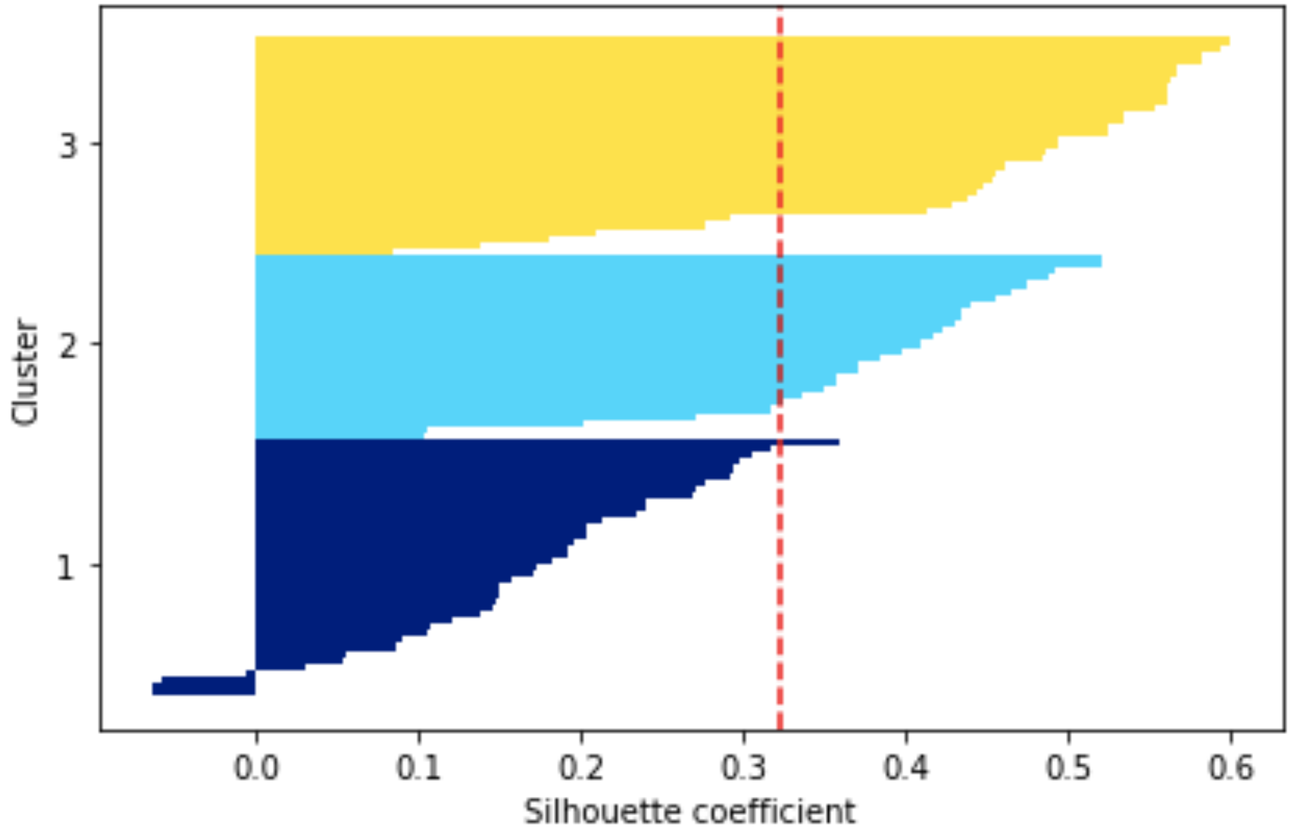
```
# Now look at the goodness of the clusters by looking
# at the silhouette coefficient
from sklearn.metrics import silhouette_samples

cluster_labels = np.unique(y_km3)
n_clusters = cluster_labels.shape[0]
silhouette_vals = silhouette_samples(candies, y_km3, metric = 'euclidean')
y_ax_lower, y_ax_upper = 0, 0
yticks = []
for i, c in enumerate(cluster_labels):
    c_silhouette_vals = silhouette_vals[y_km3 == c]
    c_silhouette_vals.sort()
    y_ax_upper += len(c_silhouette_vals)
    color = cm.jet(float(i) / n_clusters)
    plt.barh(range(y_ax_lower, y_ax_upper), c_silhouette_vals, height = 1.0,
             edgecolor = 'none', color = color)
    yticks.append((y_ax_lower + y_ax_upper) / 2.)
    y_ax_lower += len(c_silhouette_vals)
```

Step 5: measure the goodness

Script

```
silhouette_avg = np.mean(silhouette_vals)
plt.axvline(silhouette_avg, color = "red", linestyle = "--")

plt.yticks(yticks, cluster_labels + 1)
plt.ylabel('Cluster')
plt.xlabel('Silhouette coefficient')

plt.tight_layout()

plt.show()

plt.savefig('plots/candy/silhouette_k3.png', dpi = 200)
#plt.close()
```

Step 5: measure the goodness



We see that two of three clusters are past the 'average silhouette value' and therefore can be considered pretty good. One looks questionable. We can try running kmeans again with two clusters, or we can analyze the results and see if this makes more sense. This is why the knowledge of the subject is very important

Application of results

- We now can append the chocolate names back and filter them by cluster number to see the actual groups of chocolates

- By doing this, we can actually see the groups and get ready for action on the insights - we can use pandas for this!

Script

```
# Look at the chocolates in each cluster
candy_names = candies.index.values
candy_names[y_km3 == 0]
candy_names[y_km3 == 1]
candy_names[y_km3 == 2]
```

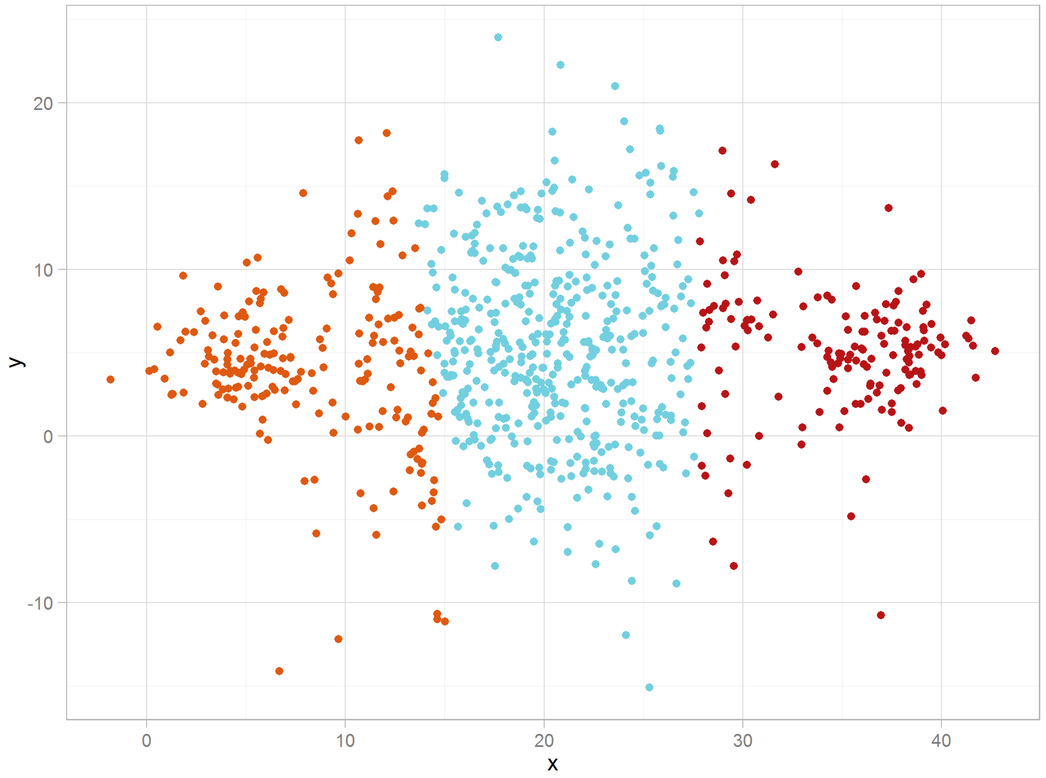
```
In [57]: candy_names[y_km3 == 0]
Out[57]:
array(['Nastly Sinful Fruit Chews', "Good Ivan's Dark Peanut-Butter
Fudge",
       'Nastly Honey Chocolate Coated Nonpariels',
```

k-means pitfalls to remember

Common pitfalls with clustering

Problem 1: clusters overlap when data points are unequally distributed

Original data

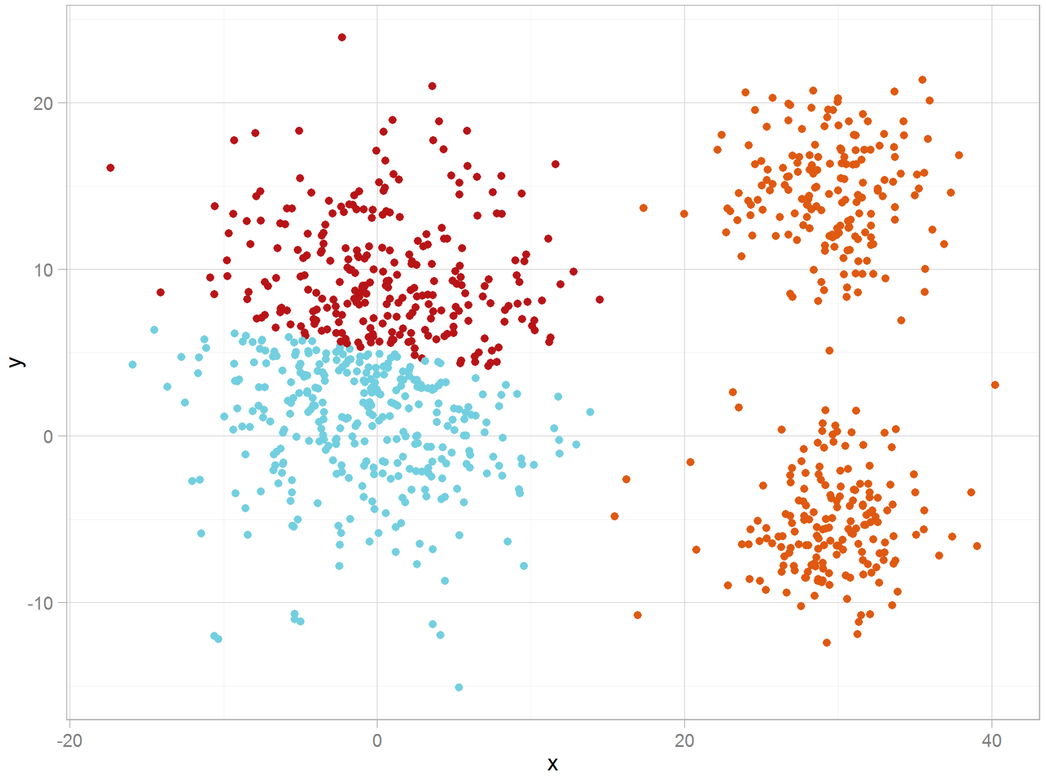


k-means clustering

Common pitfalls with clustering

Problem 2: clusters should have roughly the same density or else they may split!

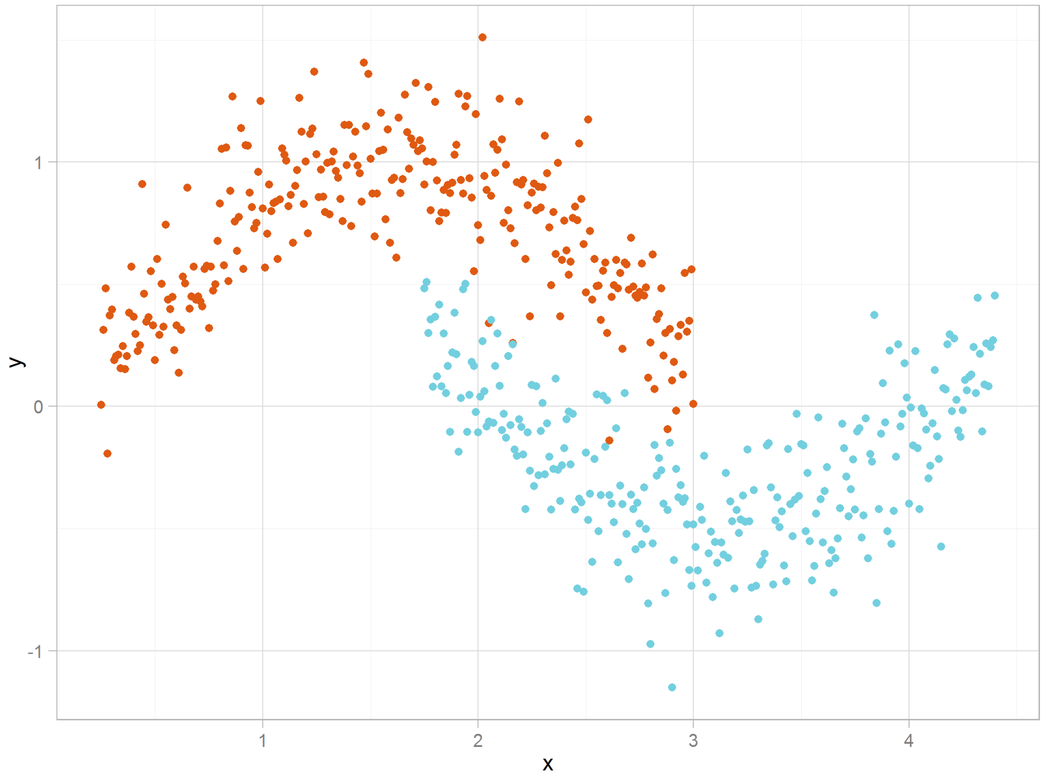
Original data



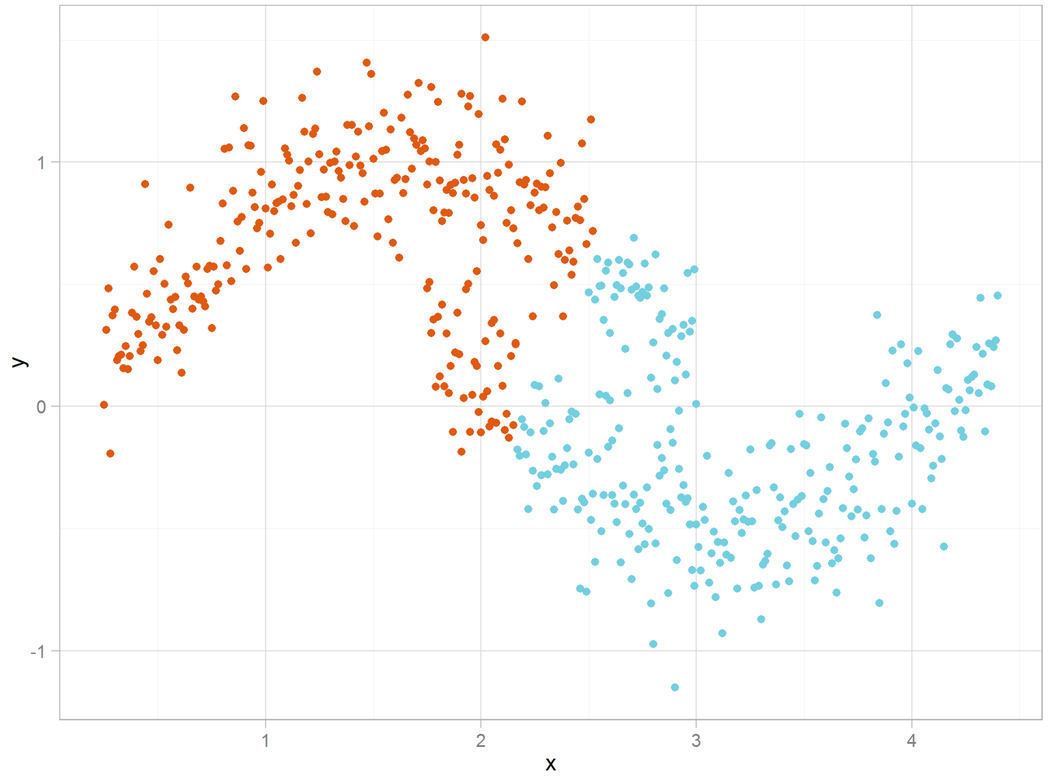
k-means clustering

Common pitfalls with clustering

Problem 3: clusters should be circular / elliptical. We can't use this method for certain distributions of data



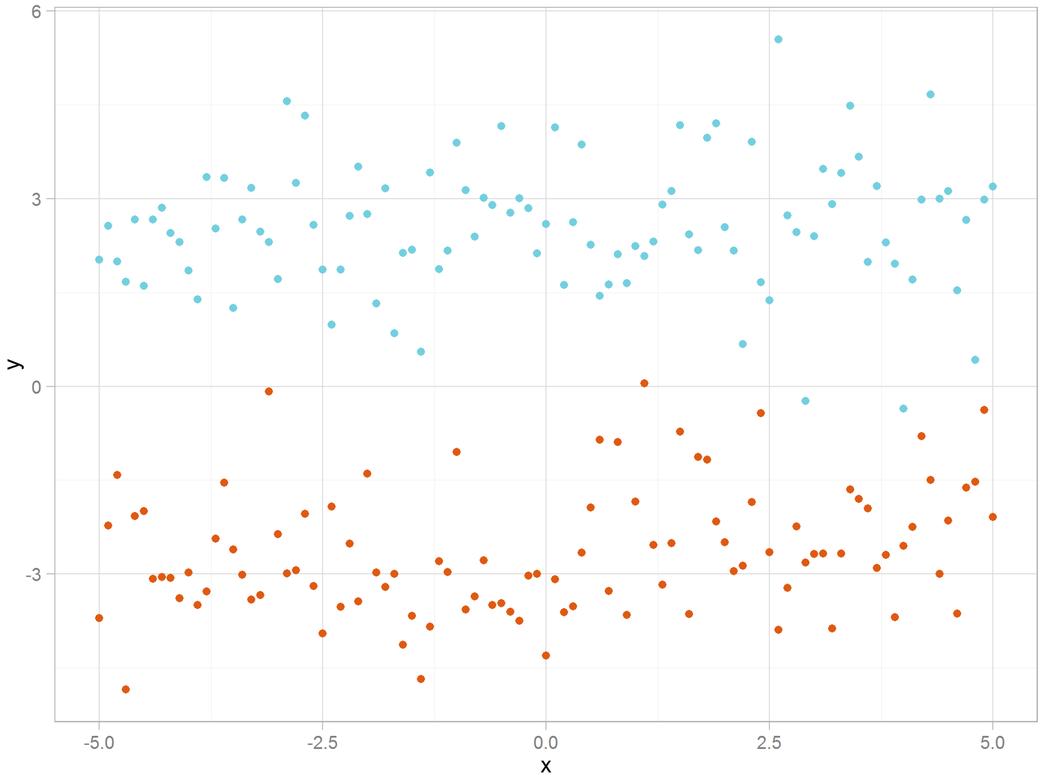
Original data



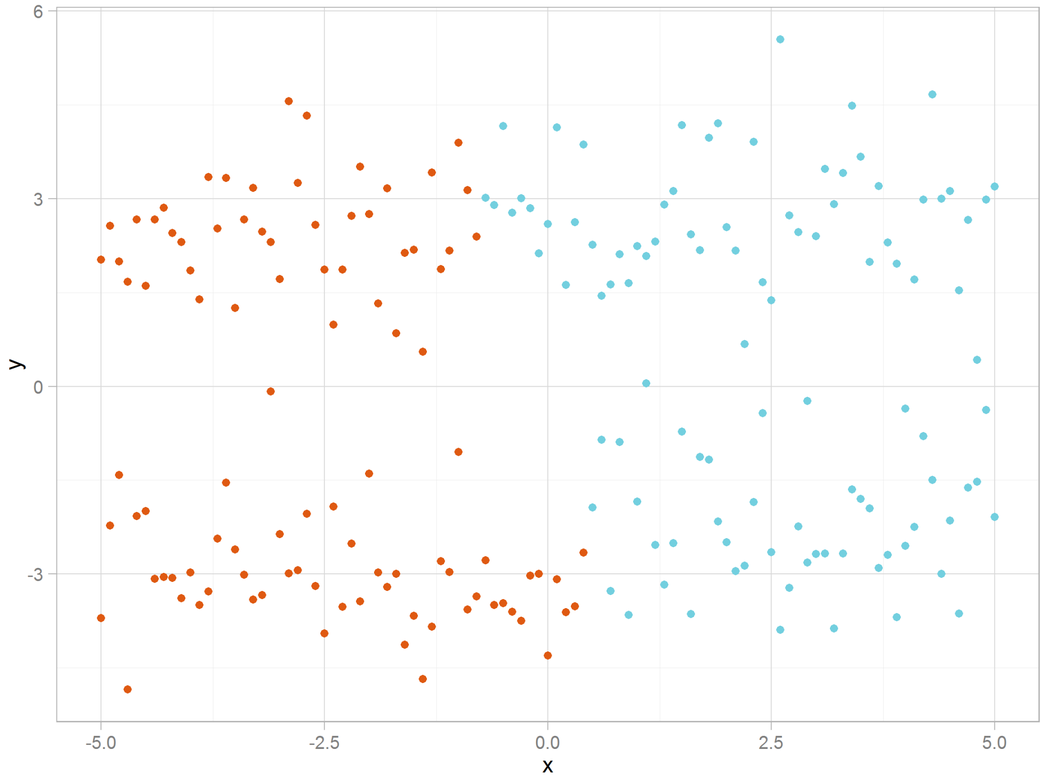
k-means clustering

Common problems with clustering

Problem 4: a bad starting point can lead to bad clustering!



Original data



k-means clustering

When to use k-means?

Your data should be

1. Equally distributed in expected clusters
2. Roughly similar density (distance) within clusters
3. Circular / elliptical data (visual assessment)

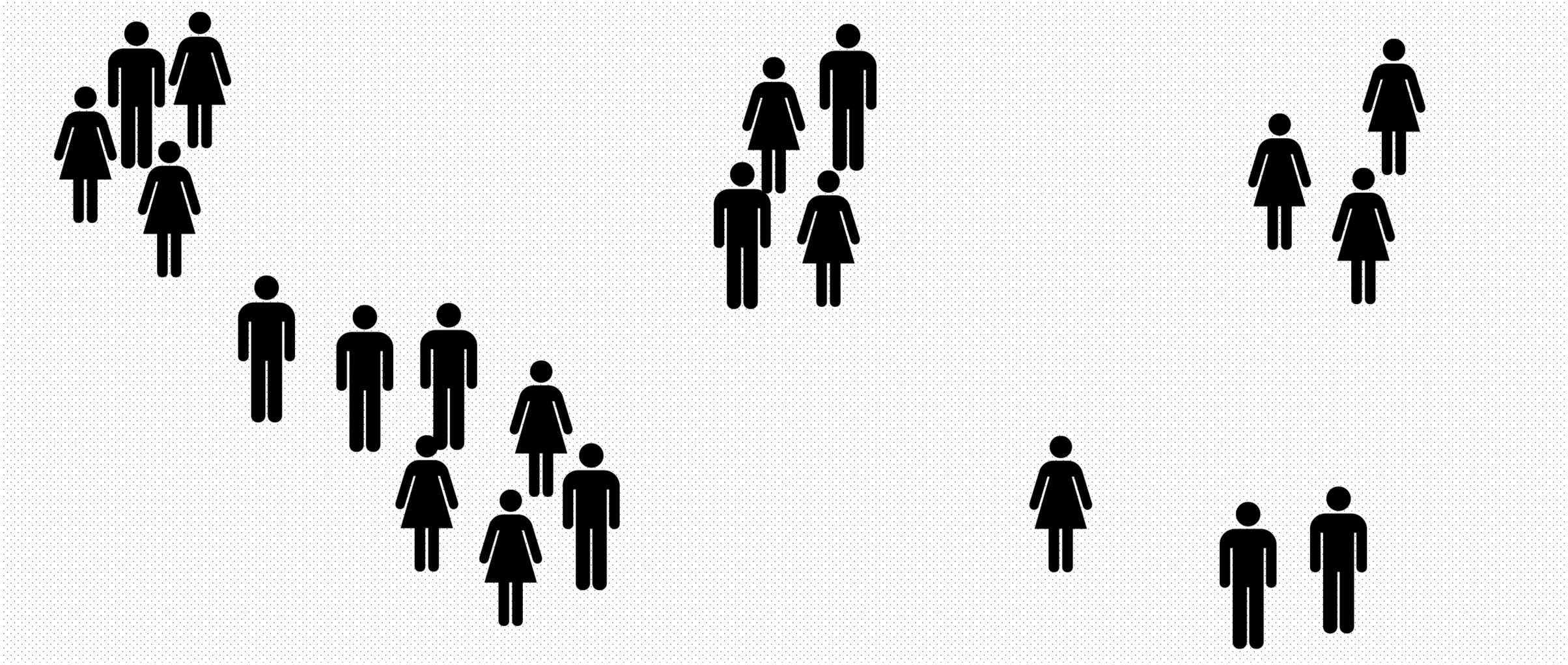
Best practices

1. Check that the conditions on the left are roughly satisfied by a visual inspection
2. Run the algorithm multiple times from different centroid starting points
3. Remove outlier data points when you first run the algorithm

Hierarchical clustering

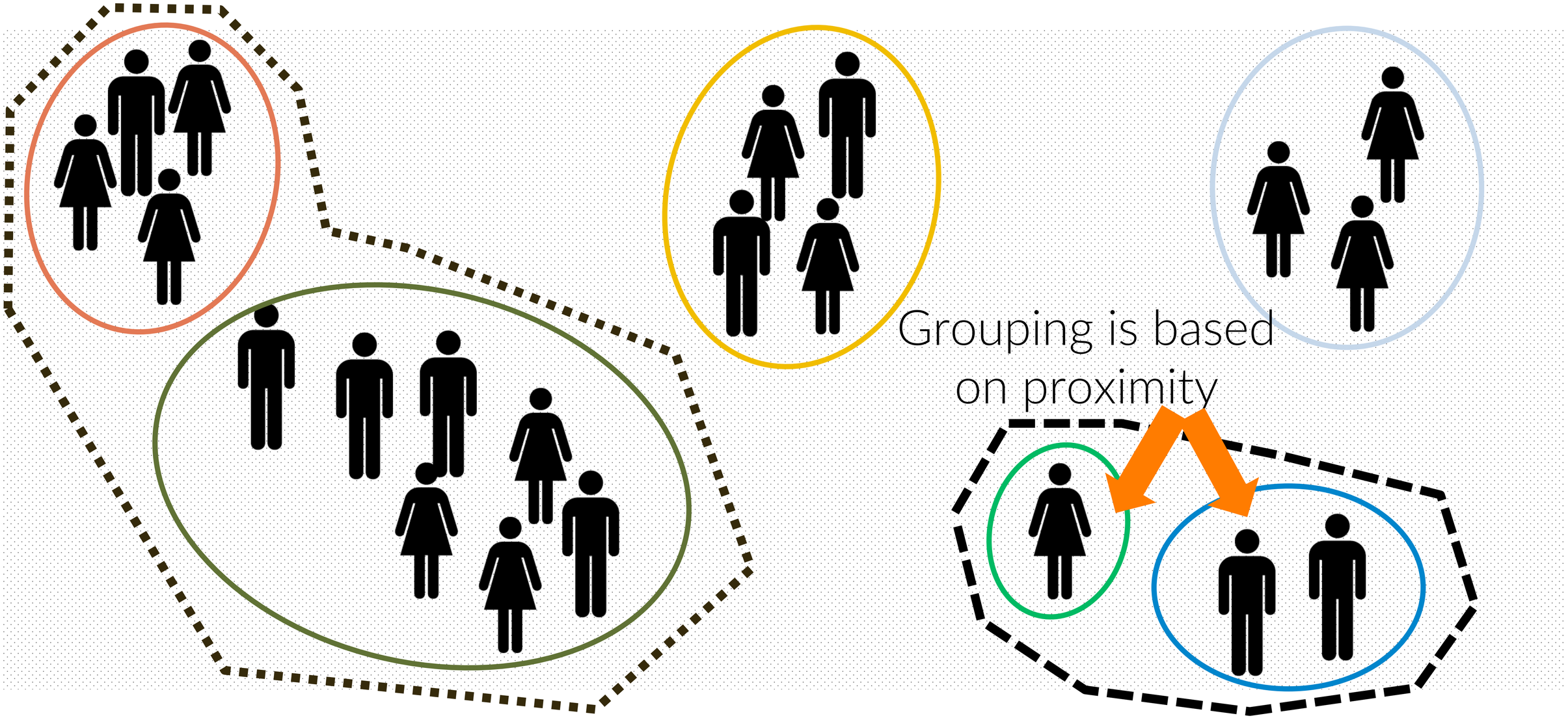
Tool / topic	Question to answer	Real world example
k-means clustering	Is there a pattern? How do we structure unstructured data?	Do members of Congress vote in patterns?
Hierarchical clustering	Is there a pattern? How do we structure unstructured data?	What is the structure of students at a local university?

Intuition: school yard



Arrange individuals into groups

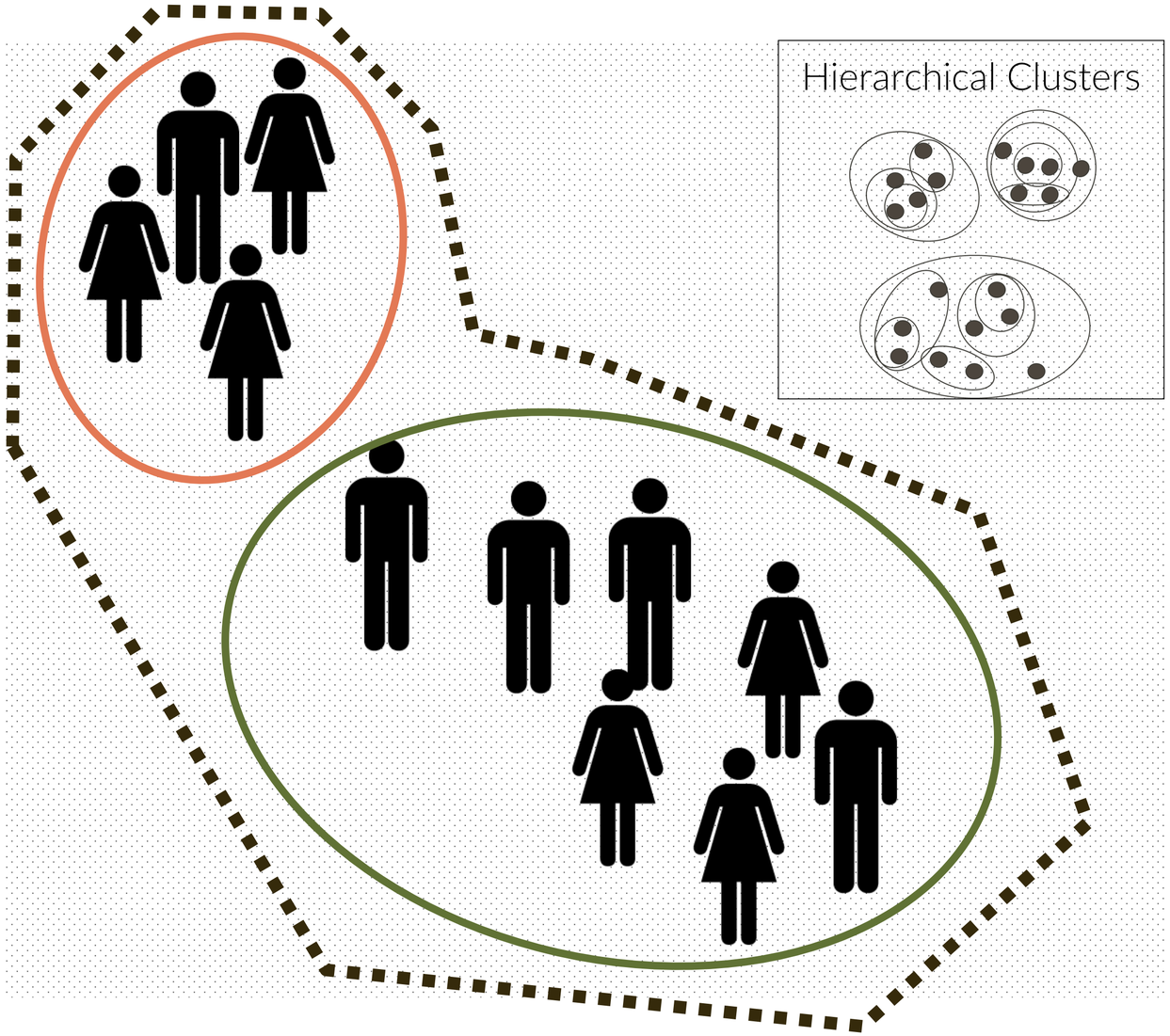
Arrange groups into larger groups



Group hierarchy

Some Notes

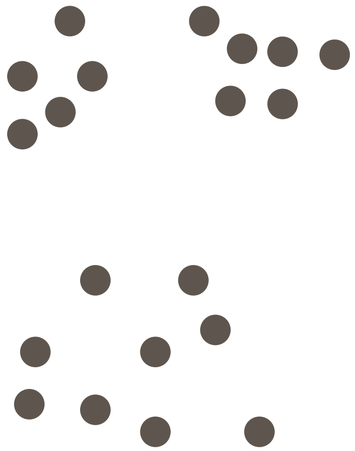
- Hierarchical Clustering
  - A method to generate clusters
  - Hierarchy = groups within groups
  - Grouping is based on proximity between individuals/ groups
  - Iterative process



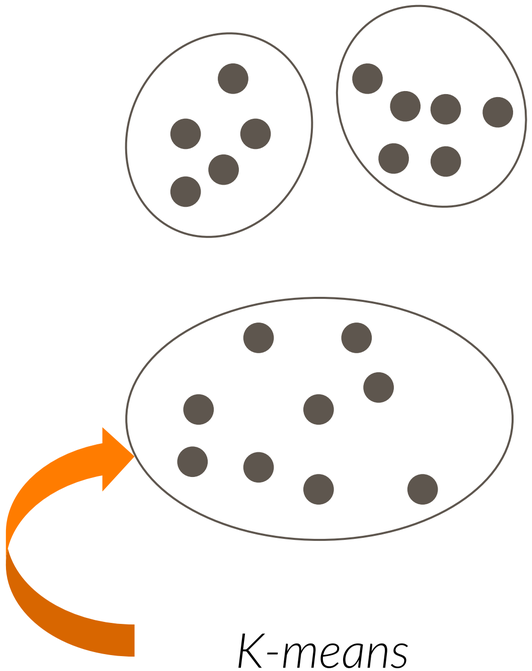
k-means vs. hierarchical clustering

- Partitioned clusters create singular subgroups, while hierarchical clusters tier the data points within a subgroup

Original Data

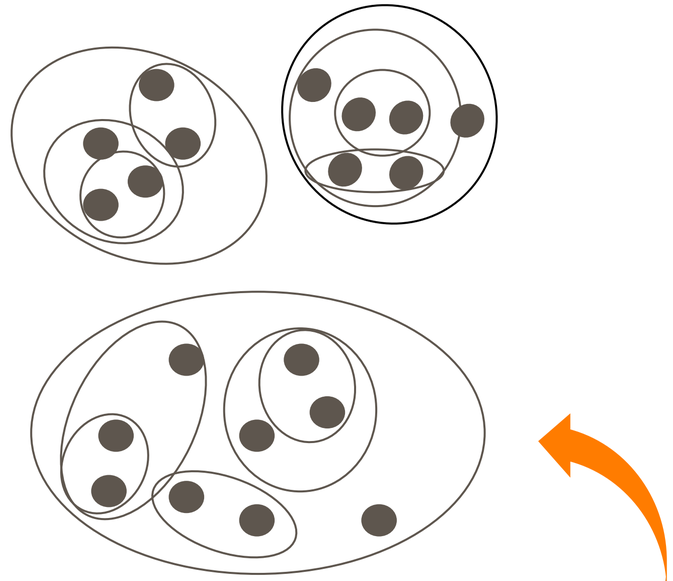


Partitioned Clusters



K-means

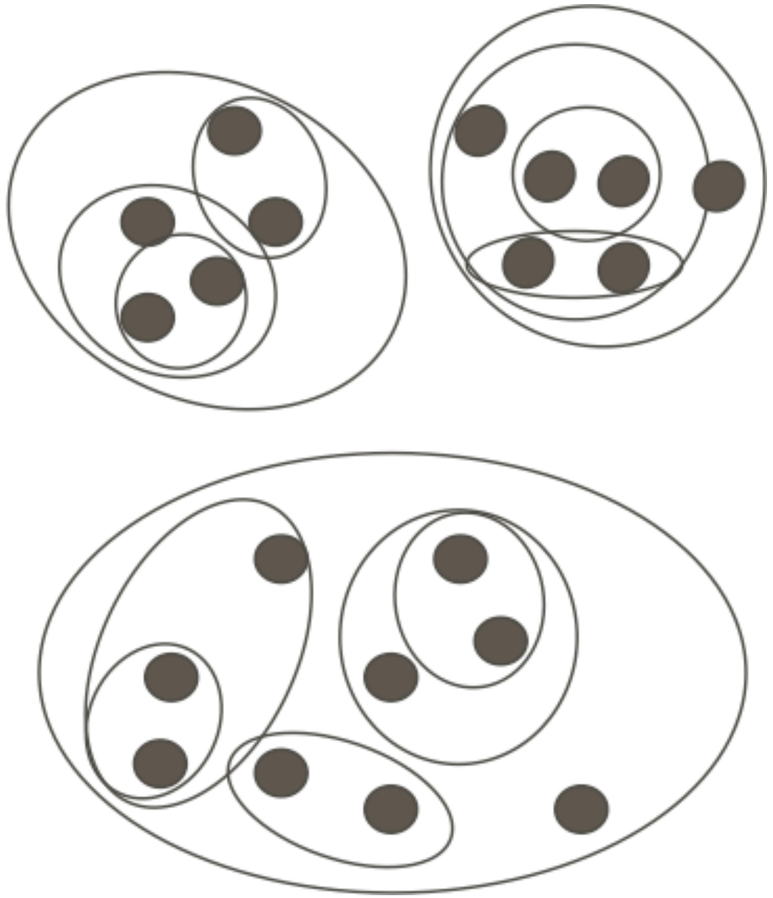
Hierarchical Clusters



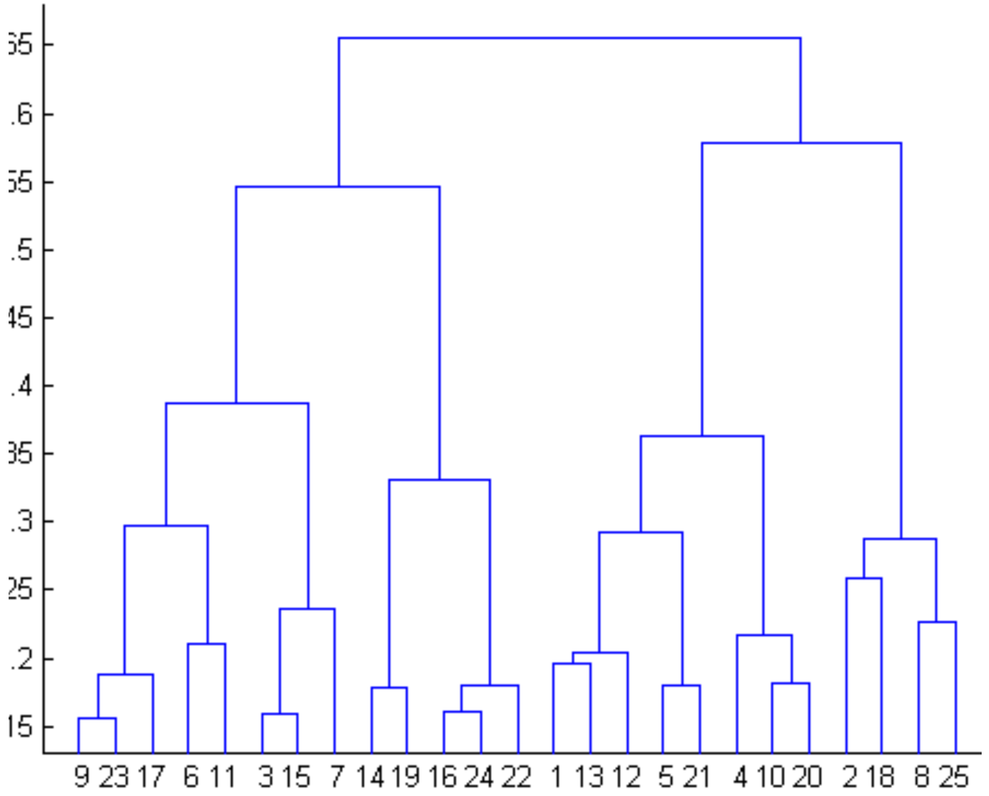
Hierarchical means subgroups exist within clusters

Hierarchical clustering: visualization

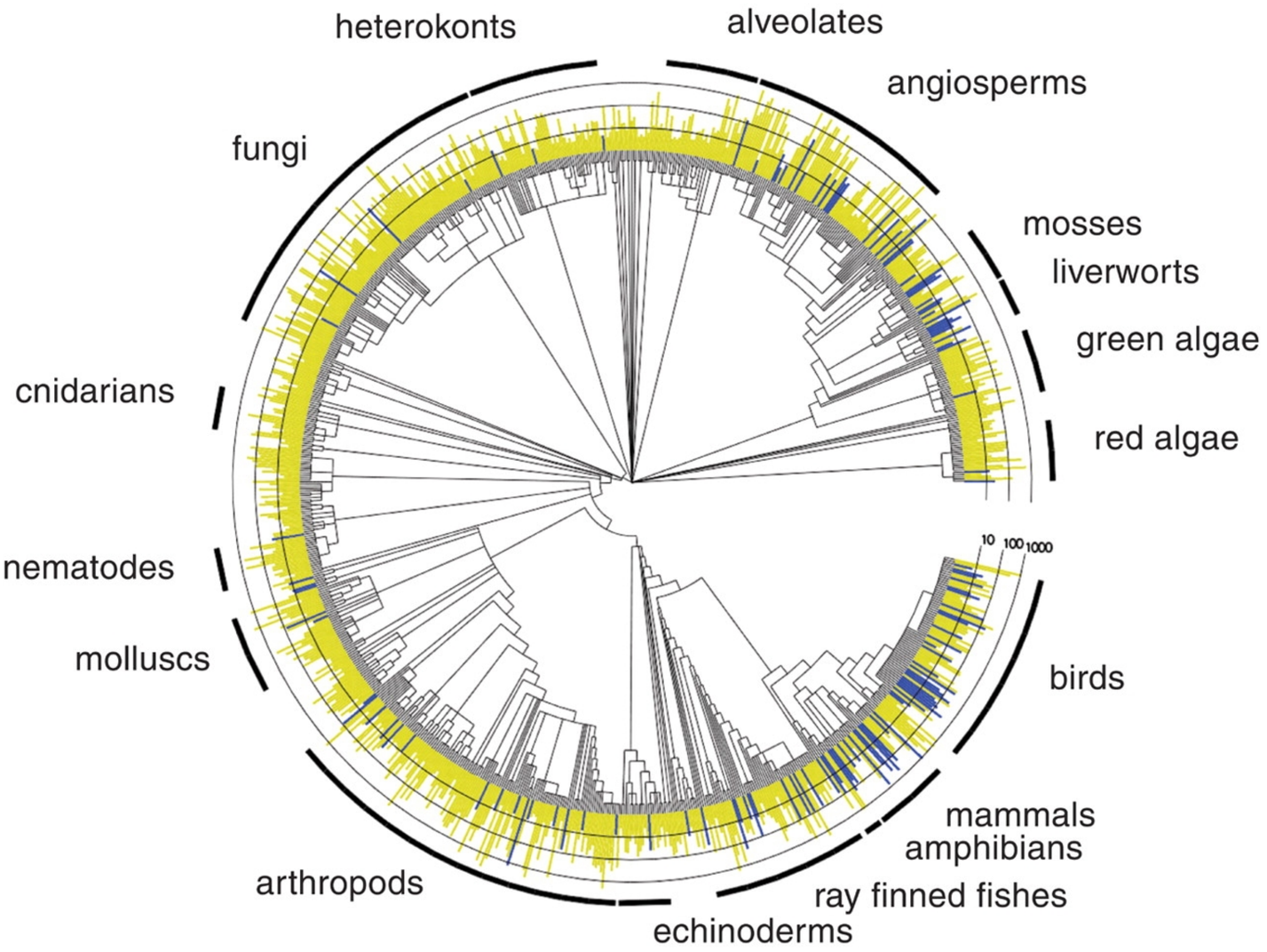
Scatterplot



Dendrogram



Circular dendrograms in biology



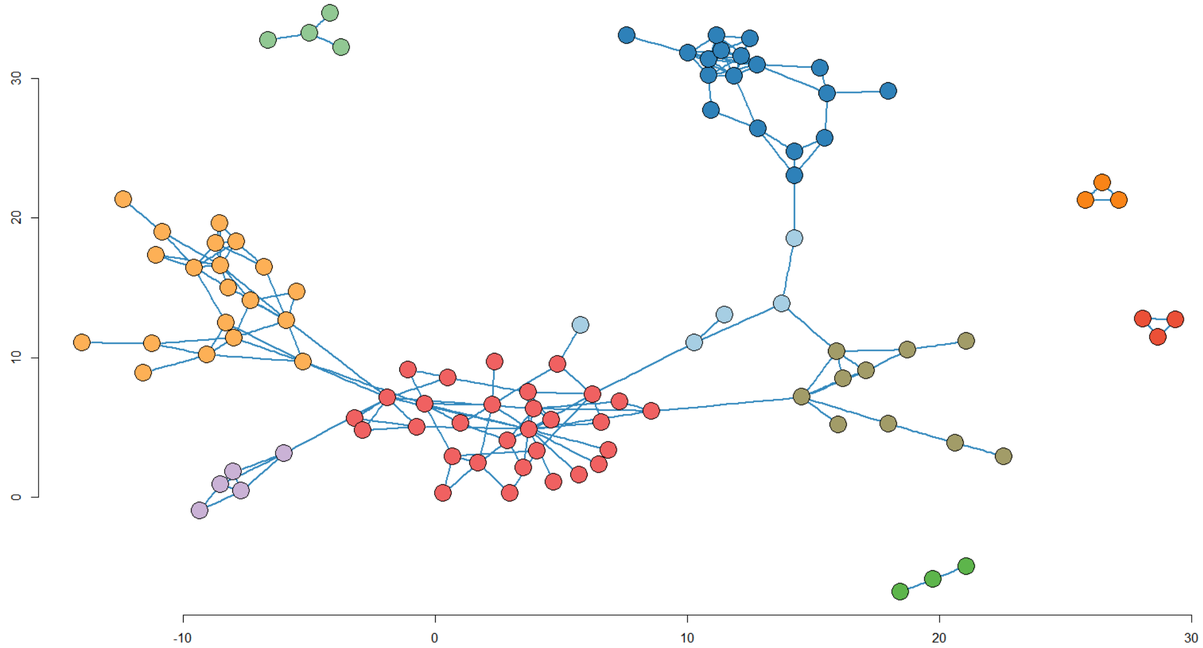
Considerations

1. What are the types of hierarchical clustering?

2. How do we measure distance between groups?

3. How do we select the number of clusters?

4. How do we validate clusters?



Overview

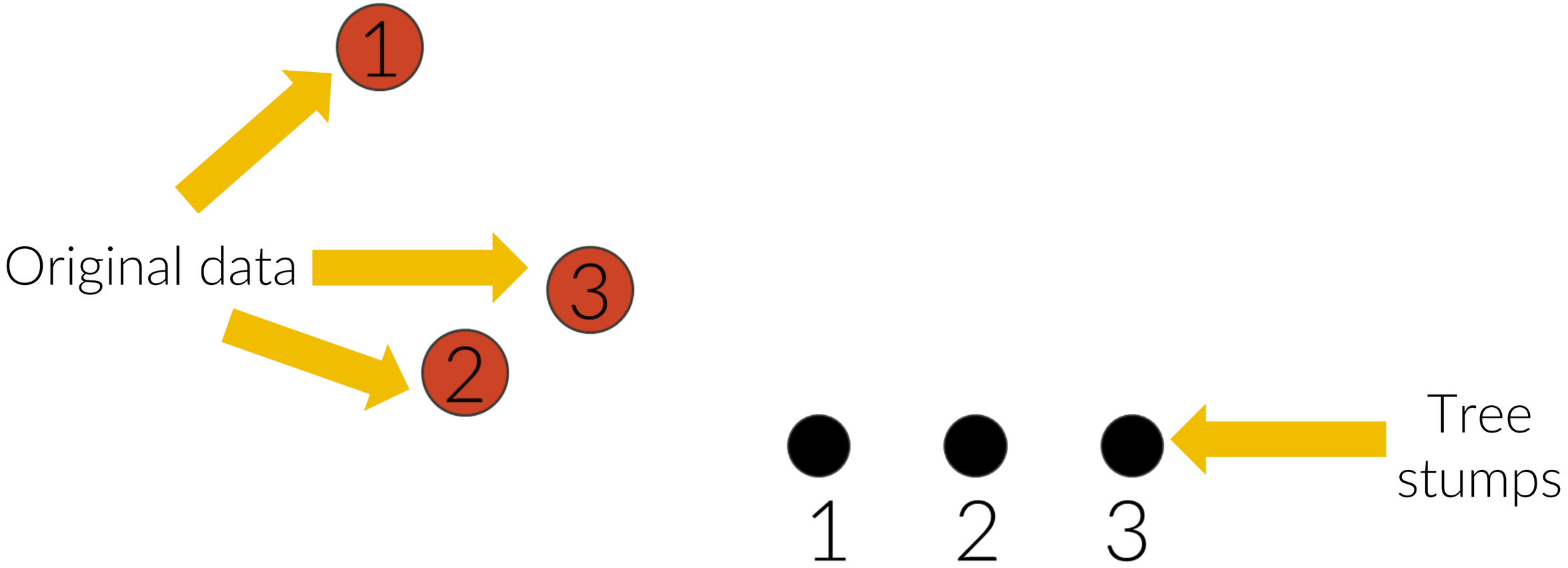
What types of clustering exist?

- Two types:
  - Agglomerative Nesting (Agnes)
  - Divisive Analysis (Diana)
- Results of clustering can be presented as a dendrogram
- Hierarchical clustering algorithms are greedy because they focus on short term optimization rather than optimal solutions

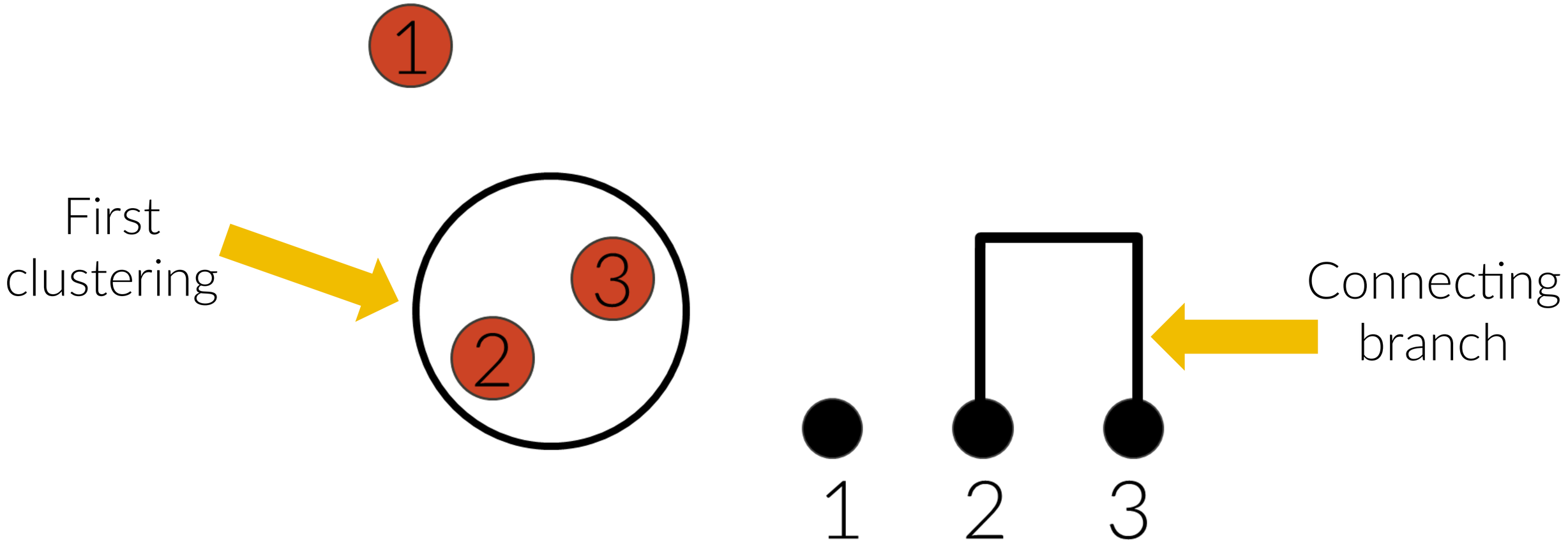
Some Notes

- Hierarchical clustering is deterministic, you will always get the same results no matter the starting points.
- In contrast, k-means is a randomized algorithm because it depends on the starting points

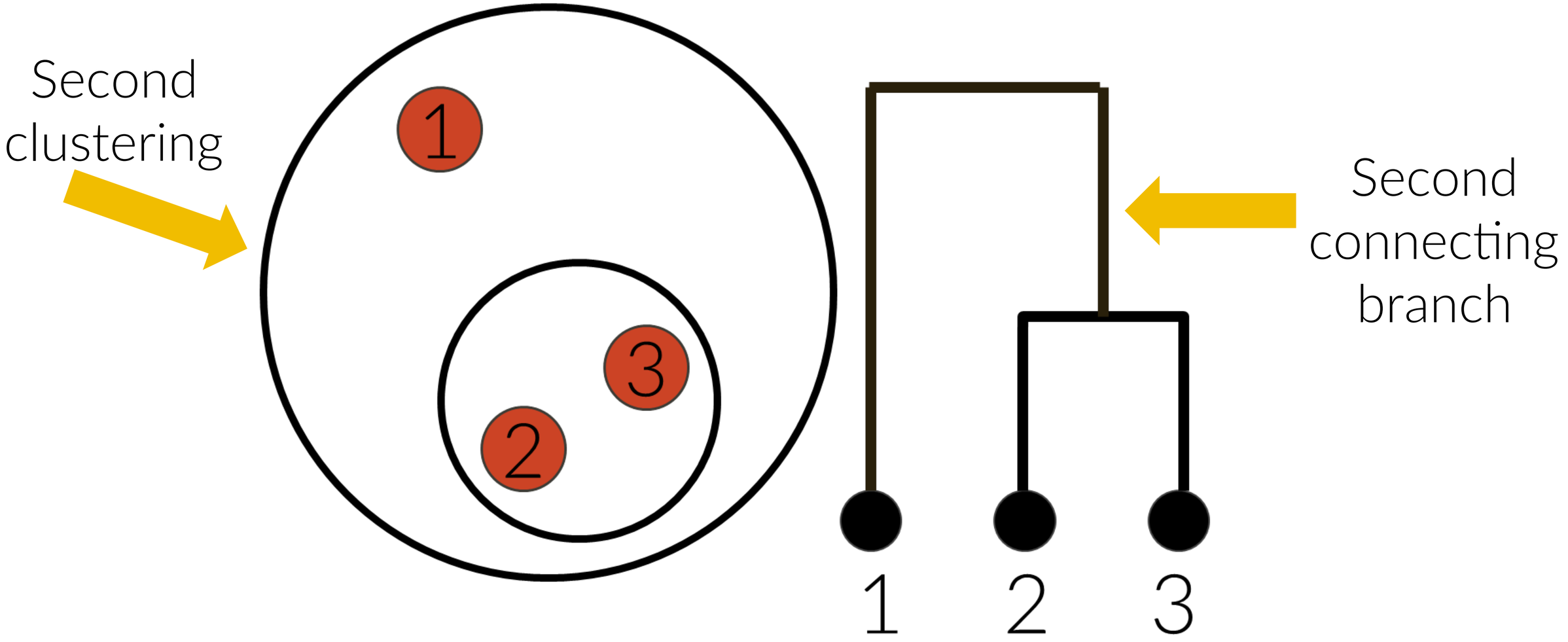
Agglomerative clustering



Agglomerative clustering

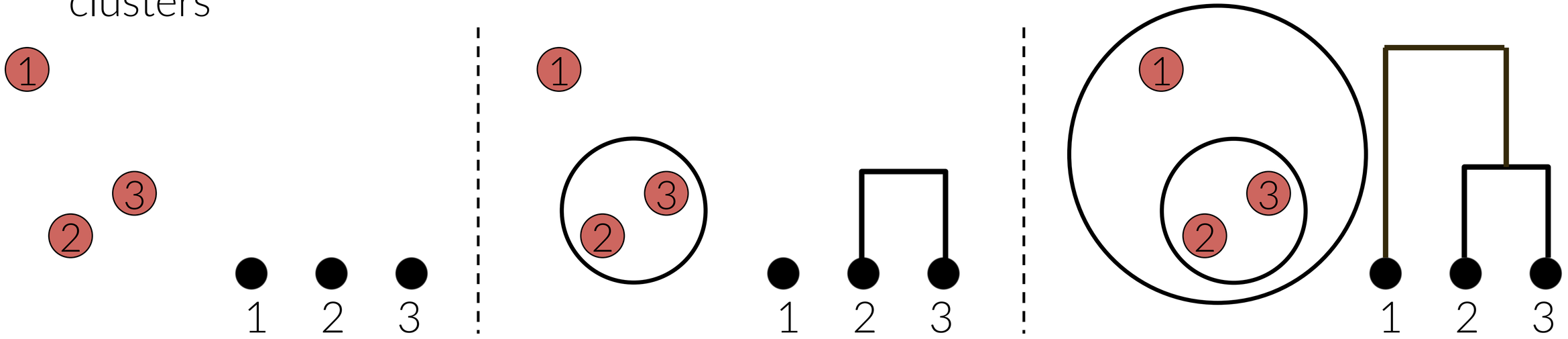


Agglomerative clustering



Agglomerative clustering (AGNES)

- *More popular hierarchal method*
- Bottom up approach
- Each observation starts in its own clusters
- Pairs are merged to form larger clusters
- Merging is based on similarity metric



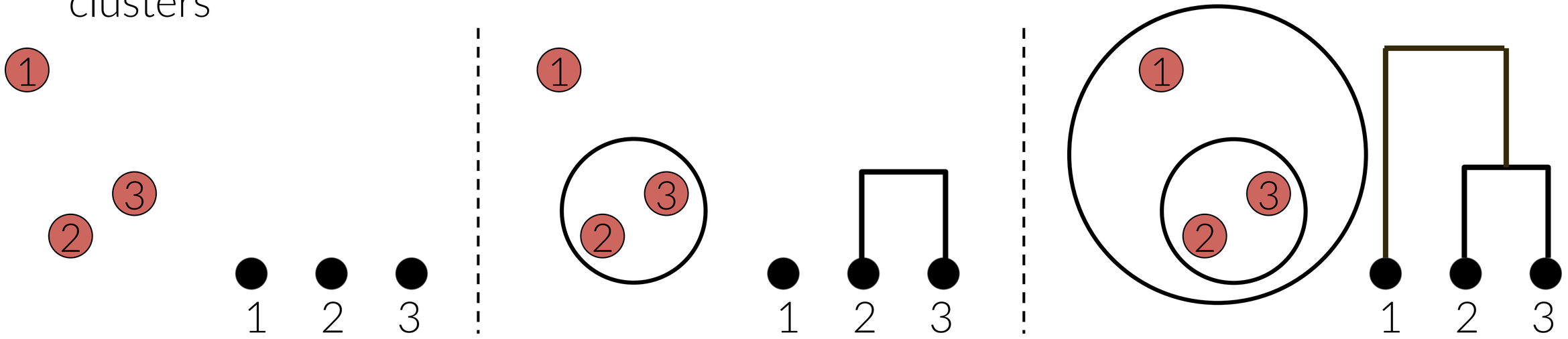
Original data

First cluster

Second cluster

Agglomerative clustering (AGNES)

- *More popular hierarchal method*
- Bottom up approach
- Each observation starts in its own clusters
- Pairs are merged to form larger clusters
- Merging is based on similarity metric



Original data

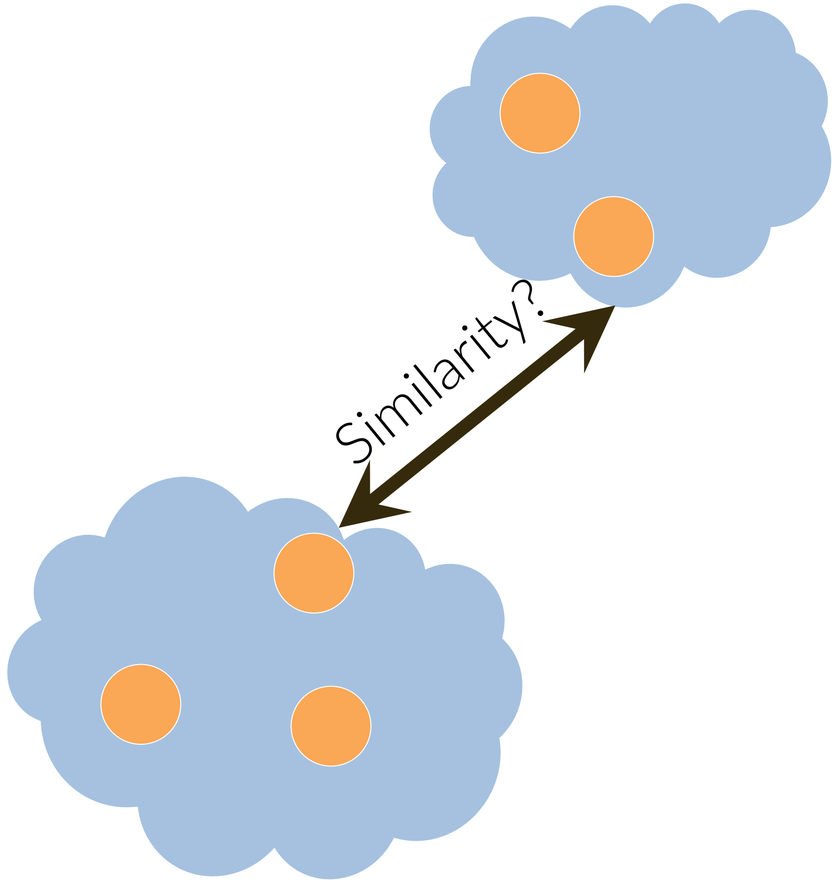
First cluster

Second cluster

What is closest?

There are several ways to measure inter-cluster closeness

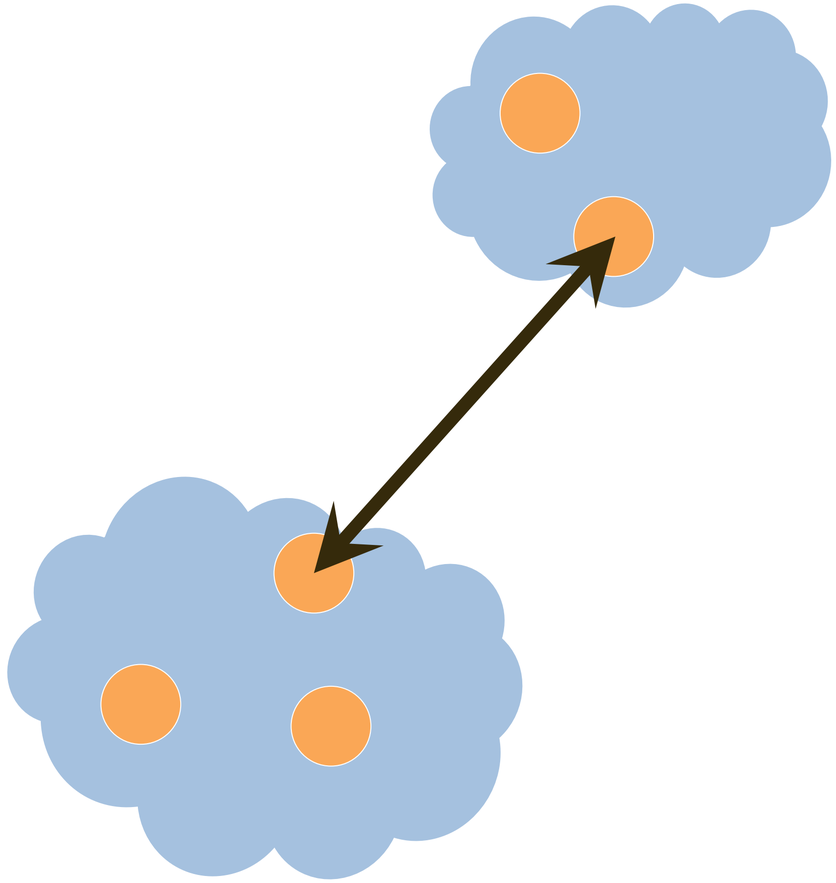
1. Minimum distance between points
2. Maximum distance between points
3. Group distance average
4. Distance between centroids



What is closest?

There are several ways to measure inter-cluster closeness

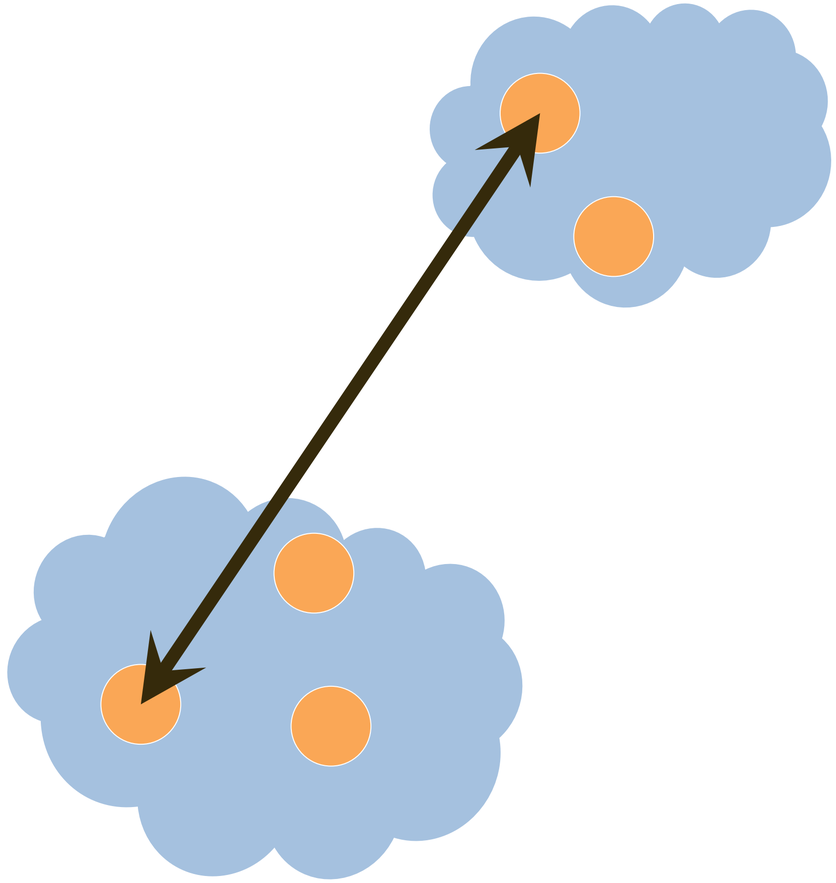
1. Minimum distance between points
2. Maximum distance between points
3. Group distance average
4. Distance between centroids



What is closest?

There are several ways to measure inter-cluster closeness

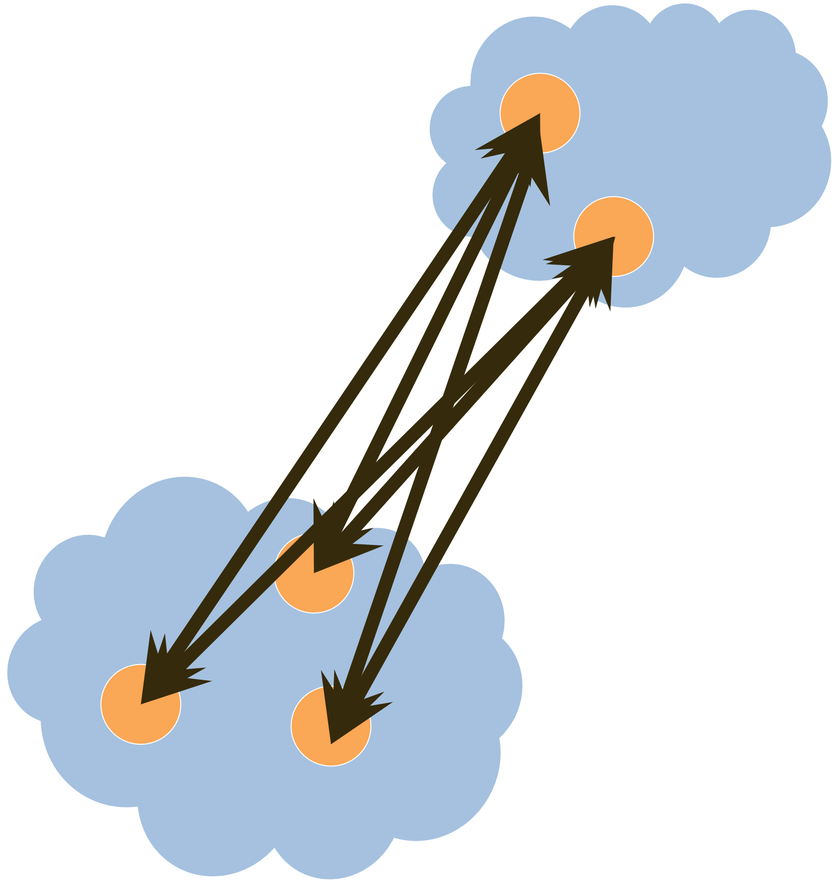
1. Minimum distance between points
2. Maximum distance between points
3. Group distance average
4. Distance between centroids



What is closest?

There are several ways to measure inter-cluster closeness

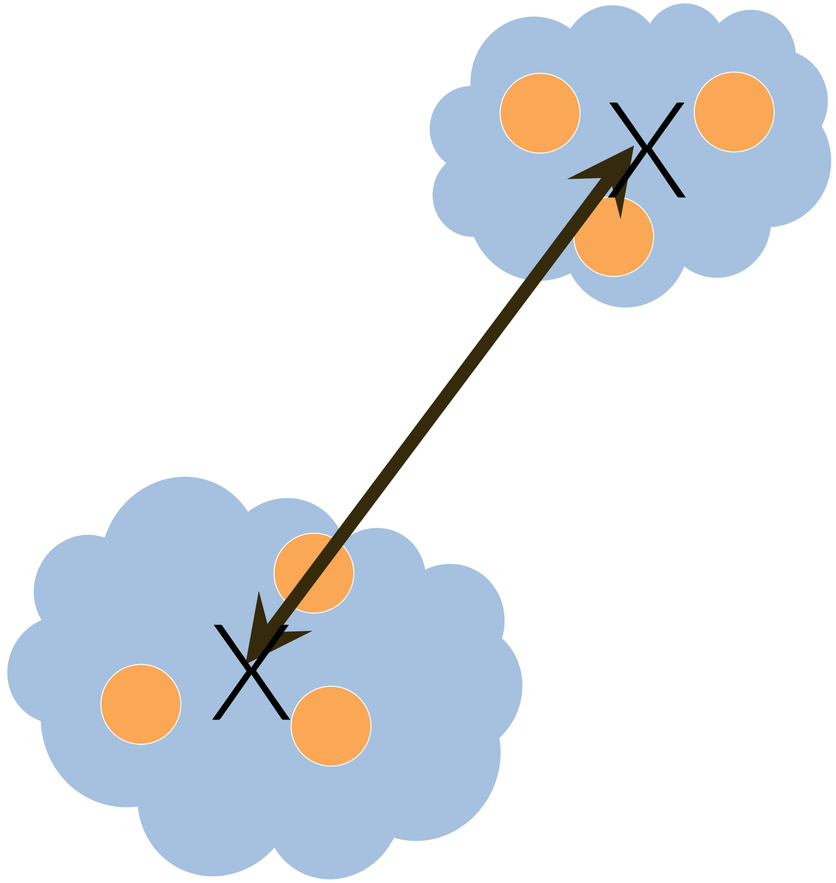
1. Minimum distance between points
2. Maximum distance between points
3. Group distance average
4. Distance between centroids



What is closest?

There are several ways to measure inter-cluster closeness

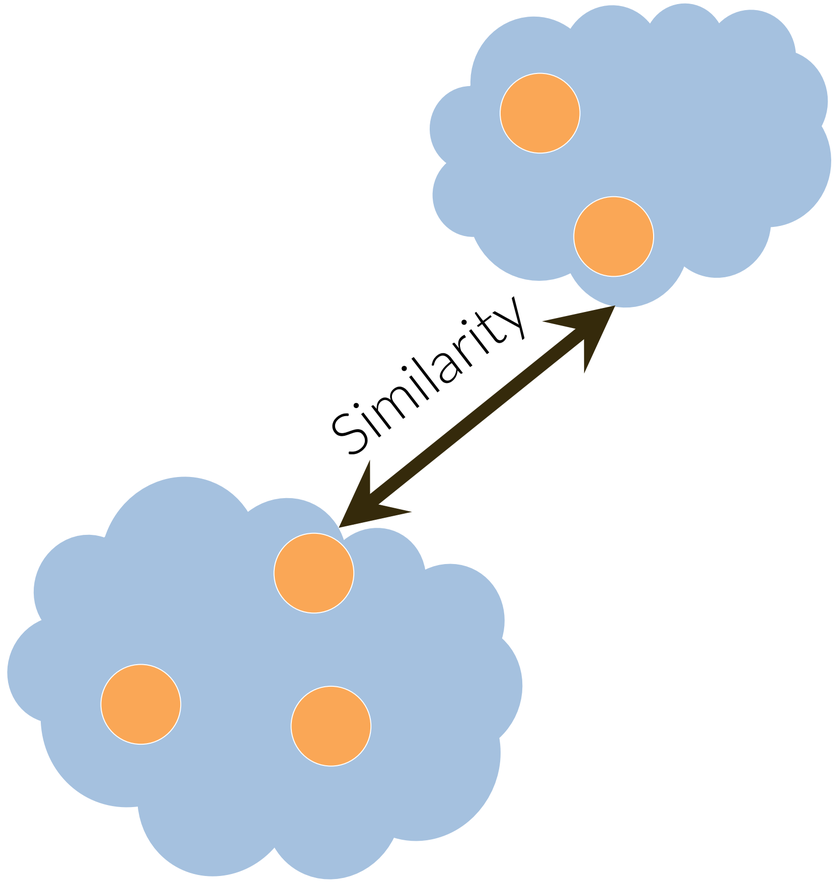
1. Minimum distance between points
2. Maximum distance between points
3. Group distance average
4. Distance between centroids



What is closest?

There are several ways to measure inter-cluster closeness

1. Minimum distance between points
   - Also known as: "single linkage"
2. Maximum distance between points
   - Also known as: "complete linkage"
3. Group distance average
   - Also known as: "average linkage"
4. Distance between centroids



AGNES: implementation in Python

- First, we build a sample data frame and distance matrix based on that data frame

Script

```
import pandas as pd
np.random.seed(123)

variables = ['X', 'Y', 'Z']
labels = ['ID_0', 'ID_1', 'ID_2', 'ID_3', 'ID_4']

# We create a random table with features X,Y and Z.
# The rows are the IDs and for each ID there is feature X,Y and Z.

X = np.random.random_sample([5, 3]) * 10
df = pd.DataFrame(X, columns = variables,
                  index = labels)
df
```

```
    ...: df
Out[58]:
           X         Y         Z
ID_0  6.964692  2.861393  2.268515
ID_1  5.513148  7.194690  4.231065
ID_2  9.807642  6.848297  4.809319
ID_3  3.921175  3.431780  7.290497
ID_4  4.385722  0.596779  3.980443
```

AGNES: implementation in Python

- To calculate the distance matrix as input for hierarchical clustering, we will use the pdist function from SciPy's spatial.distance submodule.
- We are going to calculate the Euclidean distance between each pair of sample points in our dataset based on features X, Y and Z.

Script

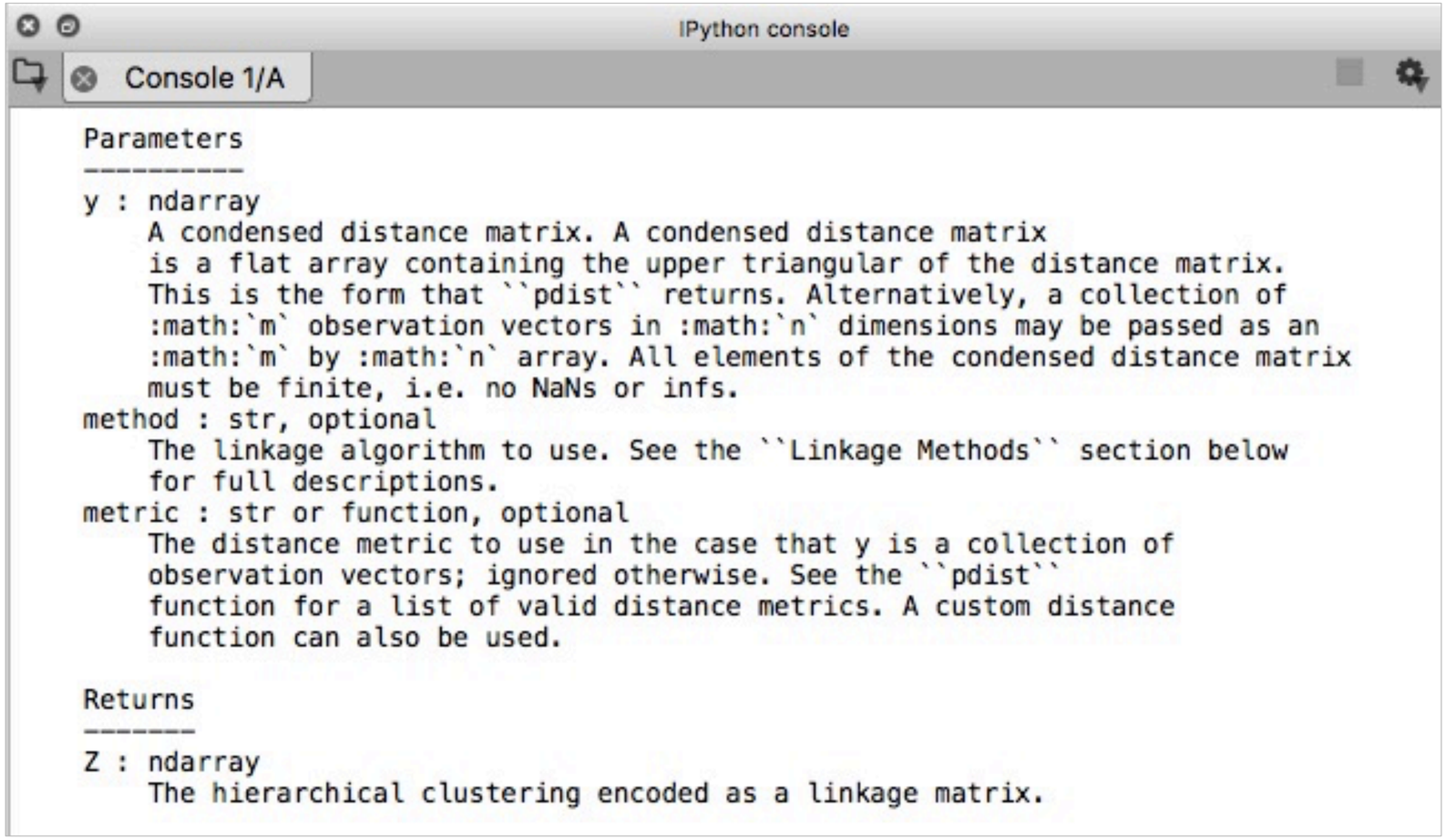
```
from scipy.spatial.distance import pdist, squareform
row_dist = pd.DataFrame(squareform(pdist(df, metric = 'euclidean')),
                                   columns = labels,
                                   index = labels)


# We provide the condensed distance matrix -
# returned by pdist - as input. The
# squareform function creates a symmetrical
# matrix of pair-wise distances.
row_dist
```

```
In [60]: row_dist
Out[60]:
          ID_0      ID_1      ID_2      ID_3      ID_4
ID_0  0.000000  4.973534  5.516653  5.899885  3.835396
ID_1  4.973534  0.000000  4.347073  5.104311  6.698233
ID_2  5.516653  4.347073  0.000000  7.244262  8.316594
ID_3  5.899885  5.104311  7.244262  0.000000  4.382864
ID_4  3.835396  6.698233  8.316594  4.382864  0.000000
```

AGNES: implementation in Python

Script

```
from scipy.cluster.hierarchy import linkage
help(linkage)
```



AGNES: implementation in Python

- We can either pass a condensed distance matrix (upper triangular) from the pdist function, or we can pass the "original" data array and define the metric as 'euclidean' in linkage.
- However, we should not pass the squareform distance matrix, which would yield different distance values although the overall clustering could be the same.

Script

```
# Incorrect approach: Squareform distance matrix
row_clusters = linkage(row_dist,
                       method = 'complete', metric = 'euclidean')
pd.DataFrame(row_clusters,
             columns = ['row label 1', 'row label 2',
                        'distance', 'no. of items in clust.'],
             index = ['cluster %d' % (i + 1)
                      for i in range(row_clusters.shape[0])])
```

This will yield an error: "ClusterWarning"

AGNES: implementation in Python

Script

```
# Correct approach: Condensed distance matrix
row_clusters = linkage(pdist(df, metric = 'euclidean'),
                       method = 'complete')
pd.DataFrame(row_clusters,
             columns = ['row label 1', 'row label 2',
                        'distance', 'no. of items in clust.'],
             index = ['cluster %d' % (i + 1)
                      for i in range(row_clusters.shape[0])])
```

IPython console

Console 1/A

Out[63]:

	row label 1	row label 2	distance	no. of items in clust.
cluster 1	0.0	4.0	3.835396	2.0
cluster 2	1.0	2.0	4.347073	2.0
cluster 3	3.0	5.0	5.899885	3.0
cluster 4	6.0	7.0	8.316594	5.0

AGNES: implementation in Python

Script

```
# Correct approach: Input sample matrix
row_clusters = linkage(df.values, method = 'complete',
                       metric = 'euclidean')
pd.DataFrame(row_clusters,
             columns = ['row label 1', 'row label 2',
                        'distance', 'no. of items in clust.'],
             index = ['cluster %d' % (i + 1)
                      for i in range(row_clusters.shape[0])])
```

IPython console

Console 1/A

```
In [68]: clusters
Out[68]:
           row label 1  row label 2  distance  no. of items in clust.
cluster 1          0.0          4.0  3.835396                     2.0
cluster 2          1.0          2.0  4.347073                     2.0
cluster 3          3.0          5.0  5.899885                     3.0
cluster 4          6.0          7.0  8.316594                     5.0
In [69]:
```

AGNES: implementation in Python

Script

```
# We can refer to each column from the matrix:
clusters[['row label 1', 'row label 2']]
# These two columns denote the most dissimilar members in each cluster

clusters[['distance']]
# The third column represents the distance between the
# dissimilar members

clusters[['no. of items in clust.']]
# The last column returns the count of the members in each cluster
```

IPython console

Console 1/A

```
In [69]: clusters[['row label 1','row label 2']]
Out[69]:
           row label 1  row label 2
cluster 1          0.0          4.0
cluster 2          1.0          2.0
cluster 3          3.0          5.0
cluster 4          6.0          7.0
```

AGNES: implementation in Python

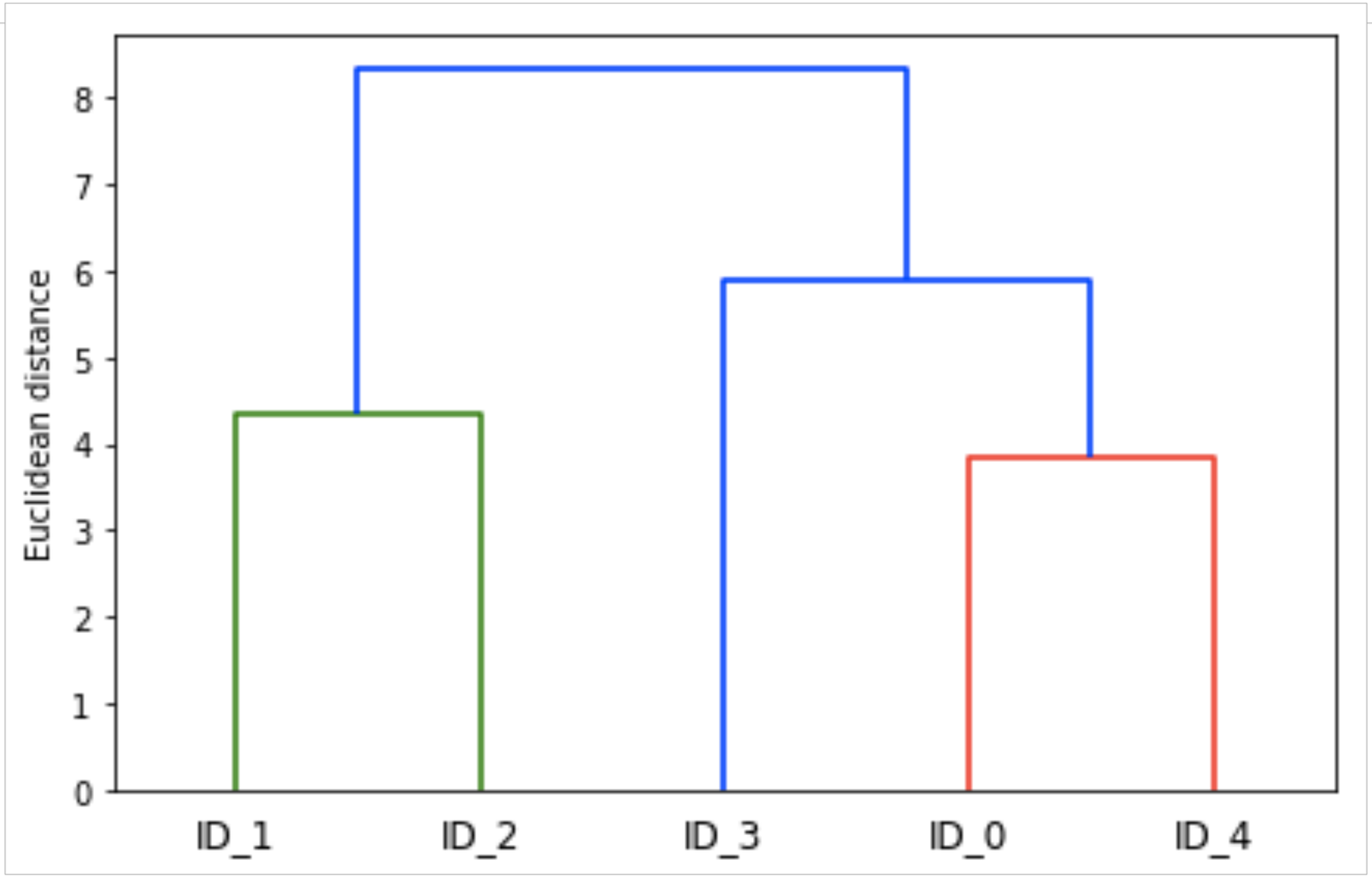
Script

```
# Let's visualize the results as a dendrogram.
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram

# make dendrogram black (part 1/2)
# from scipy.cluster.hierarchy import set_link_color_palette
# set_link_color_palette(['black'])

row_dendr = dendrogram(row_clusters,
                       labels = labels,
                       # make dendrogram black (part 2/2)
                       # color_threshold = np.inf
                       )
plt.tight_layout()
plt.ylabel('Euclidean distance')
# plt.savefig('images/11_11.png', dpi = 300,
#             bbox_inches = 'tight')
```

AGNES: implementation in Python

```
plt.show()
```

Script

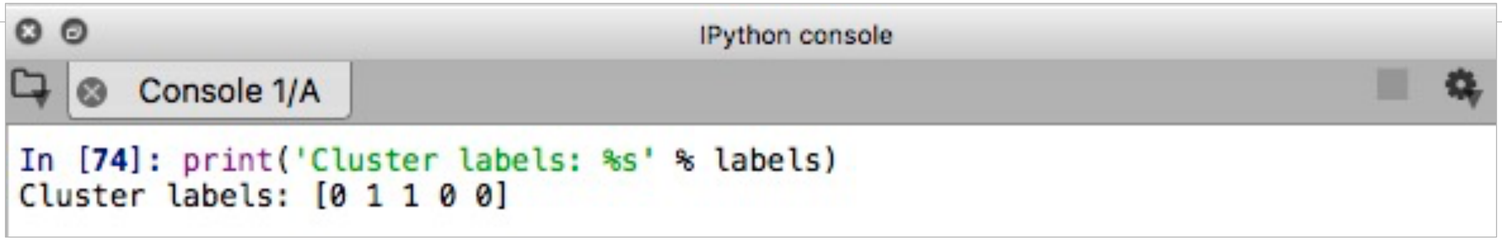


AGNES: implementation in Python

- There is also an Agglomerative Clustering implementation in scikit-learn which allows us to choose the number of clusters that we want to return.
- This is useful if we want to prune the hierarchical cluster tree.

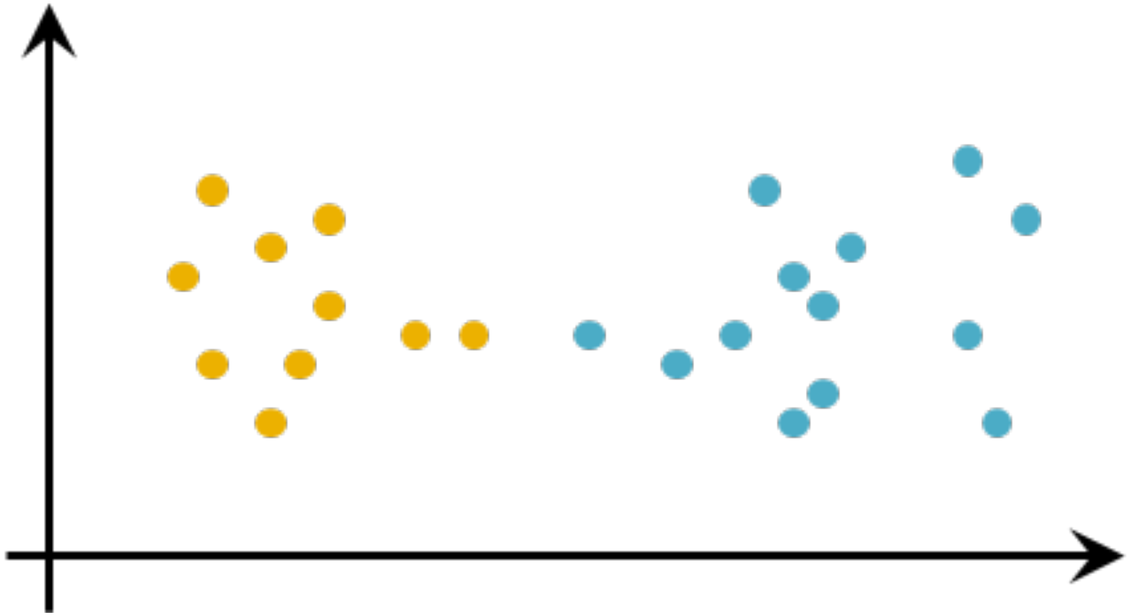
Script

```
# Using scikit-learn to implement AGNES
from sklearn.cluster import AgglomerativeClustering

ac = AgglomerativeClustering(n_clusters = 2,
                             affinity = 'euclidean',
                             linkage = 'complete')
labels = ac.fit_predict(X)
print('Cluster labels: %s' % labels)
```

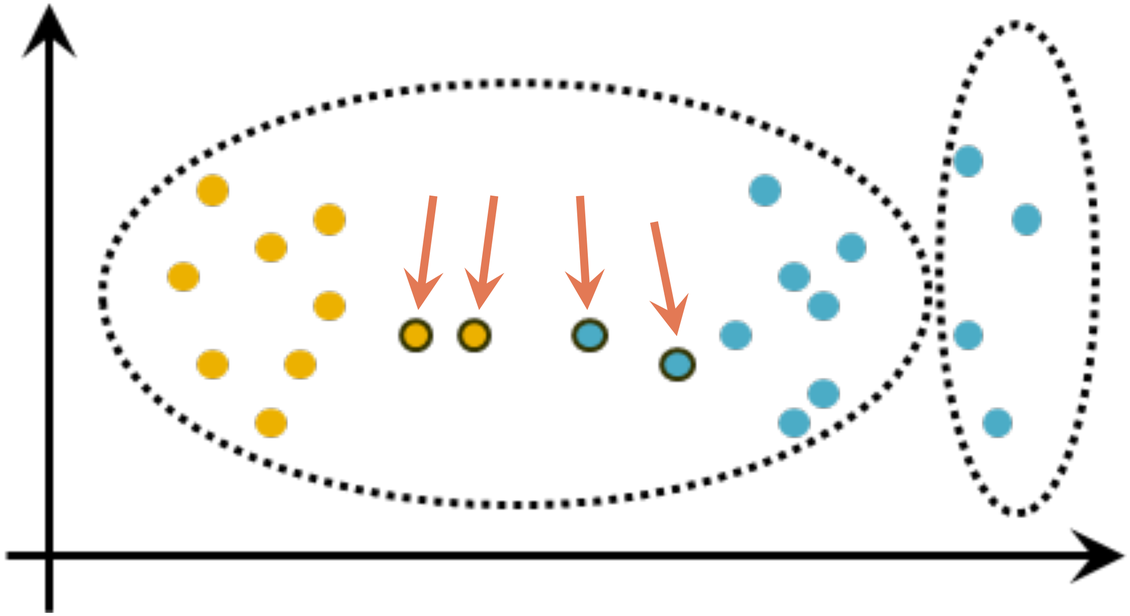


4 common problems

Problem 1: Single linkage (minimum distance between points) clustering analysis may produce a chain effect



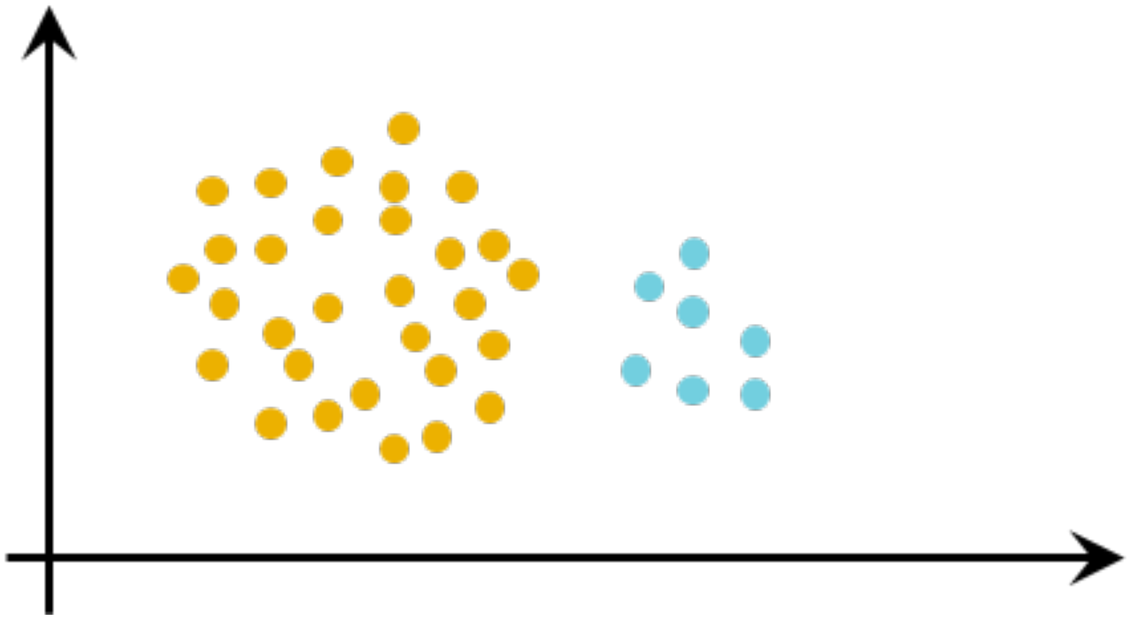
Original data



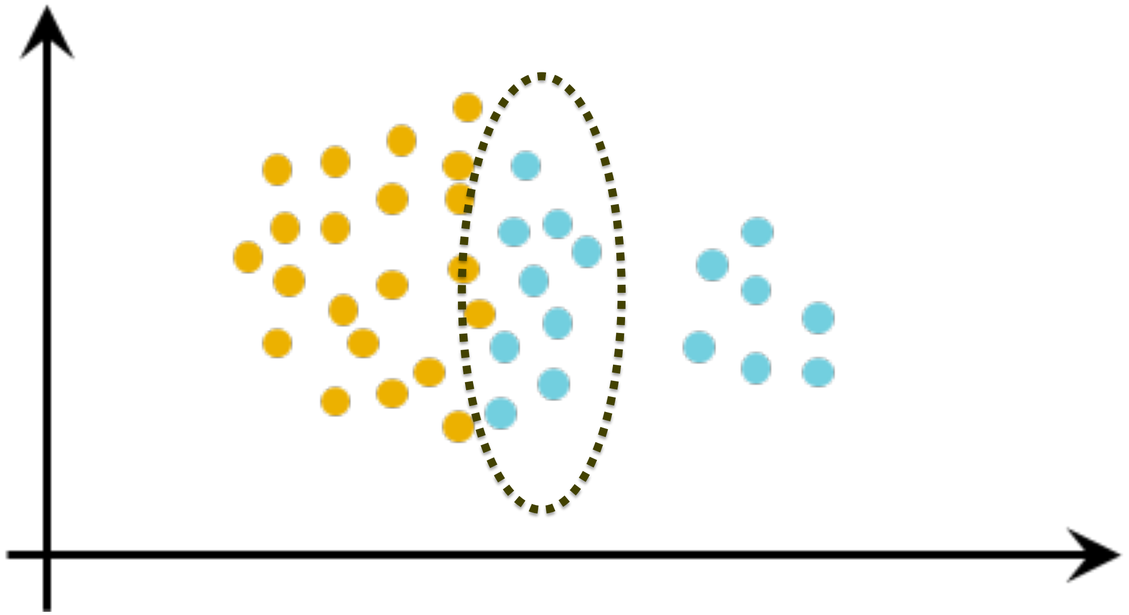
Chain effect

4 common problems

Problem 2: Complete linkage (maximum distance between points) clustering analysis break large clusters



Original data



Breaks large clusters

4 common problems

Problem 3: amount of *space* required

Space required is n^2 (where n is the number of points)

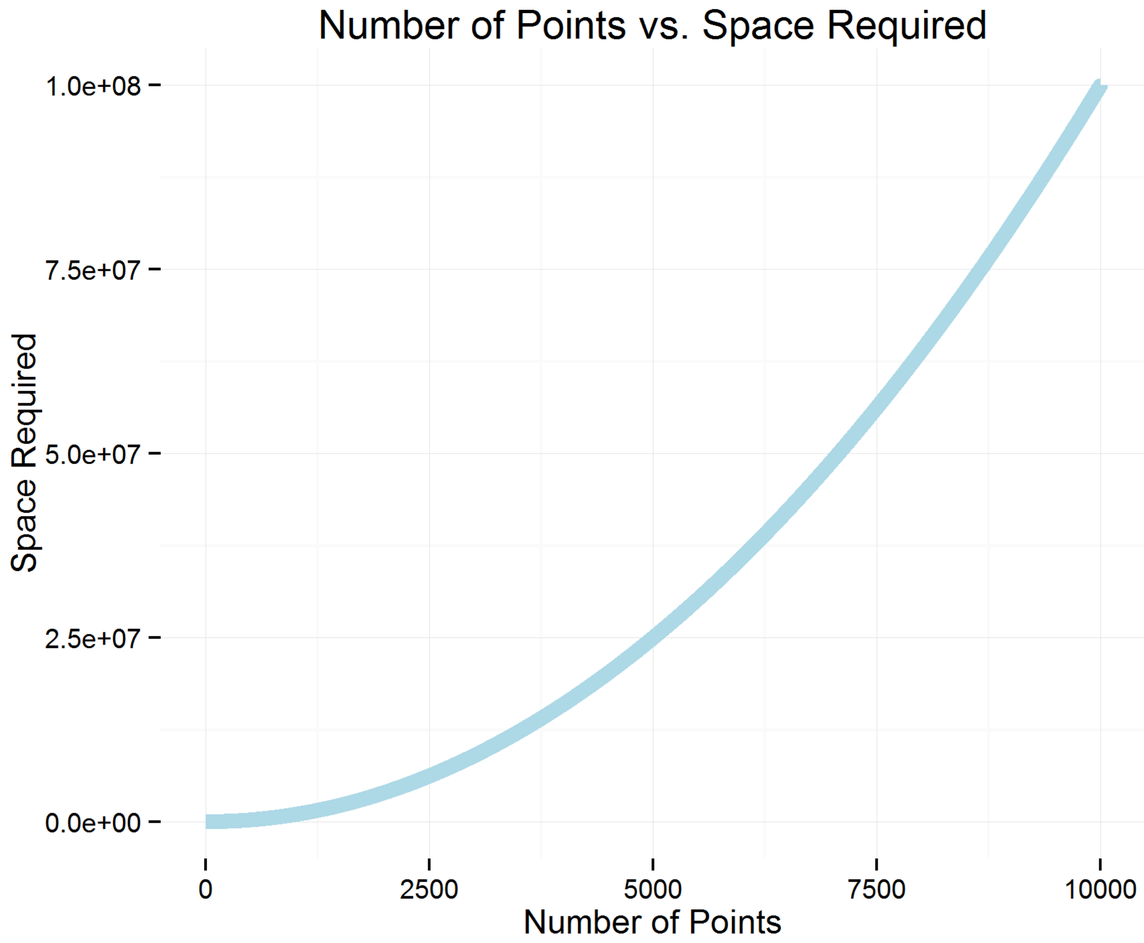
Example:

1,000 data points would require:

1,000 x 1,000 = 1,000,000 cell matrix
(61.04 MB)

10,000 data points would require:

10,000 x 10,000 = 100,000,000 cell matrix
(6,104 MB)



4 common problems

Problem 4: amount of *time* required

Time required to run AGNES is n^3
(where n is the number of points)

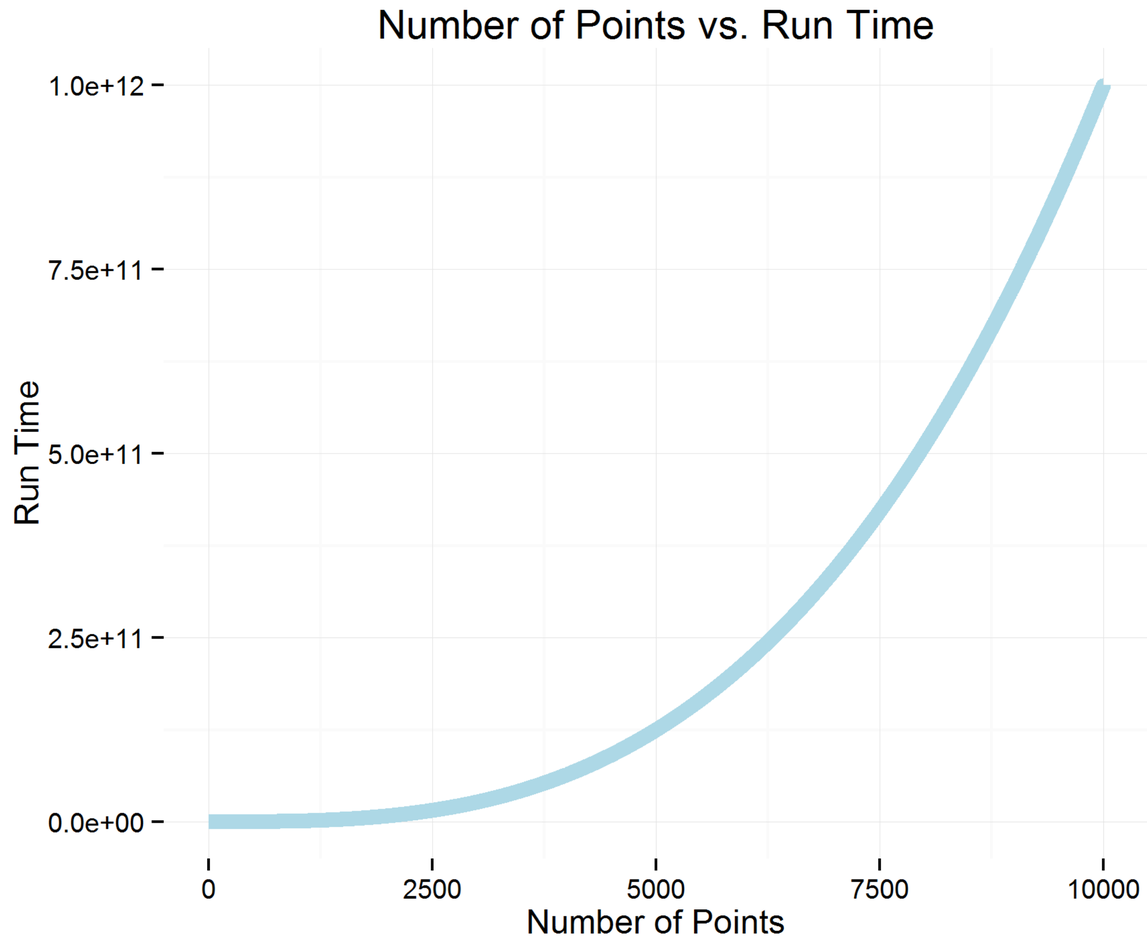
Example:

1,000 data points would require:

.05 seconds

10,000 data points would require:

.05 x 10 x 10 x 10 = 50 seconds



Exercise time!