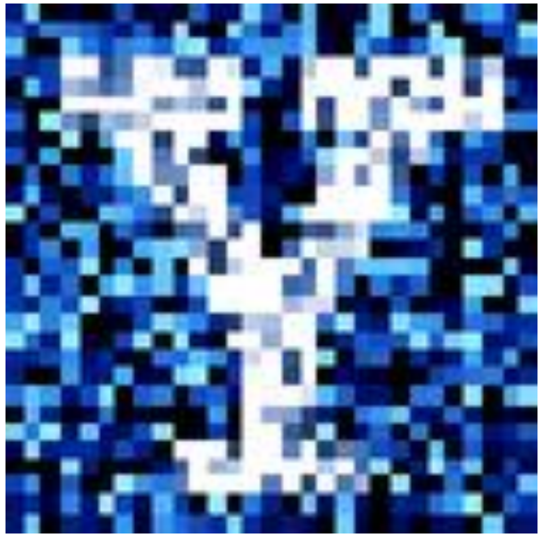
YData: Introduction to Data Science



Lecture 25: Unsupervised learning

Overview

Linear regression continued

Unsupervised learning/clustering

- K-means cluster
- Hierarchical clustering

Brief discussion of LLMs

Project timeline

~~Sunday, November 16th~~

- ~~Projects are due on Gradescope at 11pm~~
- ~~Email a **pdf** of your project to your peer reviewers~~
  - ~~A list of whose paper you will review is posted on Canvas~~
  - ~~Fill out the draft reflection on Canvas~~

~~Sunday, November 23rd~~

- ~~A template for doing your review has been posted~~
- ~~Jupyter notebook files with your reviews need to be emailed to the authors~~
- ~~A pdf containing all three reviews needs to be uploaded Gradescope~~

Sunday, December 7th

- Project is due on Gradescope
- Add the peer reviews of your project to the Appendix of your project
- Be sure to fill out the final project/class reflection **on Canvas**

Announcement

Exam review session: Tuesday December 9th from 2:45-3:45pm in this room

Final exam: Monday December 17th at 2pm

Linear regression continued

Regression

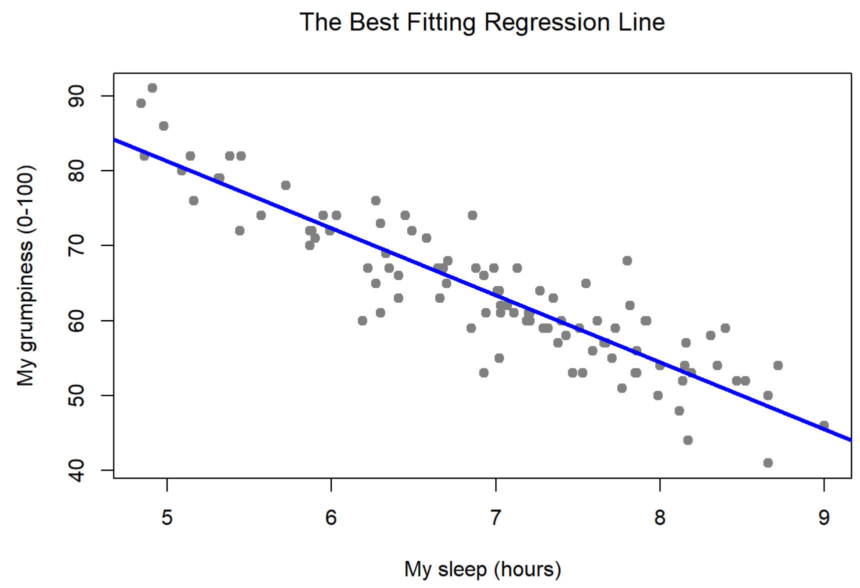
Regression is method of using one variable ***x*** ***to predict*** the value of a quantitative variable ***y***

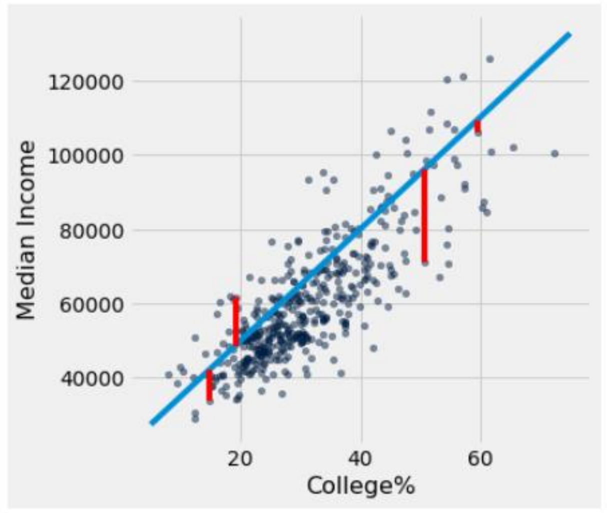
- i.e., $\hat{y} = f(x)$
- Linear regression: $\hat{y} = \text{intercept} + \text{slope} \cdot x$

  $\hat{y} = b_0 + b_1 \cdot x$

The coefficients for these regression models are found by minimizing root mean square error (RMSE)

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$



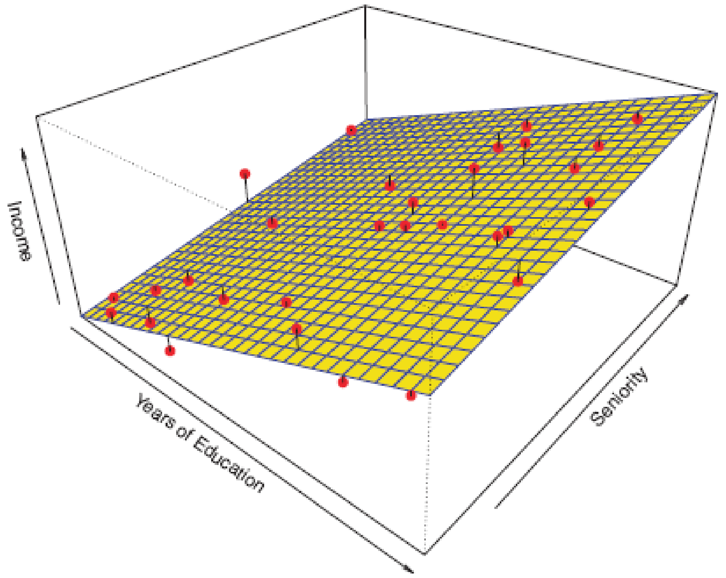


Regression line app

Multiple regression

In multiple regression we try to predict a quantitative response variable *y* using several features $x_1, x_2, \dots, x_k$

We estimate coefficients using a data set to make predictions ŷ



$$\hat{y} = b_0 + \underset{\uparrow}{b_1 \cdot x_1 + b_2} \cdot x_2 + \dots + b_k \cdot x_k$$

Learn the b_i's on the training set. Assess prediction accuracy on test set.

Linear regression models in scikit-learn

We can use scikit-learn to create linear regression models

- You can also use the stats models package to do this

```
linear_model = LinearRegression()  # construct a linear regression model

linear_model.fit(X_train_features, y_train)      # train the classifier
```

"Learns" b_0, b_1, … by minimizing the RMSE on the training data

Numeric values to predict

```
y_preditions = linear_model.predict(X_test_features)   # make predictions

RMSE = np.sqrt(np.mean((y_test - y_predictions)**2))   # get the RMSE
```

Real world example

Last class we predicted the mass of penguins!

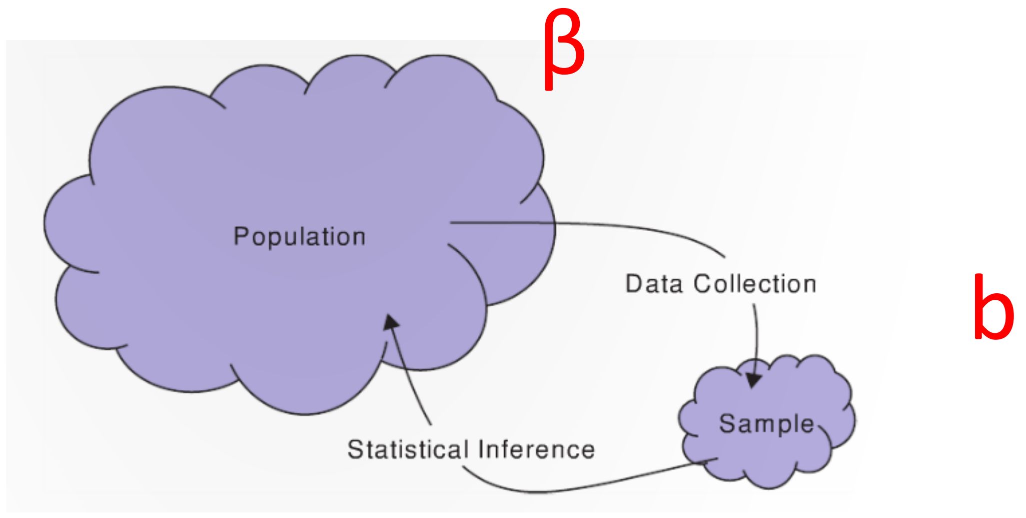
Let’s quickly review the code in Jupyter!

Inference for simple linear regression

Inference for simple linear regression

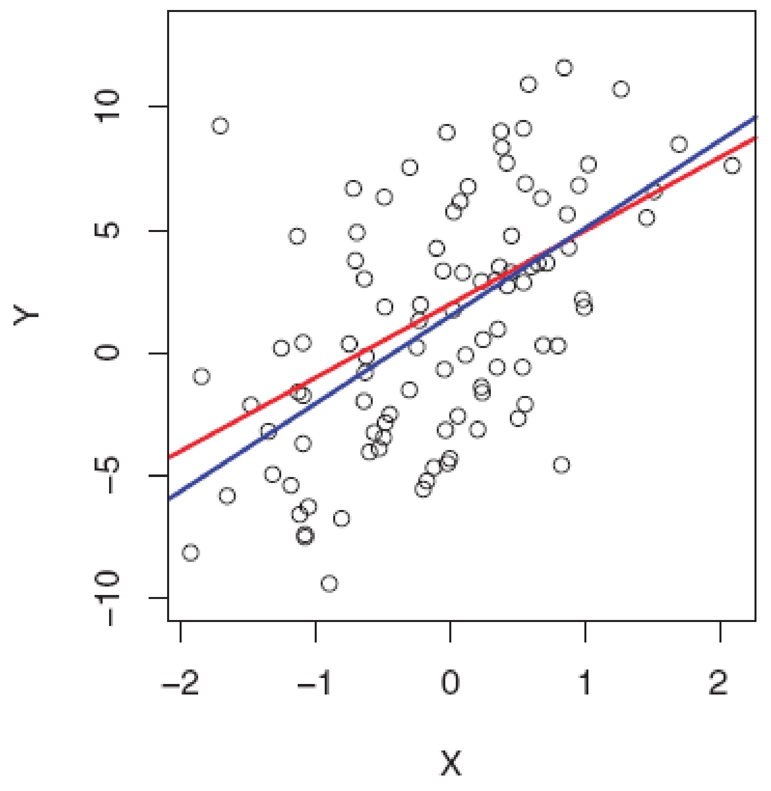
The Greek letter **β** is used to denote the slope ***of the population***

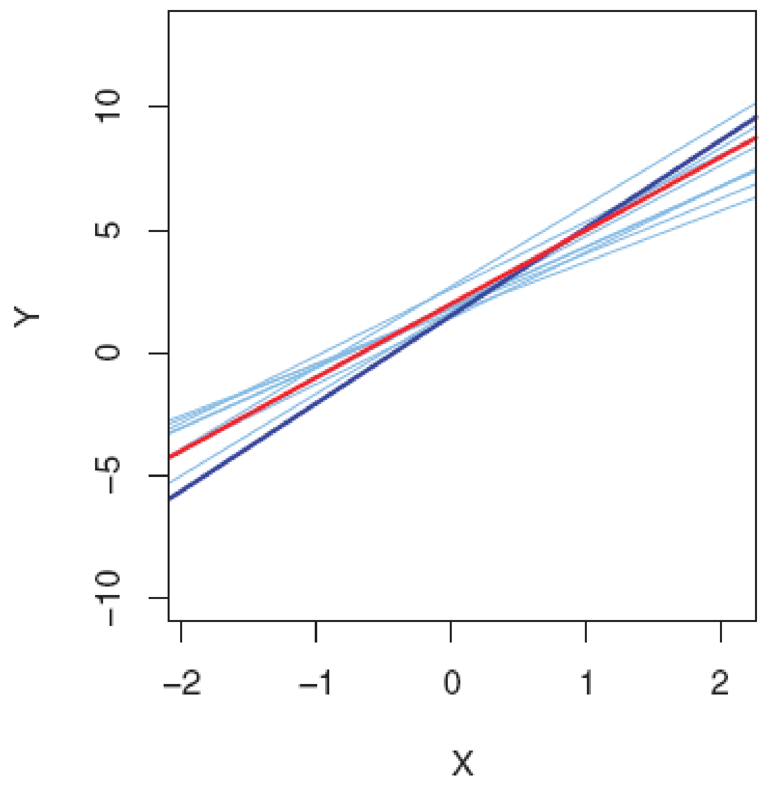
The letter **b** is used to denote the slope ***of the sample***



Population: β_o and β_1

Sample estimates: b_0 and b_1

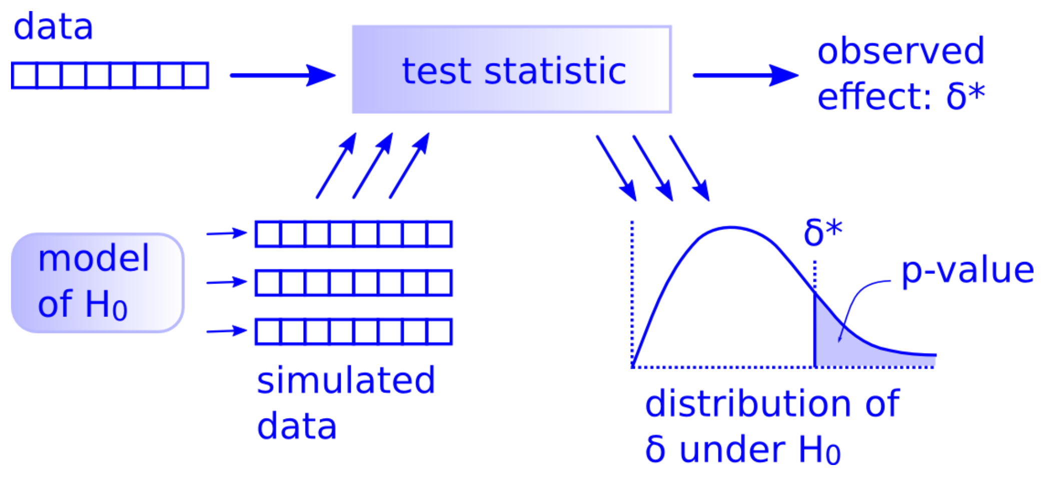


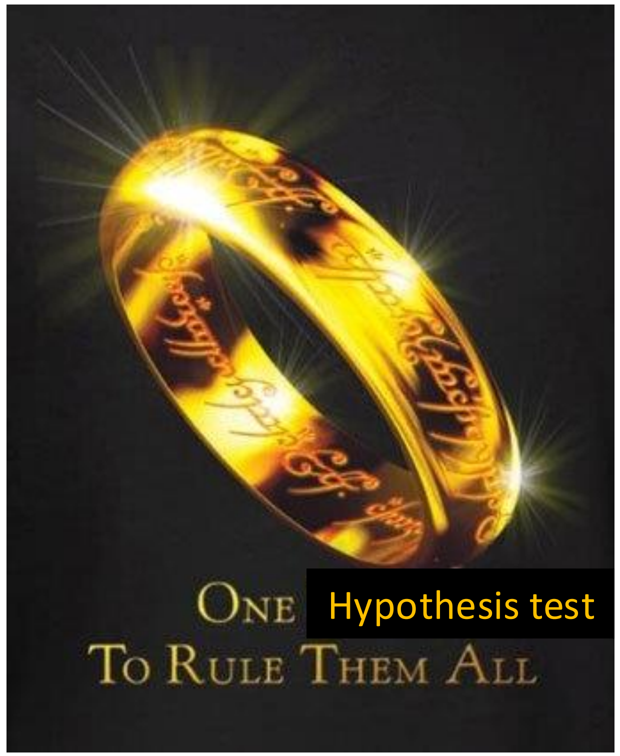


Hypothesis test for regression coefficients

We can run a hypothesis test to see if there is a linear relationship between a feature x, and a response variable y

There is only one hypothesis test!





Hypothesis test for regression coefficients

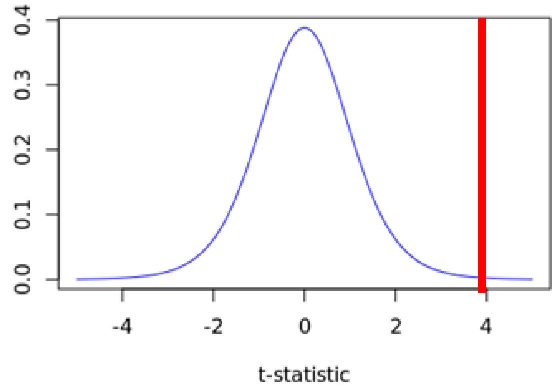
We can run a hypothesis test to see if there is a linear relationship between a feature x, and a response variable y

- H_0: $\beta_1 = 0$ (slope is 0, so no linear relationship between x and y
- H_A: $\beta_1 \neq 0$

One type of hypothesis test we can run is based on a t-statistic: $t = \frac{b_1 - 0}{\hat{SE}_{b_1}}$

- The t-statistic comes from a t-distribution with n - k degrees of freedom
  - (If a few conditions are met)

We could also shuffle our data, fit models, and extract b_1 coefficients to create a null distribution



Confidence intervals for regression coefficients

The confidence interval for the slope coefficients:

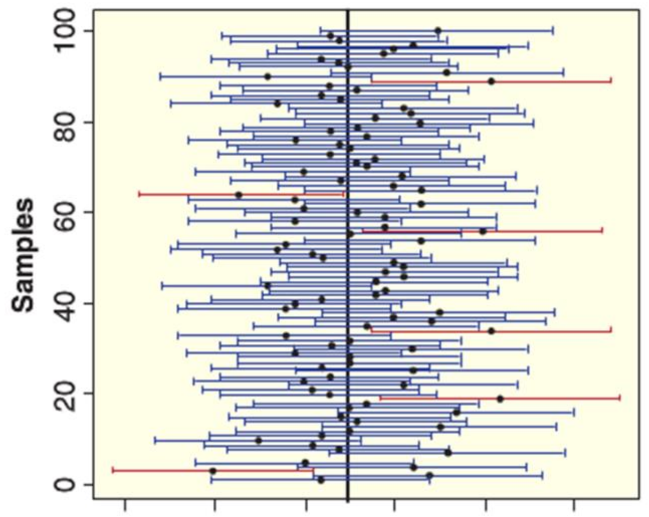
β_1

We can use the statsmodels package to run hypothesis tests and create confidence intervals using “parametric” methods based on t-distributions

```
import statsmodels.api as sm

sm_linear_model = sm.OLS(y_train,
                         X_train_with_constant).fit()

sm_linear_model.summary()
```



Let’s try this in Jupyter!

Unsupervised learning

Supervised learning and unsupervised learning

In **supervised learning** we have a set of features X, along with a label y

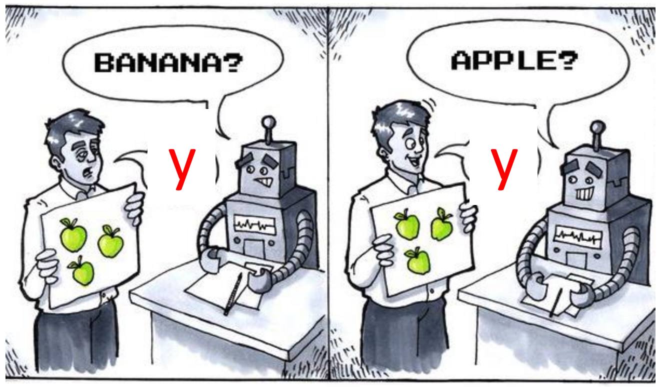
- We use the features X to predict y on new data

In **unsupervised learning**, we have features X, but **no** response variable y

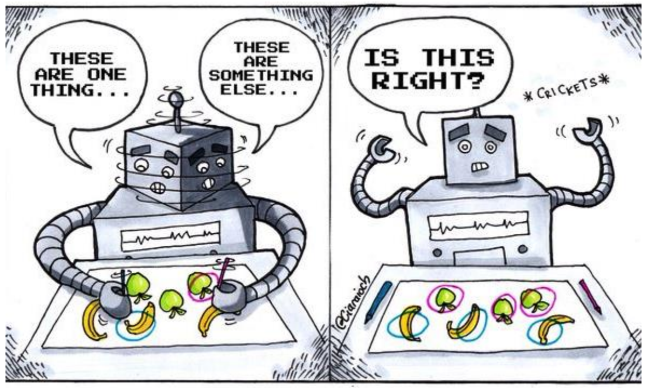
Unsupervised learning can be useful in order to find structure in the data and to visualize patterns

A key challenge in unsupervised learning is that there is no real ground truth response variable y

- So we don't have measures like the mean prediction accuracy



Supervised Learning



Unsupervised Learning

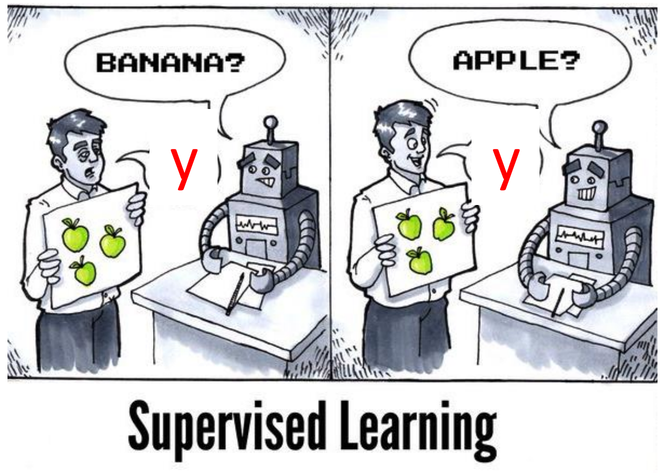
Unsupervised learning

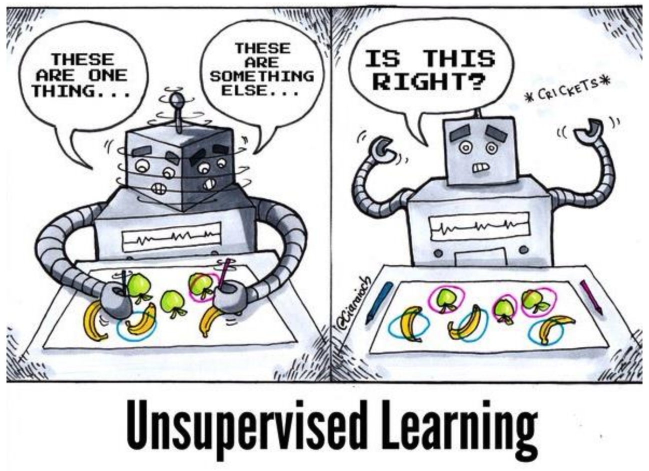
Given we are almost at the end of the semester, we will focus on clustering, which is one type of unsupervised learning

In **clustering** we try to group similar data points together

Another type of unsupervised learning:

- **Dimensionality reduction** where we try to find a smaller set of features that captures most of the variability original larger feature set
  - E.g., principal component analysis (PCA)





Clustering

Clustering

$$n\left\{\overbrace{\begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{np} \end{bmatrix}}^{p}\right.$$

Clustering divides n data points x_i's into subgroups

- Data points in the same group are similar/homogeneous
- Data points in different groups are different from each other

Examples:

- Examining gene expression levels to group cancer types together
- Examining consumer purchasing behavior to perform market segmentation

Clustering can be:

- **Flat**: no structure beyond dividing points into groups
- **Hierarchical**: Population is divided into smaller and smaller groups (tree like structure)

K-means clustering

K-means clustering partitions the data into ***K*** distinct, non-overlapping clusters

- i.e., each data point x_i belongs to exactly one cluster C_k

The number of clusters, ***K***, needs to be specified prior to running the algorithm

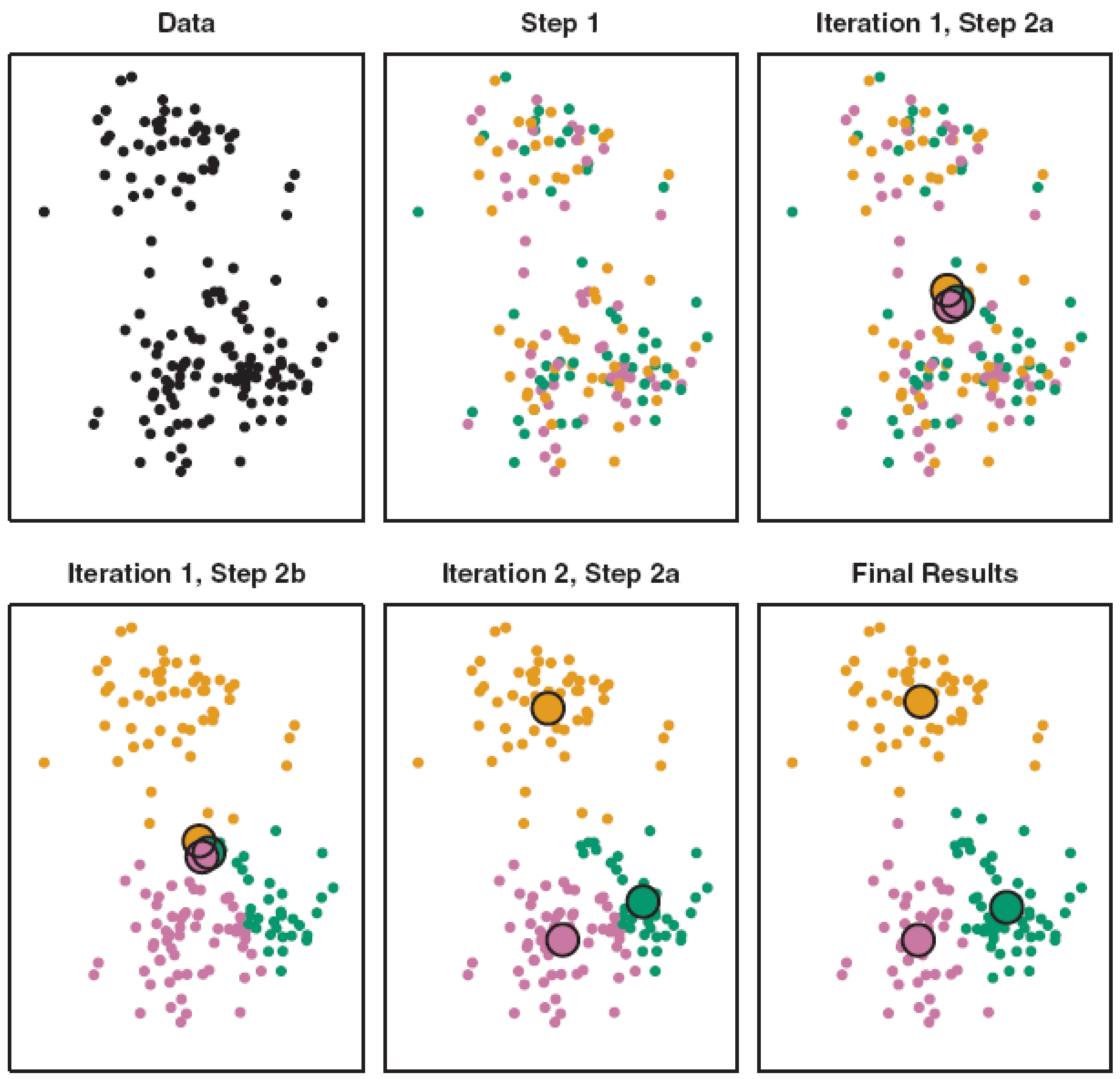
The goal is to minimize the within-cluster variation

- e.g., to make the Euclidean distance for all points within a cluster as small as possible

Finding the exact optimal solution is computationally intractable (there are k^n possible partitions), but a simple algorithm exists to find a local optimum which is often works well in practice

K-means clustering

1. Randomly assign points to clusters C_k
2. Calculate cluster centers as means of points in each cluster
3. Assign points to the closest cluster center
4. Recalculate cluster center as the mean of points in each cluster
5. Repeat steps 3 and 4 until convergence

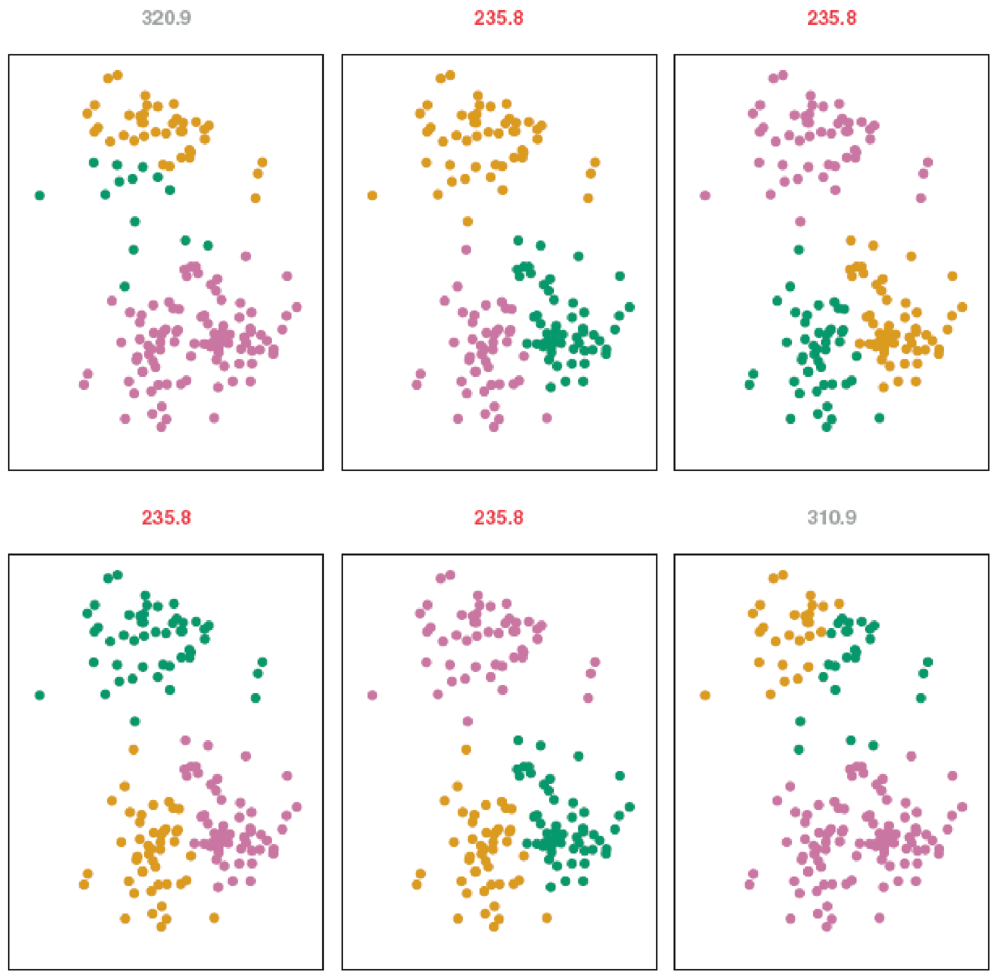


K-means clustering

Because only a local minimum is found, different random initializations will lead to different solutions

- One should run the algorithm multiple times to get better solutions

Let's explore this in Jupyter!



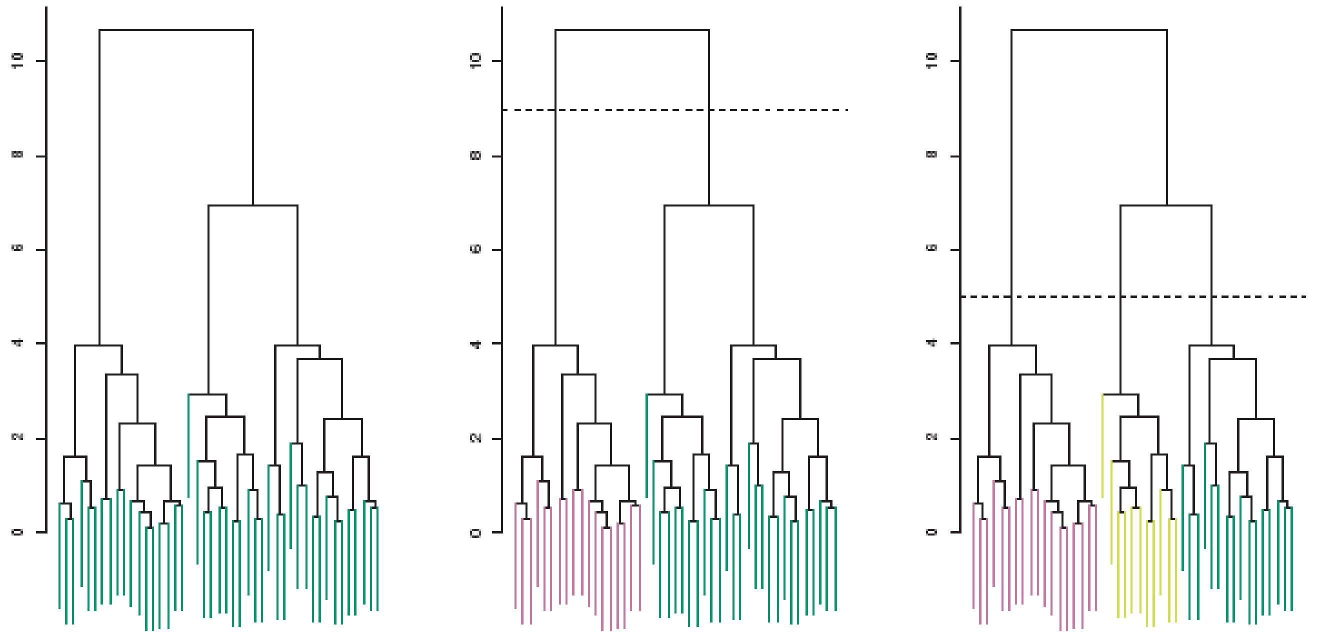
Hierarchical clustering

Hierarchical clustering

In **hierarchical clustering** we create a dendrogram which is a tree-based representation of successively larger clusters.

We can cut the dendrogram at any point to create as many clusters as desired

- i.e., don't need to specify the number of clusters, K, beforehand



Hierarchical clustering

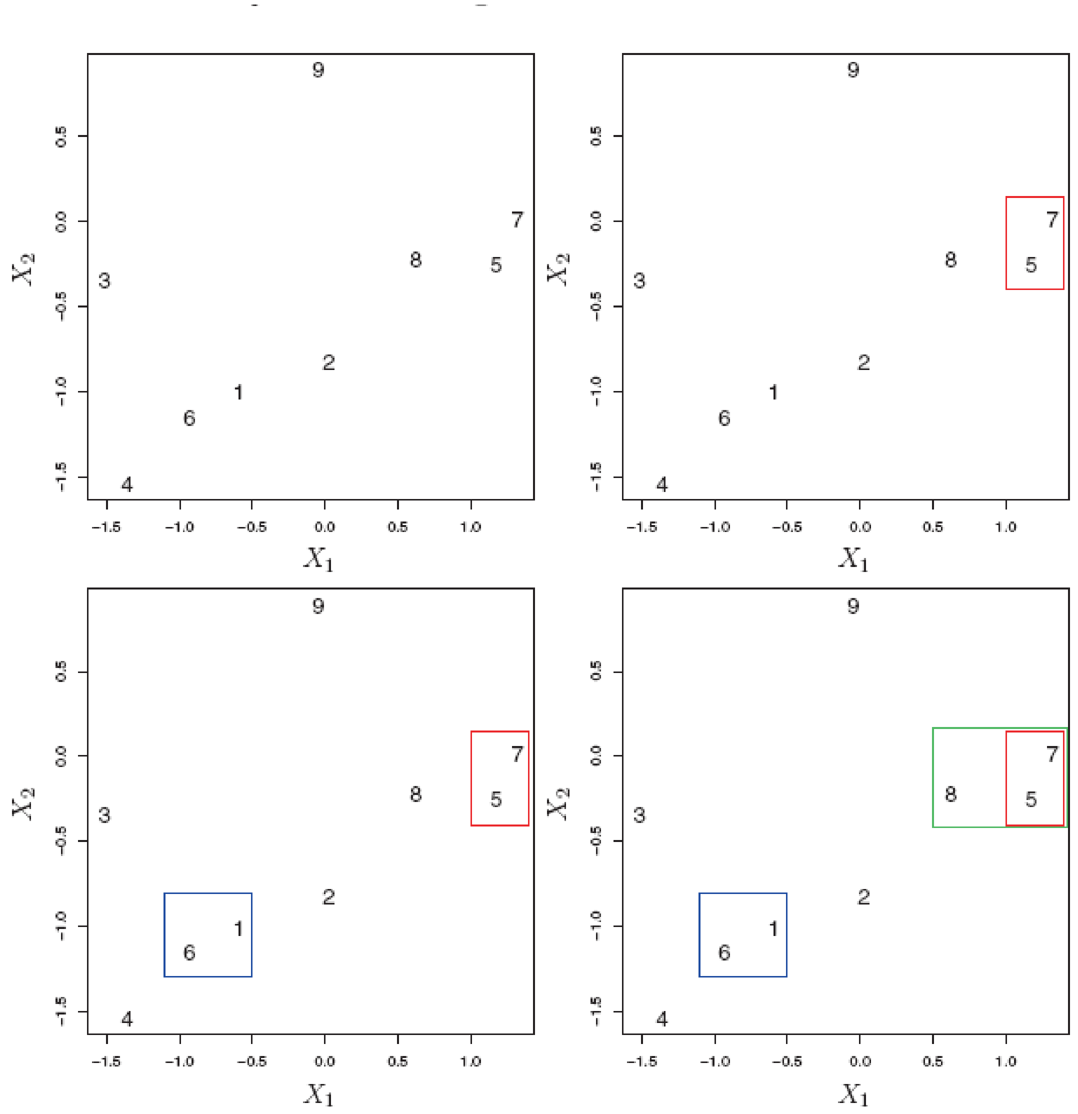
We can create a hierarchical clustering of the data using simple bottom-up agglomerative algorithm:

1. Choosing a (dis)similarity measure
   - E.g., The Euclidean distance

2. Initializing the clustering by treating each point as its own cluster

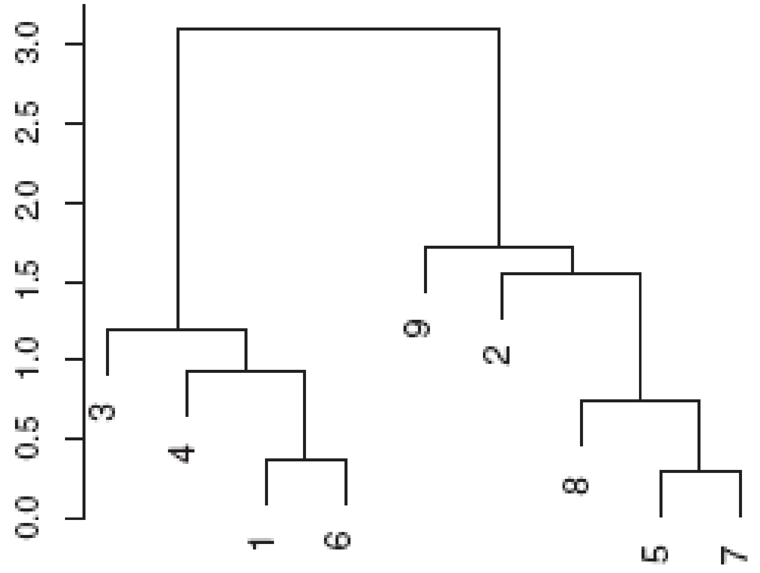
3. Successively merging the pair of clusters that are most similar
   - i.e., calculate the similarity between all pairs of clusters and merging the pair that is most similar

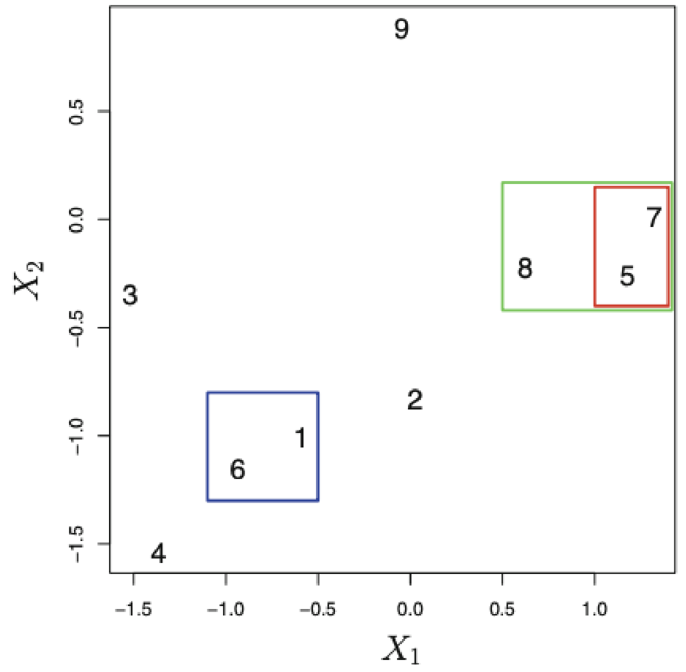
4. Stopping when all points have been merged into a single cluster



Hierarchical clustering

The vertical height that two clusters/points merge show how similar the two ***clusters*** are

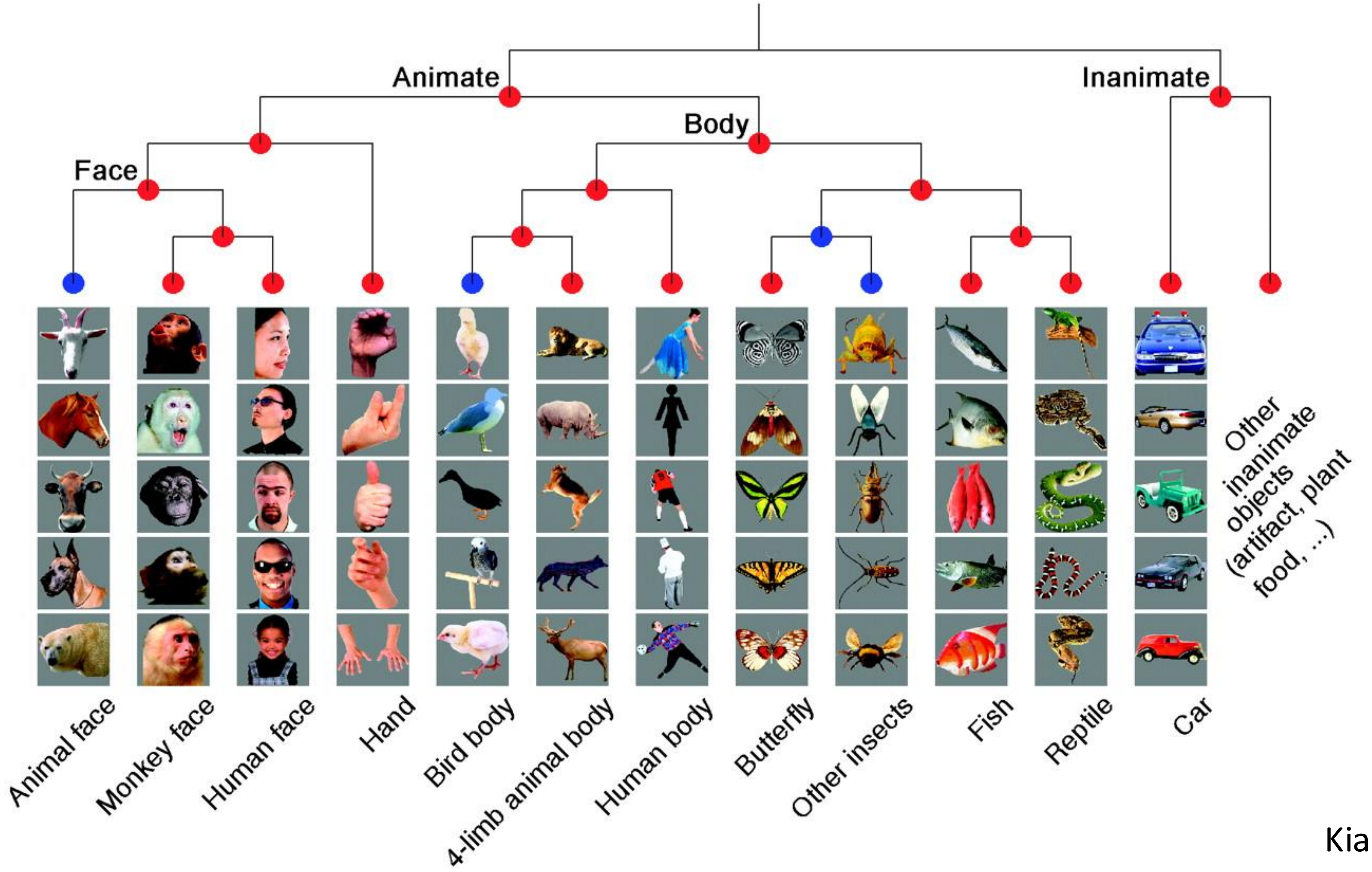




Note: horizontal distance between ***individual points*** is not important:

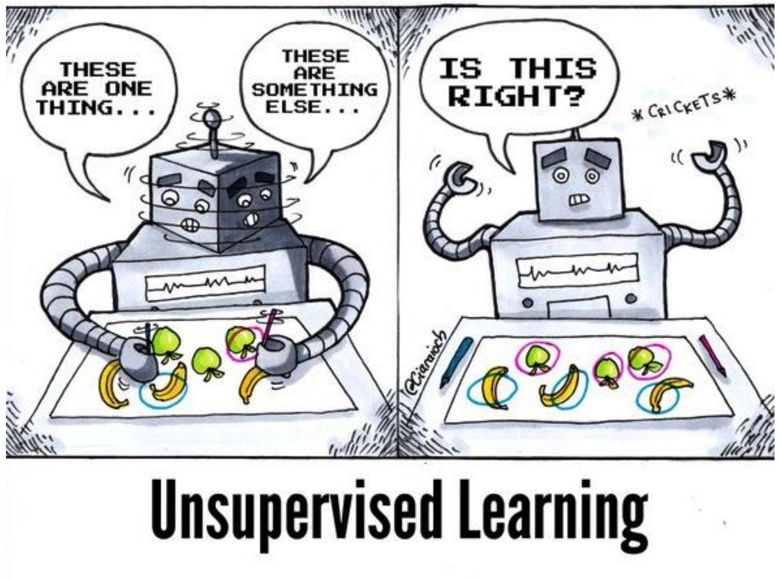
- point 9 is considered as similar to point 2 as it is to point 7

Hierarchical clustering example



Kiani et al, 2007

Issues with clustering



Choices made can effect the results:

- Feature normalization and/or dissimilarity measure
- K-means: choice of K
- For hierarchical cluster: linkage and cut height

Potential approaches to deal with these issues:

- Try a few methods and see if one gives interesting/useful results
- Validate that you get similar results on a second set of data

Let's explore this in Jupyter!

Brief discussion of Large Language Models

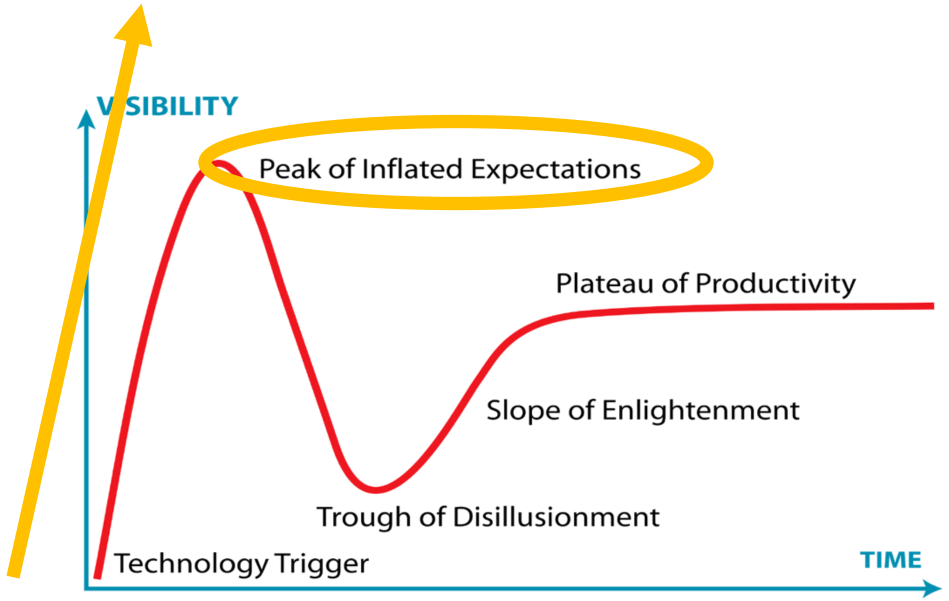
Brief discussion of Large Language Models

Large language models (LLMs) are taking over the world

ChatGPT

Gemini

Claude



Brief discussion of Large Language Models

LLMs can write code and analyze data

One can download free, open source, LLMs through the Hugging Face platform

Let’s very briefly look at running a LLM locally on our computers…