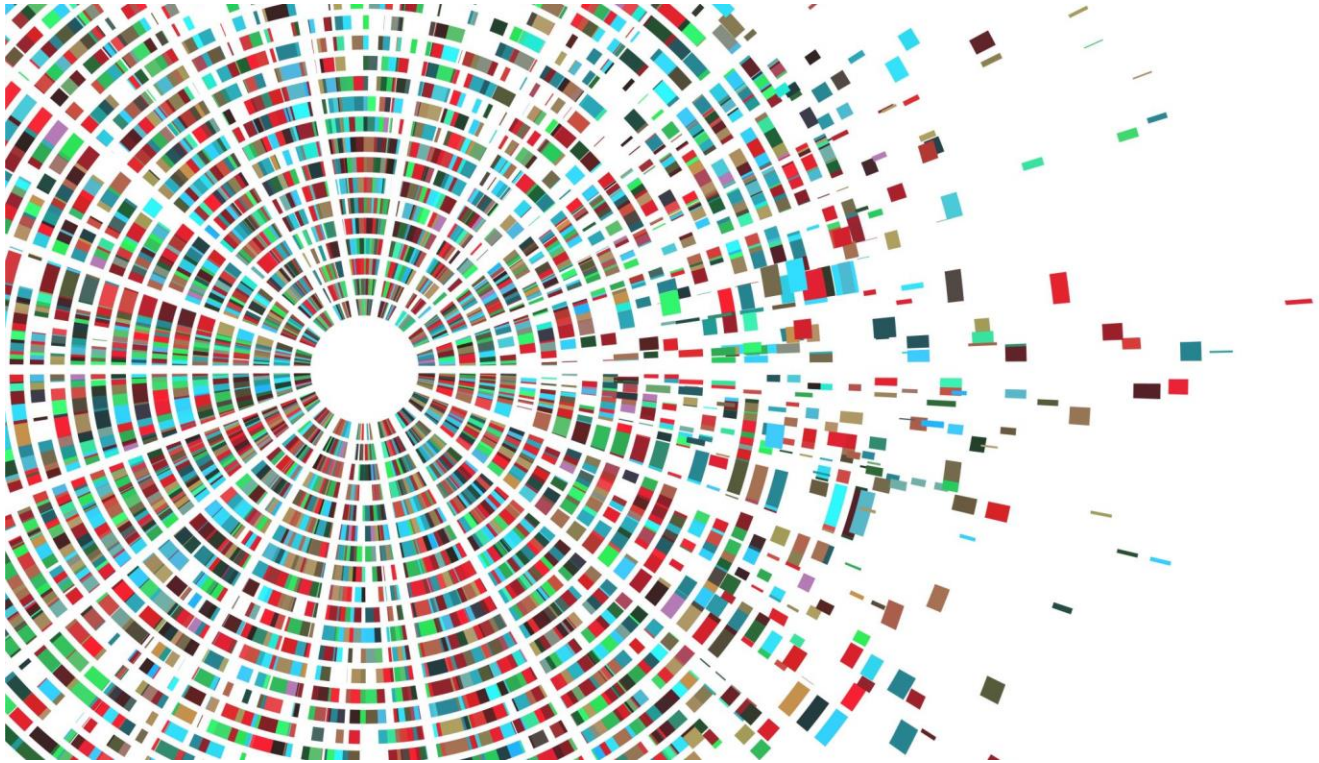
CHULA ENGINEERING
Foundation toward Innovation

Unsupervised learning

2110574: AI for Engineers



Duangdao Wichadakul, Ph.D.

Department of Computer Engineering, Faculty of Engineering, Chulalongkorn University

duangdao.w@chula.ac.th

Unsupervised learning

- Find patterns in data
- E.g., clustering customers by their purchases, clustering gene expression in biological science, cluster movements of people according to their mobility / customer segmentation, image segmentation, data augmentation, data embedding, PM2.5 density
- Reduce complexity of data dimensions

Source image.

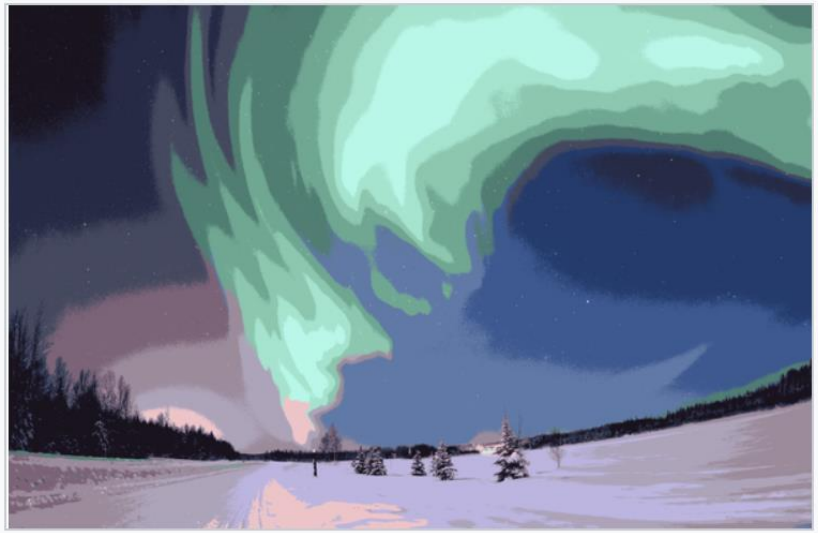
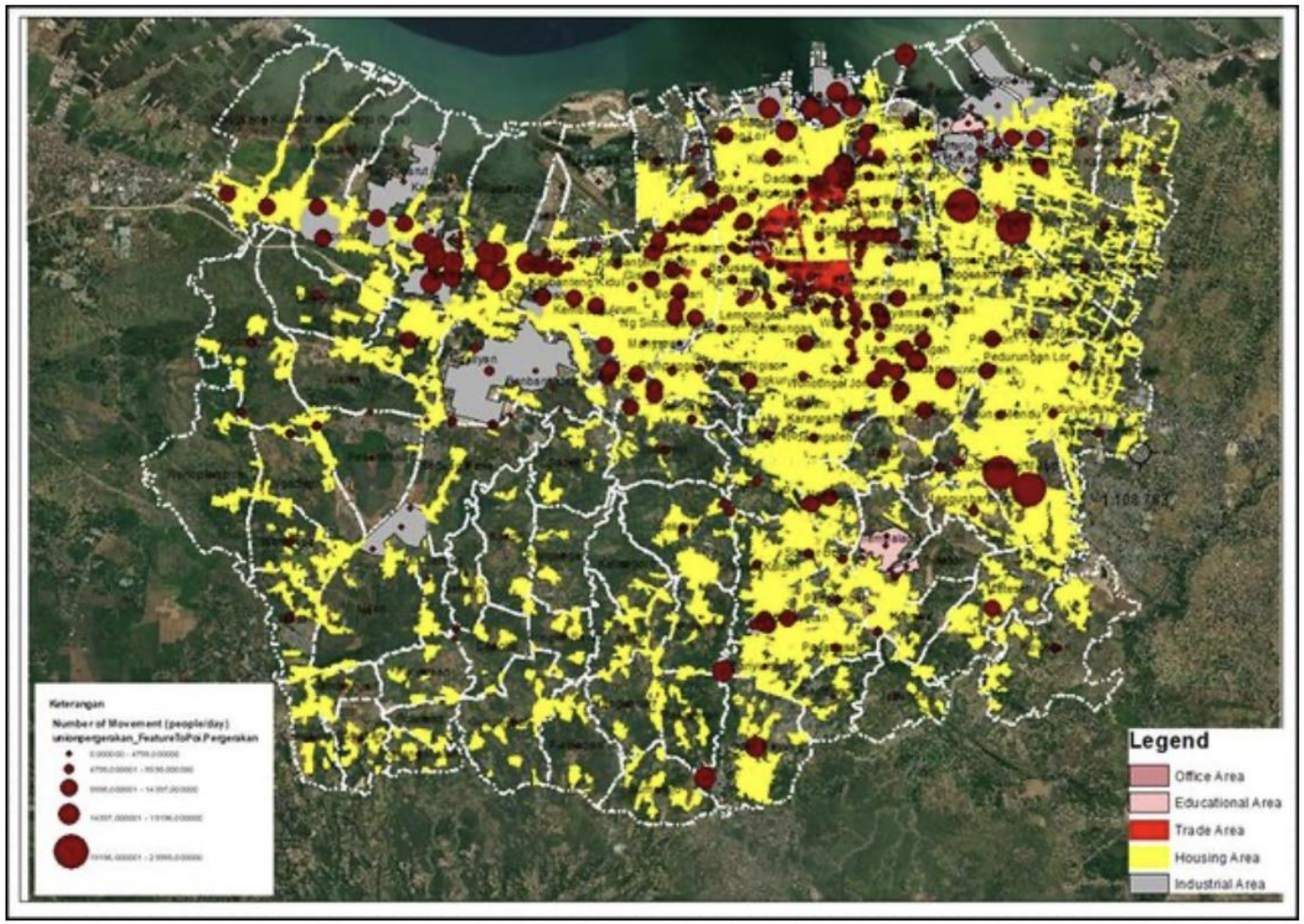


Image after running *k*-means with *k = 16*. Note that a common technique to improve performance for large images is to downsample the image, compute the clusters, and then reassign the values to the larger image if necessary.

Image segmentation with K-means clustering with k = 16

https://en.wikipedia.org/wiki/Image_segmentation



Figure

Caption

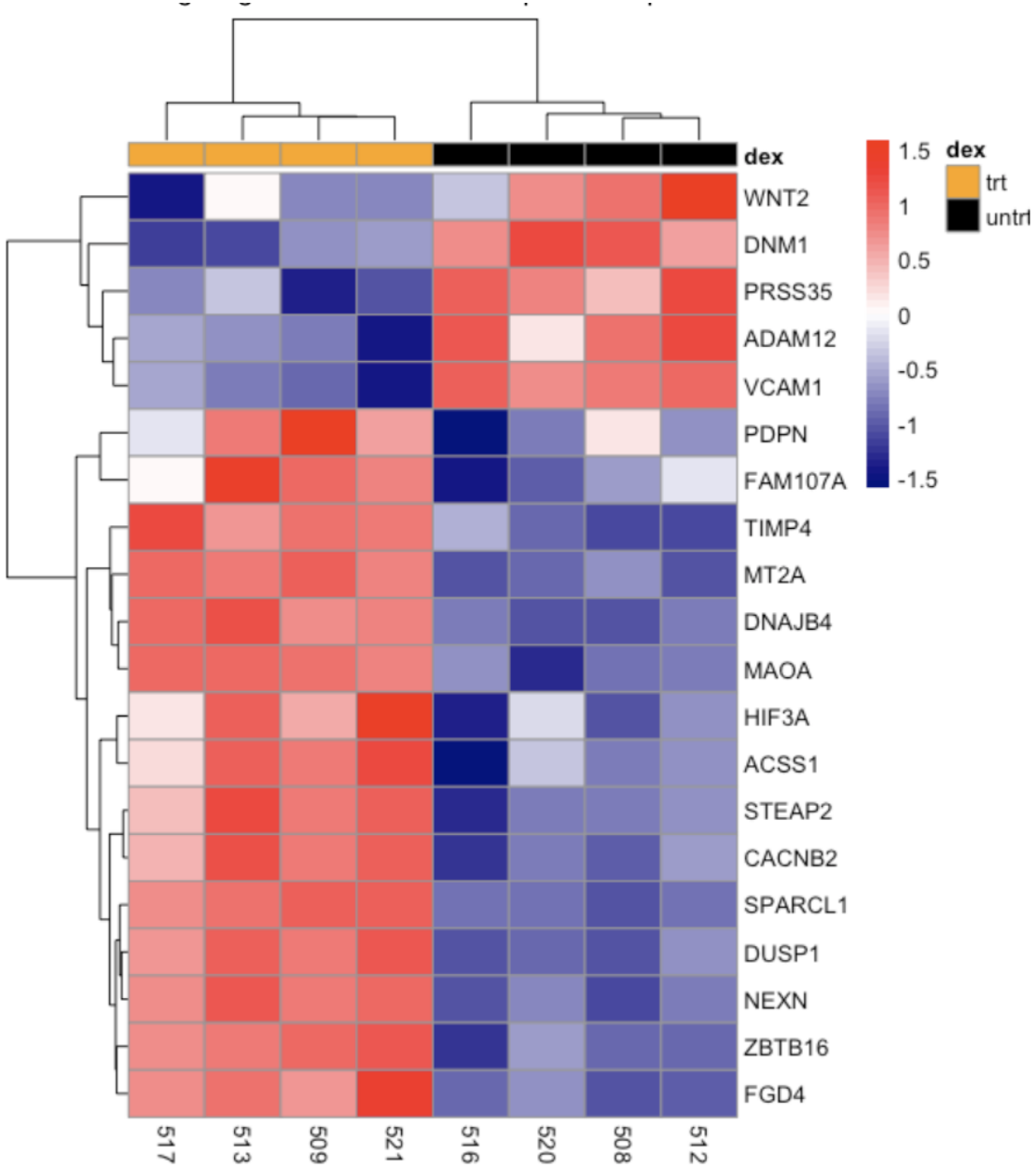
Figure 2. Mobility Pattern in Semarang City (Source: Analysis, 2020)

IOP Conference Series: Earth and Environmental Science

PAPER · OPEN ACCESS

Spatial Regression Modelling Impact of Population Movement Intensity and Land Use to Air Temperature in Semarang City, Indonesia

W. Widjonarko and M. Maryono 2021 *IOP Conf. Ser.: Earth Environ. Sci.* **887** 012003



Gene expression clustering

https://bioinformatics.ccr.cancer.gov/docs/data-visualization-with-r/Lesson5_intro_to_ggplot/

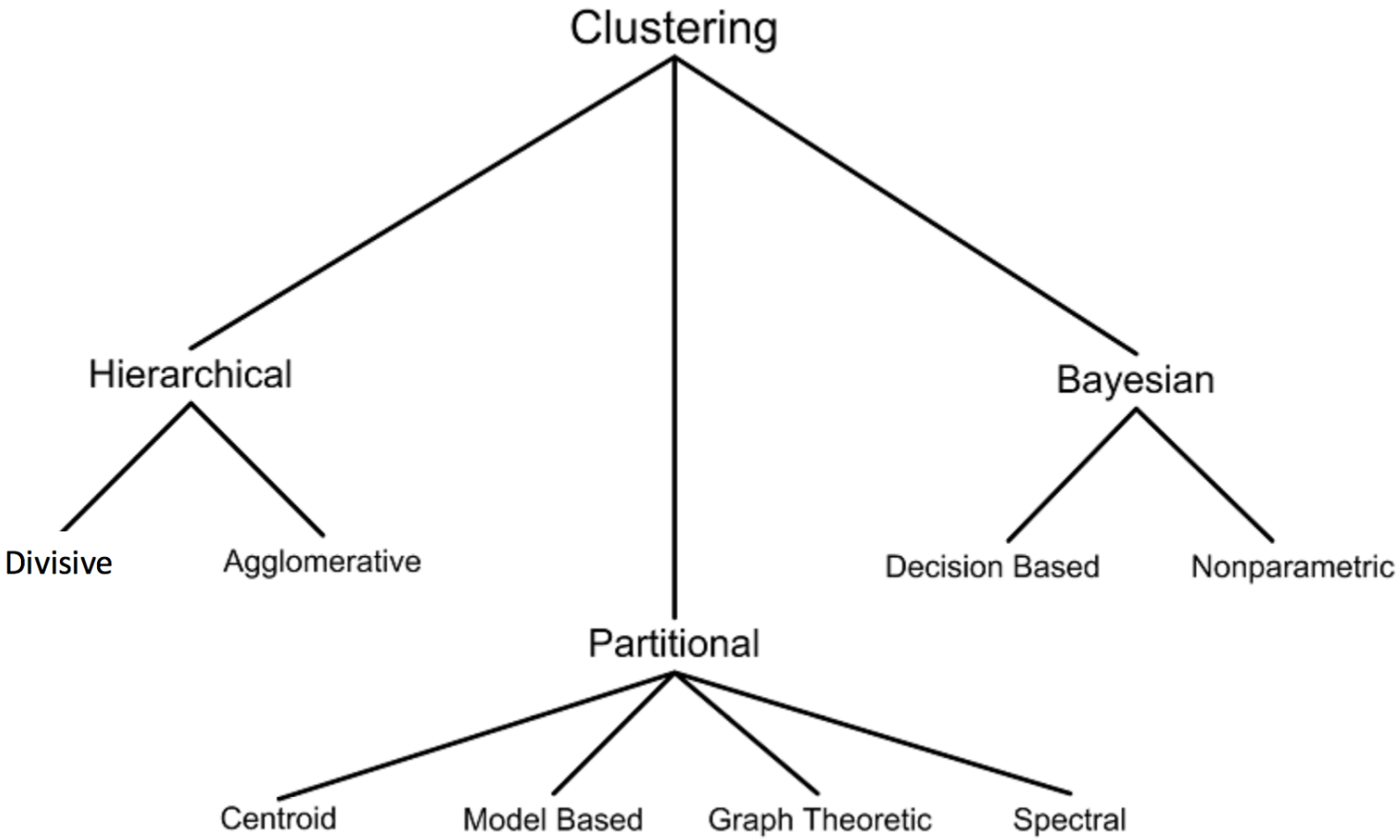
What is clustering?

- The assigning of unlabeled data into similarity groups called clusters
- A cluster contains data items which are similar between members and dissimilar from data items in other groups

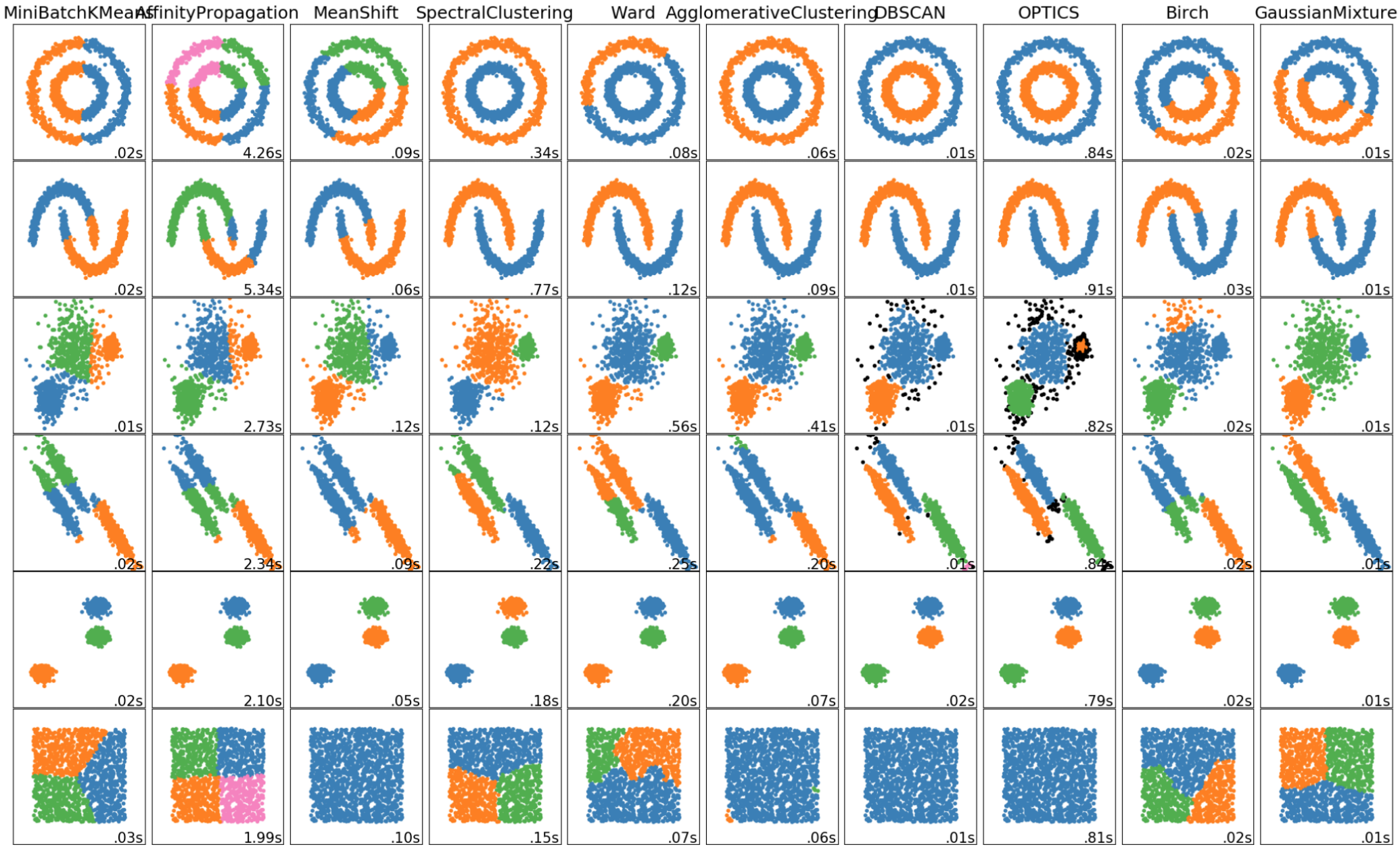
Supervised and unsupervised learning

- Supervised learning
  - Find patterns for **a *prediction* task**
  - E.g., classify if the tumor, forecast the PM2.5 value
- Unsupervised learning
  - Find patterns **w/o a *prediction task*** in mind
  - E.g. group genes according to their expression levels, group the locations according to their environmental conditions

Clustering algorithms in overall



http://www.mit.edu/~9.54/fall14/slides/Class13.pdf



A comparison of the clustering algorithms in scikit-learn

https://scikit-learn.org/stable/modules/clustering.html

Outlines

- K-means clustering
- Hierarchical clustering
- PCA
- t-SNE

K-means clustering

K-mean clustering

- Divide samples into clusters
- Need to specify k (number of clusters)

K-means algorithm

1. Initialize cluster for k centroids
2. Assign each data point to a cluster with the closest centroid **How to measure the closest?**
3. Update cluster centroids
4. Repeat steps 2-3 until the stopping condition is met.

What is the stopping condition?

How to measure the closest?

Euclidian distance or L2 distance

$$d(\mathbf{p}, \mathbf{q}) = d(\mathbf{q}, \mathbf{p}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2}$$

$$= \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}.$$

https://en.wikipedia.org/wiki/Euclidean_distance

How to measure the closest?

Euclidian distance or L2 distance

A : 7, 2, 0

c1 : 1, 5, 6

$$dist(c1,A) = (7-1)^2 + (2-5)^2 + (0-6)^2$$

$$dist(c1,A) = 36 + 9 + 36$$

c2 : 8, 1, 3

$$dist(c2,B) = (7-8)^2 + (2-1)^2 + (0-3)^2$$

$$dist(c1,A) = 1 + 1 + 9$$

c3 : 6, 7, 5

$$dist(c2,B) = (7-6)^2 + (2-7)^2 + (0-5)^2$$

$$dist(c1,A) = 1 + 25 + 25$$

How to update cluster centroids?

c2 : 8, 1, 3

Data points in c2

A : 7, 2, 0

B : 3, 3, 1

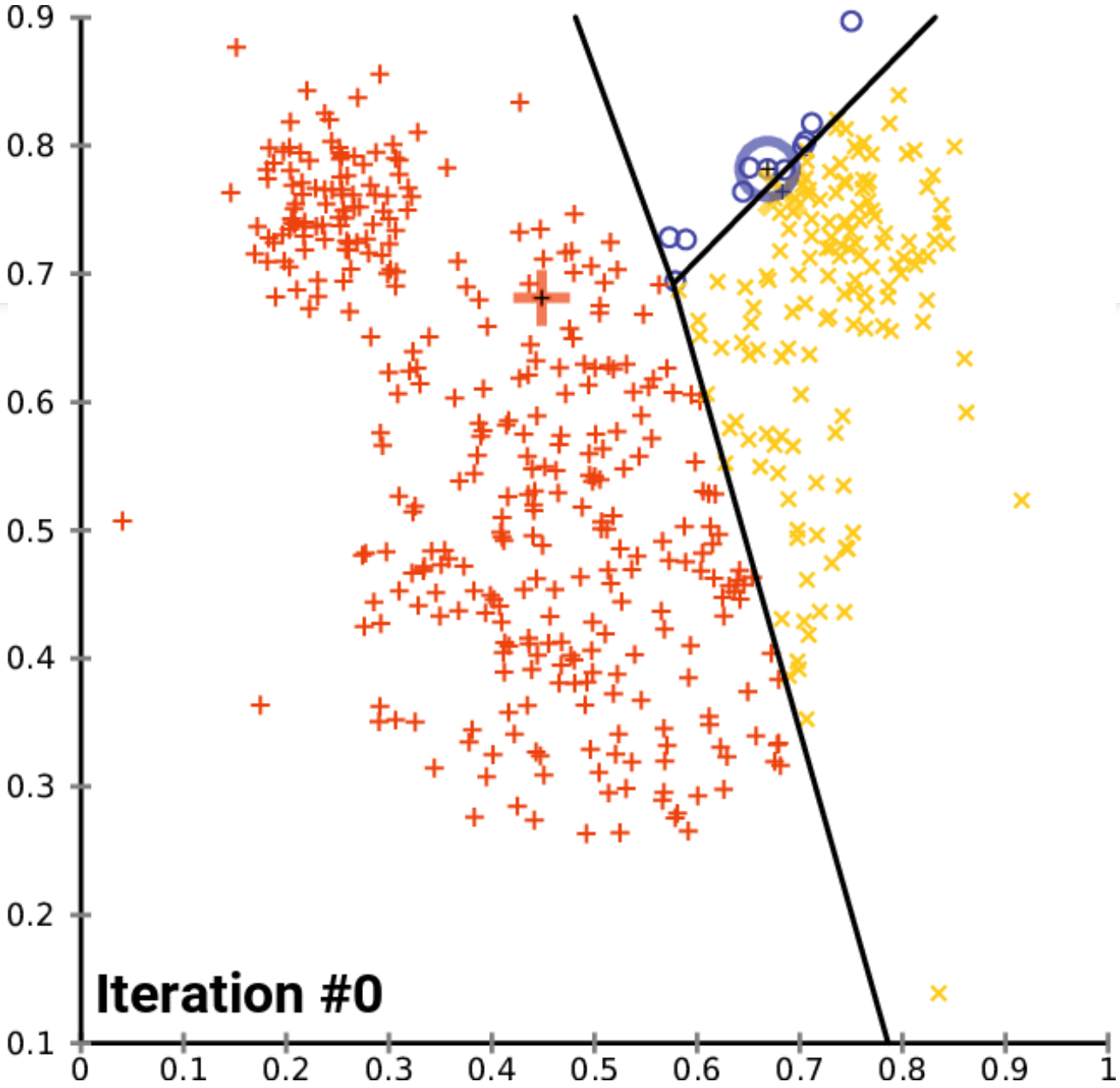
C : 4, 5, 6

D : 6, 1, 5

New centroid = Average of data points feature wise

$$\frac{(7+3+4+6)}{4} = 5 \qquad \frac{(2+3+5+1)}{4} = 2.75 \qquad \frac{(0+1+6+5)}{4} = 3$$

Convergence of K-means clustering



https://upload.wikimedia.org/wikipedia/commons/e/ea/K-means_convergence.gif

What is the stopping condition?

- The stopping criteria tells the algorithm to get out of the loop.
- The stopping criteria might not return the BEST results!!!
- Examples of stopping criteria:
  - The datapoints assigned to a specific cluster remain the same
  - Centroids remain the same
  - The distance of datapoints from their centroid in minimum (what we just set)
  - Fixed number of iterations have reached (insufficient iterations -> poor results)

Evaluation the cluster quality

Two methods to measure the cluster quality

1. Inertia: tells how far the datapoints within a cluster are. The lower the better (starting from 0 to grows up)
2. Silhouette score: tells how far away the datapoints in one cluster are from the datapoints in another cluster. The score is ranged from -1 to 1. Close to 1 is better.

K-Means clustering for Iris dataset

https://colab.research.google.com/drive/1sbCaL2kZbBxh5Tm93uxd9Lebl_ELLhzg

Iris setosa

Iris versicolor

Iris virginica

Iris dataset

- Three species
  - *Setosa*
  - *Versicolor*
  - *Virginica*
- Features of Petal length, petal width, sepal length, sepal width

Iris dataset

Features or dimensions

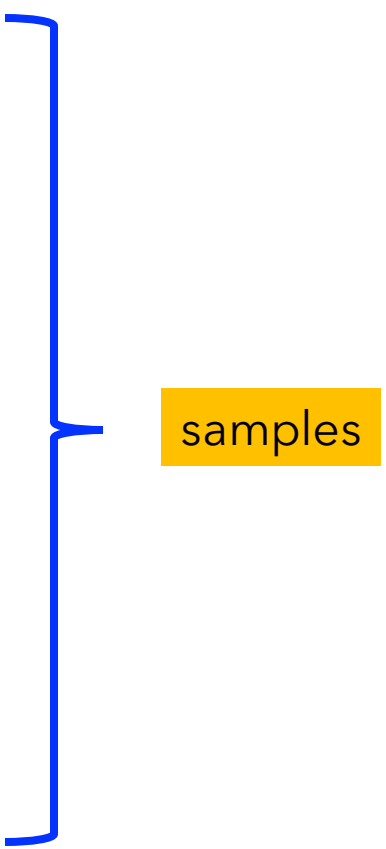
sepal_length	sepal_width	petal_length	petal_width	species
5.1	3.5	1.4	0.2	Iris-setosa
4.9	3	1.4	0.2	Iris-setosa
4.7	3.2	1.3	0.2	Iris-setosa
4.6	3.1	1.5	0.2	Iris-setosa
5	3.6	1.4	0.2	Iris-setosa
		...		
7	3.2	4.7	1.4	Iris-versicolor
6.4	3.2	4.5	1.5	Iris-versicolor
6.9	3.1	4.9	1.5	Iris-versicolor
5.5	2.3	4	1.3	Iris-versicolor
6.5	2.8	4.6	1.5	Iris-versicolor
		...		
6.3	3.3	6	2.5	Iris-virginica
5.8	2.7	5.1	1.9	Iris-virginica
7.1	3	5.9	2.1	Iris-virginica
6.3	2.9	5.6	1.8	Iris-virginica

samples

Iris dataset

Features or dimensions

sepal_length	sepal_width	petal_length	petal_width	species
5.1	3.5	1.4	0.2	
4.9	3	1.4	0.2	
4.7	3.2	1.3	0.2	
4.6	3.1	1.5	0.2	
5	3.6	1.4	0.2	
		…		
7	3.2	4.7	1.4	
6.4	3.2	4.5	1.5	
6.9	3.1	4.9	1.5	
5.5	2.3	4	1.3	
6.5	2.8	4.6	1.5	
		…		
6.3	3.3	6	2.5	
5.8	2.7	5.1	1.9	
7.1	3	5.9	2.1	
6.3	2.9	5.6	1.8	



samples

Data loading

```
# iris data
from sklearn.datasets import load_iris

# Load iris data into samples
samples = load_iris()
print(samples)
```

```
{'data': array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1],
       [5.4, 3.7, 1.5, 0.2],
```

Data exploration

```
X = samples.data
print('X shape:', X.shape)
y = samples.target
print('y shape:', y.shape)
```

```
X shape: (150, 4)
y shape: (150,)
```

```
print('x:', X[:3])
print('y:', y[:3])
```

```
x: [[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]]
y: [0 0 0]
```

Library import

```
from sklearn.cluster import KMeans
model = KMeans(n_clusters=3)
# we only put X (not y)
model.fit(X)
```

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=3, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=None, tol=0.0001, verbose=0)
```

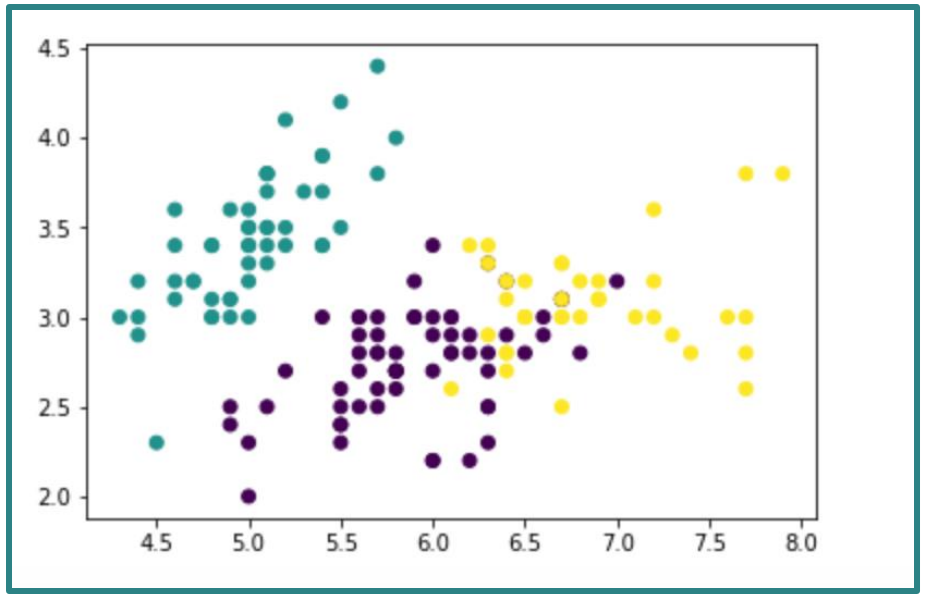
Try prediction

```
# try predict the cluster of X
y_pred = model.predict(X)
print('predict:', y_pred.shape)
print(y_pred)
```

```
predict: (150,)
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 2 2 2 2 0 2 2 2 2
 2 2 0 0 2 2 2 2 0 2 0 2 0 2 2 0 0 2 2 2 2 2 0 2 2 2 2 0 2 2 2 0 2 2 2 0 2
 2 0]
```

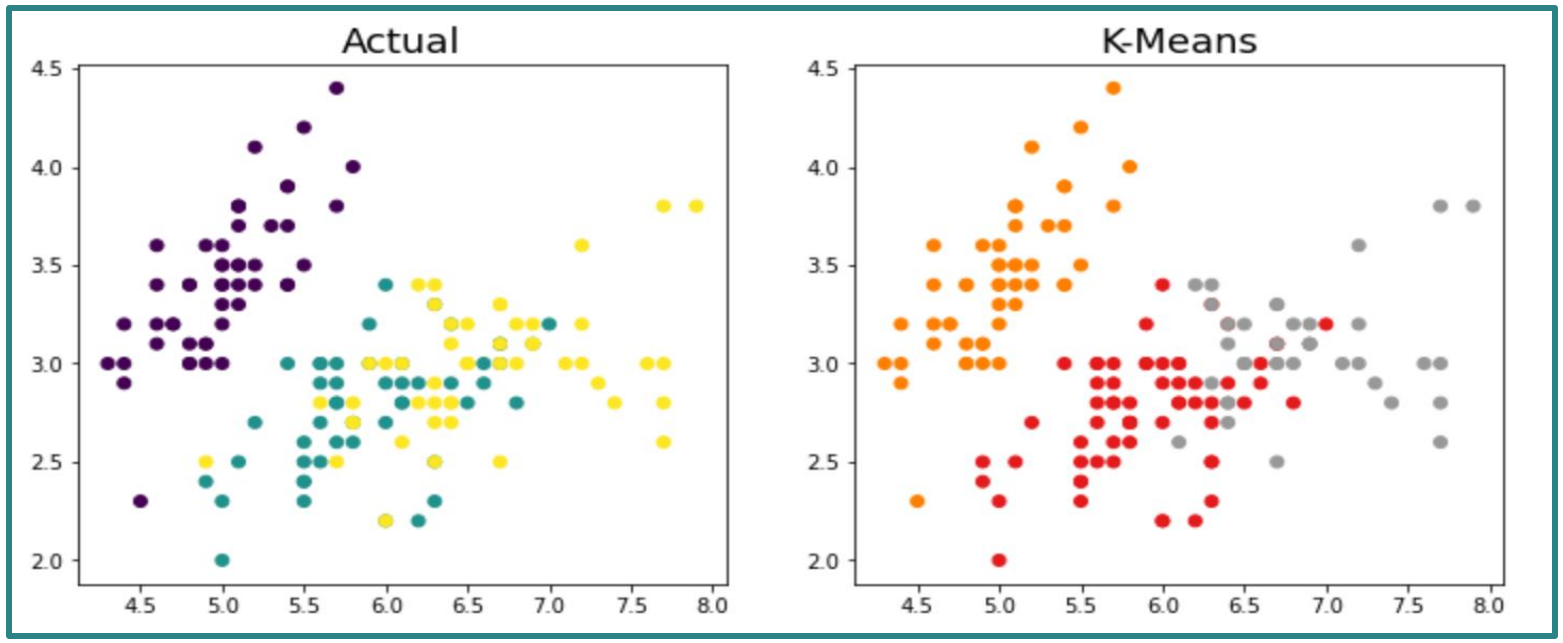
Visualize the clusters

```
import matplotlib.pyplot as plt
# we use scatter plot
# c=y; we color the dots by clusters
# however, we can only plot 2 dimensions (out of 4)
plt.scatter(X[:,0], X[:,1], c=y_pred)
plt.show()
```



Compare with the ground truth labels

```
fig, ax = plt.subplots(1, 2, figsize=(12,5))
ax[0].scatter(X[:,0], X[:,1], c=y)
ax[1].scatter(X[:,0], X[:,1], c=y_pred, cmap=plt.cm.Set1)
ax[0].set_title('Actual', fontsize=18)
ax[1].set_title('K-Means', fontsize=18)
```



Measure the fitness

```
# this is the "sum of squared" distance for all points to their centers
# better fitted model should have a lower inertia
print('inertia:', model.inertia_)
```

```
inertia: 78.85144142614601
```

Find the right 'K' using Elbow method

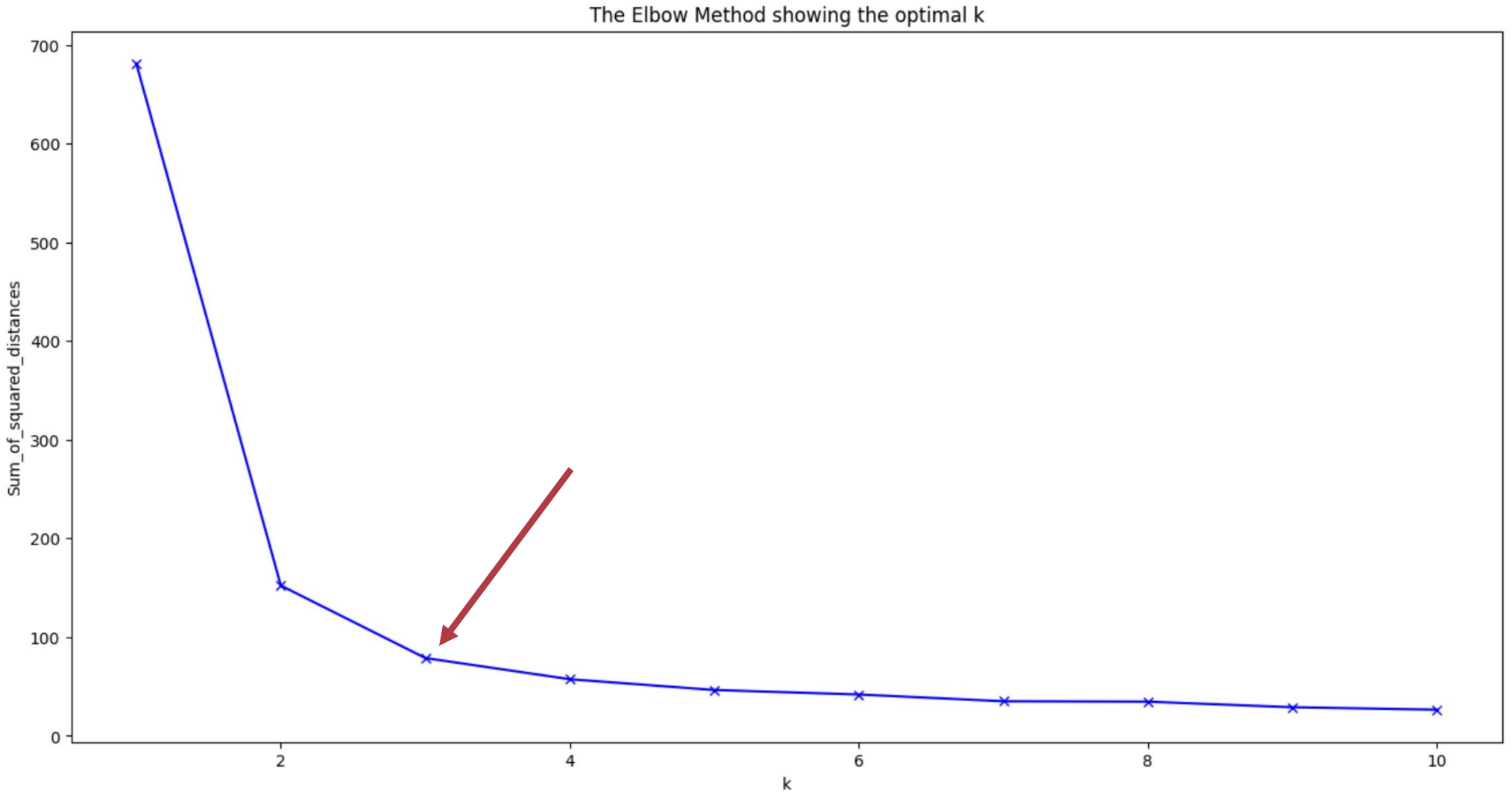
Progress bar

```
# Run K-means for a set of k
from tqdm import tqdm
inertias = []
models = []
K = list(range(1, 10+1))
X = samples.data
for k in tqdm(K):
    model = KMeans(n_clusters=k)
    model.fit(X)
    models.append(model)
    inertias.append(model.inertia_)
```

Plot the distortion of K-means

```
# Plotting the distortions of K-Means
plt.figure(figsize=(16,8))
plt.plot(K, inertias, 'bx-')
plt.xlabel('k')
plt.ylabel('Sum_of_squared_distances')
plt.title('The Elbow Method showing the optimal k')
plt.show()
```

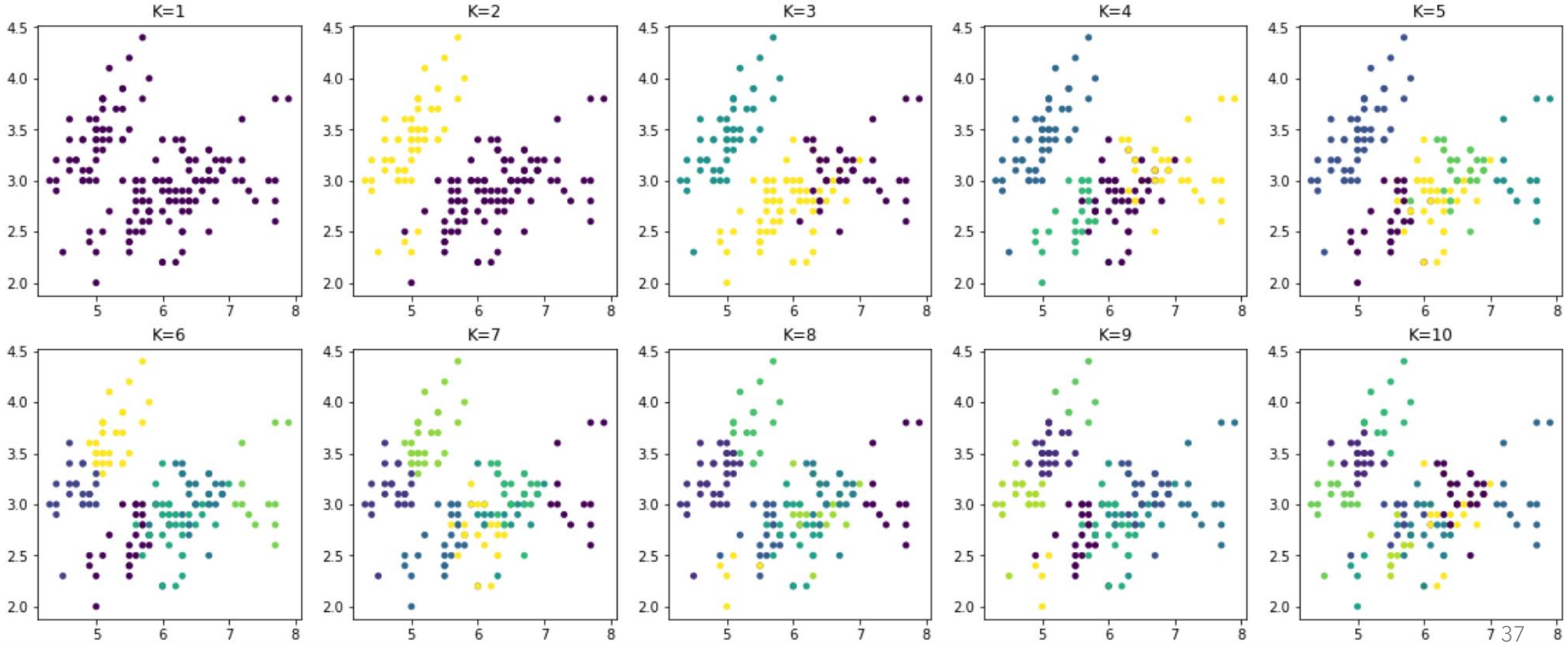
Plot the inertias of K-means



Clustering with different K

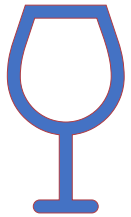
```
fig, ax = plt.subplots(2, 5, figsize=(20, 8))
ax = [*ax[0], *ax[1]]
for k, model, ax in zip(K, models, ax):
  y_pred = model.predict(X)
  ax.set_title(f'K={k}')
  ax.scatter(X[:,0], X[:,1], c=y_pred, s=16)
```

Clustering with different K



K-mean clustering (Another example)

(K-Means Piedmont Wine dataset)
https://colab.research.google.com/drive/18ZJpaNipWbSfjVAUfFqM0xJmFpEcGfhZ



class 1 59
class 2 71
class 3 48

Wine dataset

The attributes are (dontated by Riccardo Leardi, riclea@anchem.unige.it)

1) Alcohol
2) Malic acid
3) Ash
4) Alcalinity of ash
5) Magnesium
6) Total phenols
7) Flavanoids
8) Nonflavanoid phenols
9) Proanthocyanins
10)Color intensity
11)Hue
12)OD280/OD315 of diluted wines
13)Proline

Data loading

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.cluster import KMeans

# Wine data
# 178 samples with 3 distinct varieties of red wine: Barolo, Grignolino and Barbera
# Features measure chemical composition e.g. alcohol content
# https://archive.ics.uci.edu/ml/datasets/Wine
url = 'https://raw.githubusercontent.com/wichadak/wine/master/wine.data'
df = pd.read_csv(url, header=None)
df.sample(frac=1).head()
```

Wine data

Features or dimensions

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
85	2	12.67	0.98	2.24	18.0	99	2.20	1.94	0.30	1.46	2.62	1.23	3.16	450
66	2	13.11	1.01	1.70	15.0	78	2.98	3.18	0.26	2.28	5.30	1.12	3.18	502
21	1	12.93	3.80	2.65	18.6	102	2.41	2.41	0.25	1.98	4.50	1.03	3.52	770
83	2	13.05	3.86	2.32	22.5	85	1.65	1.59	0.61	1.62	4.80	0.84	2.01	515
97	2	12.29	1.41	1.98	16.0	85	2.55	2.50	0.29	1.77	2.90	1.23	2.74	428

samples

Wine type

Data preparation

```
X = df[df.columns[1:]].to_numpy(dtype=np.float32)
y = df[df.columns[0]].to_numpy()
print('X:', X.shape, X.dtype)
print('y:', y.shape, y.dtype)
```

```
X: (178, 13) float32
y: (178,) int64
```

Cluster it!

```
from sklearn.cluster import KMeans
# again we know that it has 3 classes
model = KMeans(n_clusters=3)
model.fit(X)
y_pred = model.predict(X)
print(y_pred.shape)
```

```
(178,)
```

Assess the clustering quality

```
from sklearn import metrics
print('ARI:', metrics.adjusted_rand_score(y, y_pred))
```

```
ARI: 0.37111371823084754
```

If we have ground truth labels, we can use a method called "Adjusted Rand Index" (best = 1, worst ~ 0 or -1).

Normalize the input

We should normalize the input before clustering

```
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler(with_mean=True, with_std=True)
_X = scaler.fit_transform(X)
print('before mean and sd:', X[:,0].mean(), X[:,0].std())
print('after mean and sd:', _X[:,0].mean(), _X[:,0].std())

model = KMeans(n_clusters=3)
# using the normalized X
model.fit(_X)
y_pred = model.predict(_X)
print('ARI after normalized:', metrics.adjusted_rand_score(y, y_pred))
```

```
before mean and sd: 13.000619 0.8095429
after mean and sd: -2.1430884e-08 1.0
ARI after normalized: 0.897494981509320 7
```

Don’t even need to zero the mean

```
# don't even need to zero the mean!
scaler = StandardScaler(with_mean=False, with_std=True)
_X = scaler.fit_transform(X)
print('before mean and sd:', X[:,0].mean(), X[:,0].std())
print('after mean and sd:', _X[:,0].mean(), _X[:,0].std())

model = KMeans(n_clusters=3)
# using the normalized X
model.fit(_X)
y_pred = model.predict(_X)
print('ARI after normalized:', metrics.adjusted_rand_score(y, y_pred))
```

```
before mean and sd: 13.000619 0.8095429
after mean and sd: 16.059208 0.99999994
ARI after normalized: 0.8974949815093207
```

Wrap as a pipeline

```
# Pipelines combine multiple steps ** Wrap up the above as a pipeline
from sklearn.preprocessing import StandardScaler # MaxAbsScaler and Normalizer are other examples
from sklearn.cluster import KMeans
from sklearn.pipeline import make_pipeline

scaler = StandardScaler()
kmeans = KMeans(n_clusters=3)
pipeline = make_pipeline(scaler, kmeans)
pipeline.fit(X)
y_pred = pipeline.predict(X)
print('ARI:', metrics.adjusted_rand_score(y, y_pred))
```

```
ARI: 0.8974949815093207
```

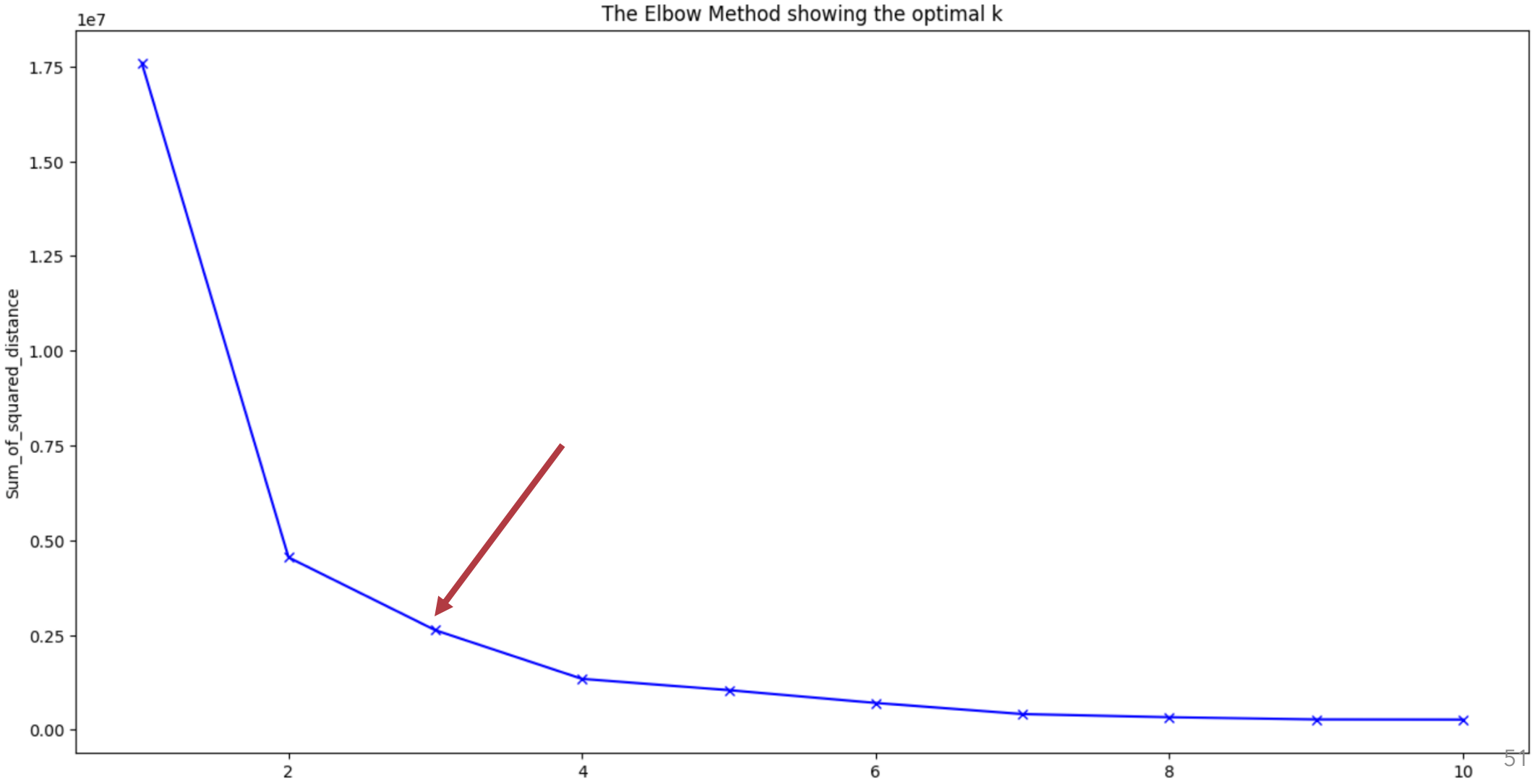
Select the best “K” with Elbow method

```
from tqdm import tqdm
# Run K-means for a set of k
inertias = []
models = []
K = range(1,10+1)
for k in tqdm(K):
    model = KMeans(n_clusters=k)
    model.fit(X)
    models.append(model)
    inertias.append(model.inertia_)
```

Plot the distortion of K-means

```
#Plotting the distortions of K-Means
plt.figure(figsize=(16,8))
plt.plot(K, inertias, 'bx-')
plt.xlabel('k')
plt.ylabel('Sum_of_squared_distance')
plt.title('The Elbow Method showing the optimal k')
plt.show()
```

Plot the inertias of K-means



Weakness of K-Means:

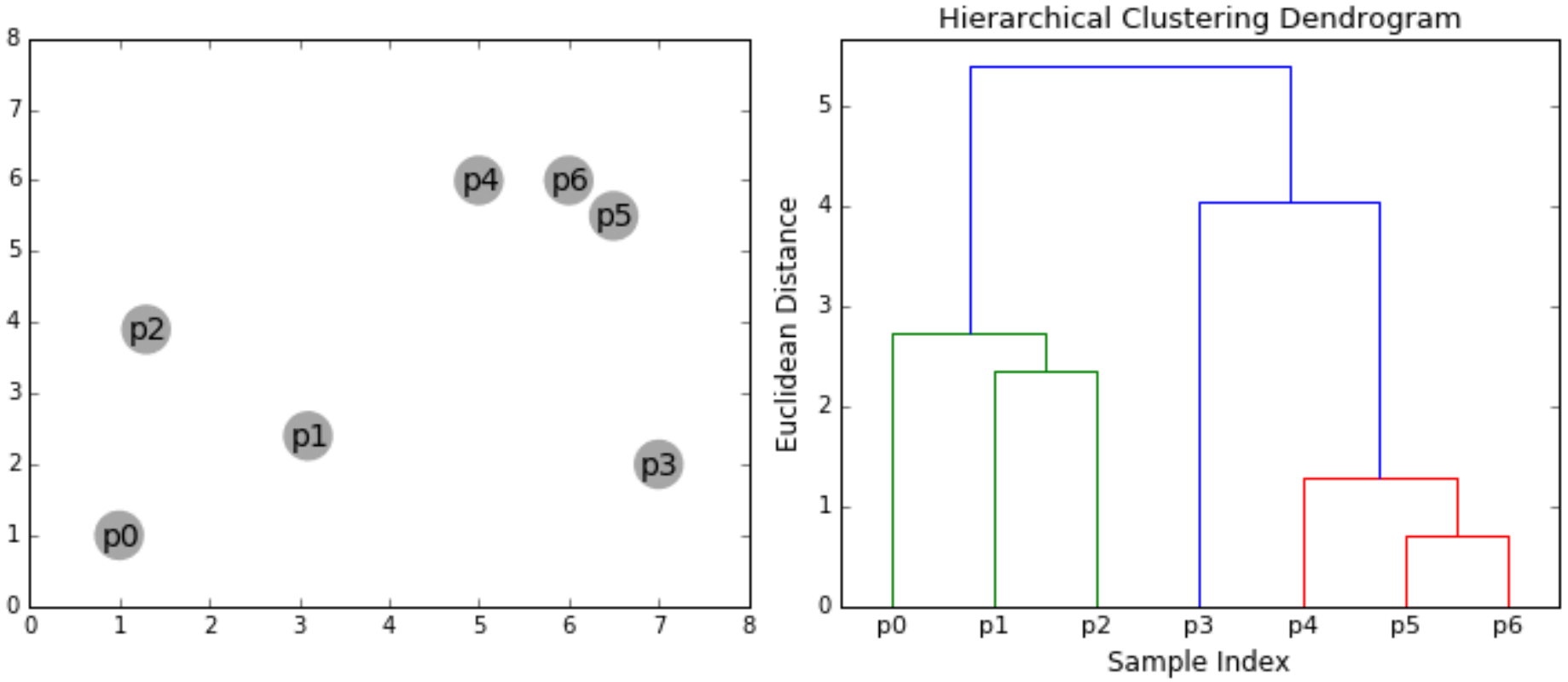
- We need k
- The algorithm is sensitive to outliers

Agglomerative hierarchical clustering algorithm

Agglomerative hierarchical clustering algorithm

- Initialize every datapoint as its own cluster.
- Find the two closest points and combine them as a cluster.
- Find the next closet points and combine them as a cluster.
- Repeat until all datapoints form a single big cluster.

How the Agglomerative hierarchical clustering works



https://www.digitalvidya.com/blog/the-top-5-clustering-algorithms-data-scientists-should-know/

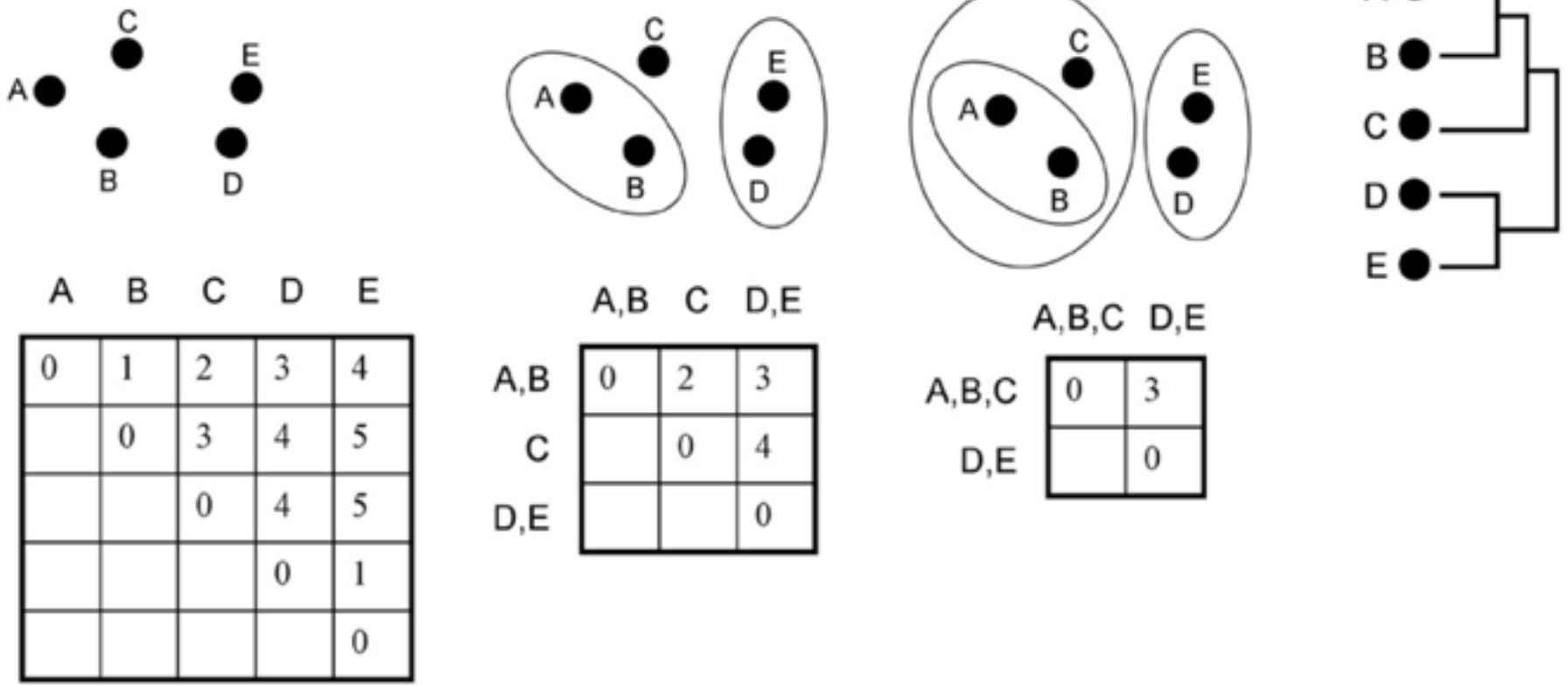
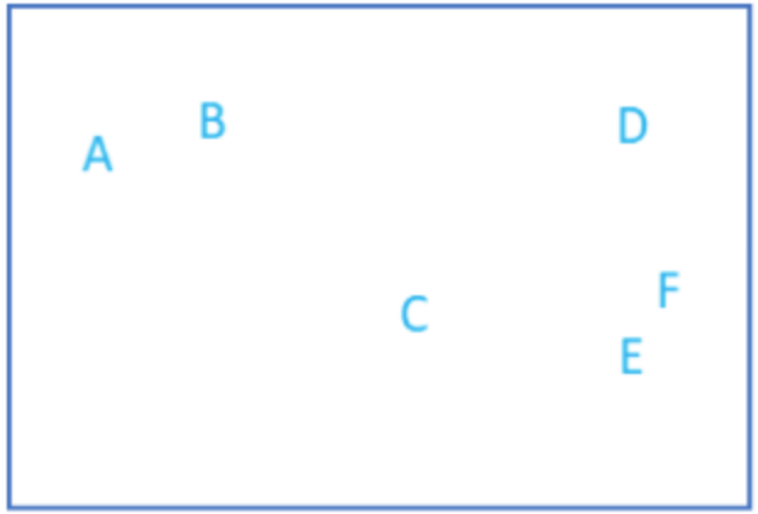


Figure 8. An example of a hierarchical clustering using single linkage algorithm. Consider five genes and the distances between them as shown in the table. In the first step, genes that are close to each other are grouped together and the distances are re-calculated using the single linkage algorithm. This procedure is repeated until all genes are grouped into one cluster. This information can be represented as a tree (shown to the right), where the distance from the branch point reflects the distance between genes or clusters. This image was adapted from Causton *et al*. (2003).

Figure 8: http://www.mrc-lmb.cam.ac.uk/genomes/madanm/microarray/chapter-final.pdf

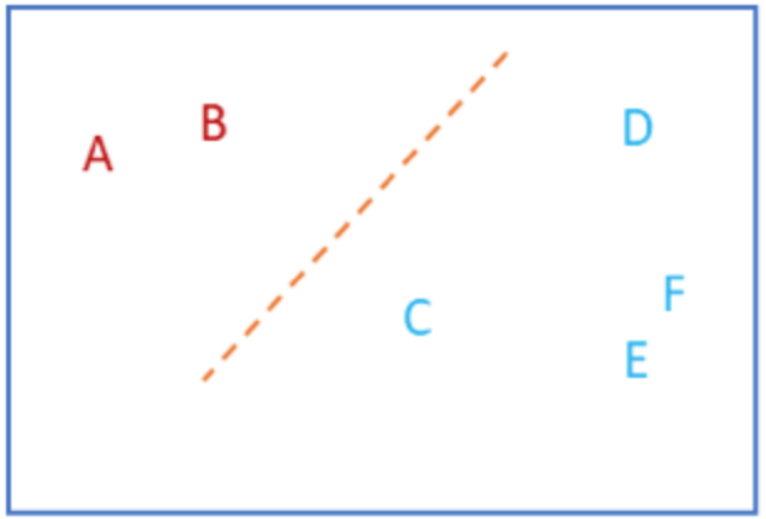
Dendrogram

- It is a summary of distance matrix.
- As occurs with most summaries, information is lost.
- B is not that far from C in the real data but not in the dendrogram.
- So, it is most accurate at the leaf nodes.

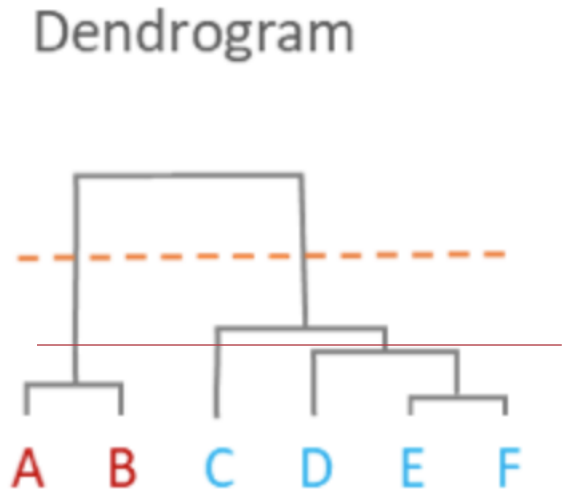


https://www.displayr.com/what-is-dendrogram/

Dendrogram (more)



- Observations are allocated to clusters by drawing a horizontal line through the dendrogram.
- Dendrogram cannot tell how many clusters we should have.



How to find the closest cluster?

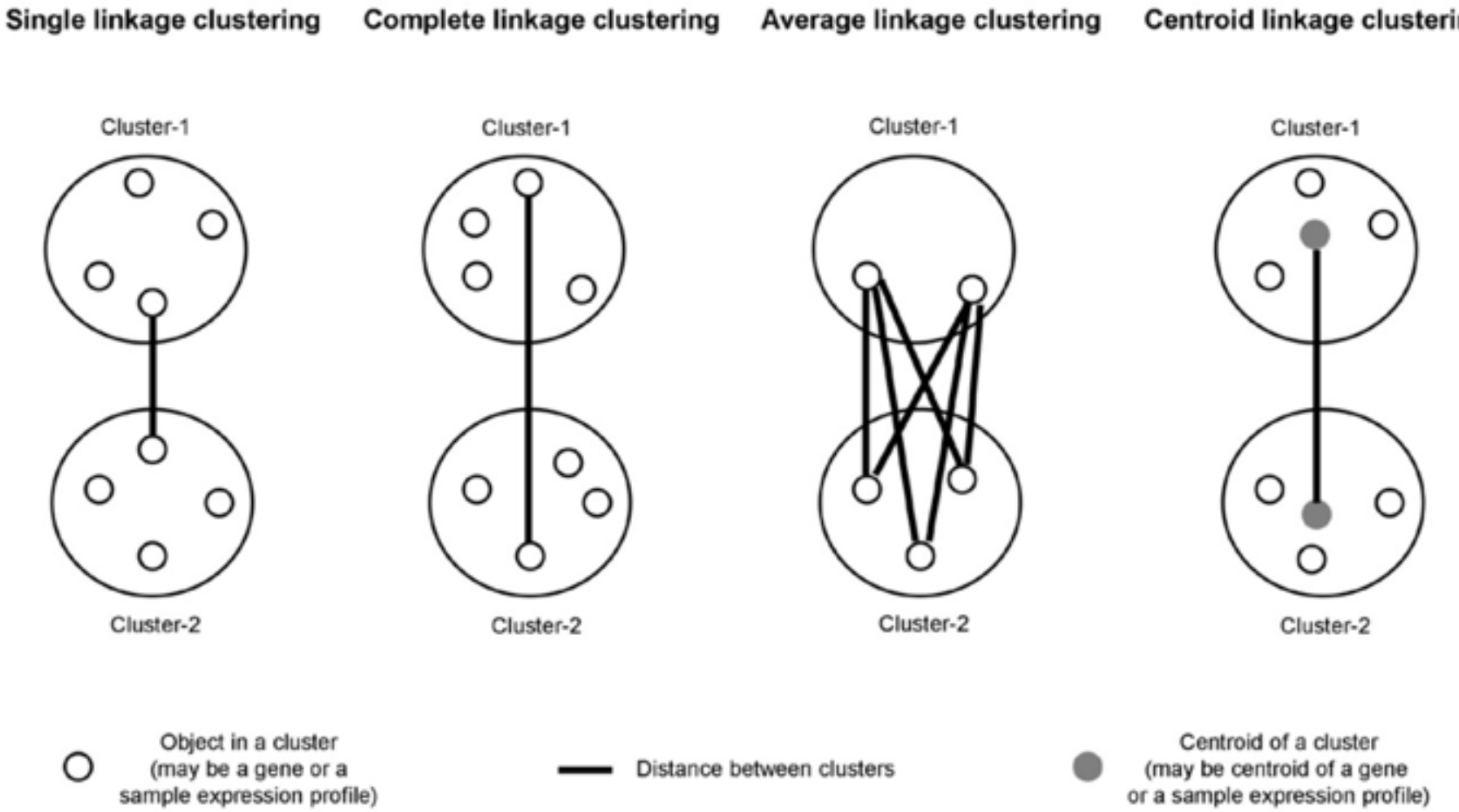


Figure 7. Different algorithms to find distance between two clusters.

Figure 7: http://www.mrc-lmb.cam.ac.uk/genomes/madanm/microarray/chapter-final.pdf

Agglomerative hierarchical clustering

Company stock moving dataset

https://colab.research.google.com/drive/1OpuXCX2Xf9tzc2NGGtd1rtMfwX2oFfAF

Prepare the dataset

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

url = 'https://raw.githubusercontent.com/wichadak/wine/master/company-stock-movements-2010-2015-incl.csv'
df = pd.read_csv(url)

# columns = stock movement by day
# rows = companies
# we can group similar companies by how similar are their stock movements
# i.e. companies in the same industries tend to move in the same direction
df.head()
```

Prepare the dataset

Features or dimensions

	companies	2010-01-04	2010-01-05	2010-01-06	2010-01-07	2010-01-08	2010-01-11	2010-01-12	2010-01-13	2010-01-14
0	Apple	0.580000	-0.220005	-3.409998	-1.170000	1.680011	-2.689994	-1.469994	2.779997	-0.680003
1	AIG	-0.640002	-0.650000	-0.210001	-0.420000	0.710001	-0.200001	-1.130001	0.069999	-0.119999
2	Amazon	-2.350006	1.260009	-2.350006	-2.009995	2.960006	-2.309997	-1.640007	1.209999	-1.790001
3	American express	0.109997	0.000000	0.260002	0.720002	0.190003	-0.270001	0.750000	0.300004	0.639999
4	Boeing	0.459999	1.770000	1.549999	2.690003	0.059997	-1.080002	0.360000	0.549999	0.530002

5 rows × 964 columns

samples

Prepare the dataset

```
X = df[df.columns[1:]].to_numpy(dtype=np.float32)
print('X:', X.shape, X.dtype)
```

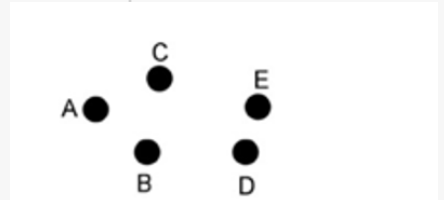
```
X: (60, 963) float32
```

```
# make labels int
i_to_name = list(sorted(df['companies'].unique()))
name_to_i = {name: i for i, name in enumerate(i_to_name)}

y = np.array([name_to_i[_y] for _y in df['companies']])
print('y:', y.shape, y.dtype)
print(y[:10])
```

```
y: (60,) int64
[4, 1, 2, 3, 6, 5, 7, 8, 9, 13]
```

Define function for getting the linkage

```
from scipy.cluster.hierarchy import dendrogram

def get_linkage(model):
    # needed for visualization
    # from: https://scikit-learn.org/stable/auto_examples/cluster/plot_agglomerative_dendrogram.
    counts = np.zeros(model.children_.shape[0])
    n_samples = len(model.labels_)
    for i, merge in enumerate(model.children_):
        current_count = 0
        for child_idx in merge:
            if child_idx < n_samples:
                current_count += 1  # leaf node
            else:
                current_count += counts[child_idx - n_samples]
        counts[i] = current_count

    linkage_matrix = np.column_stack([model.children_, model.distances_,
                                      counts]).astype(float)
    return linkage_matrix
```



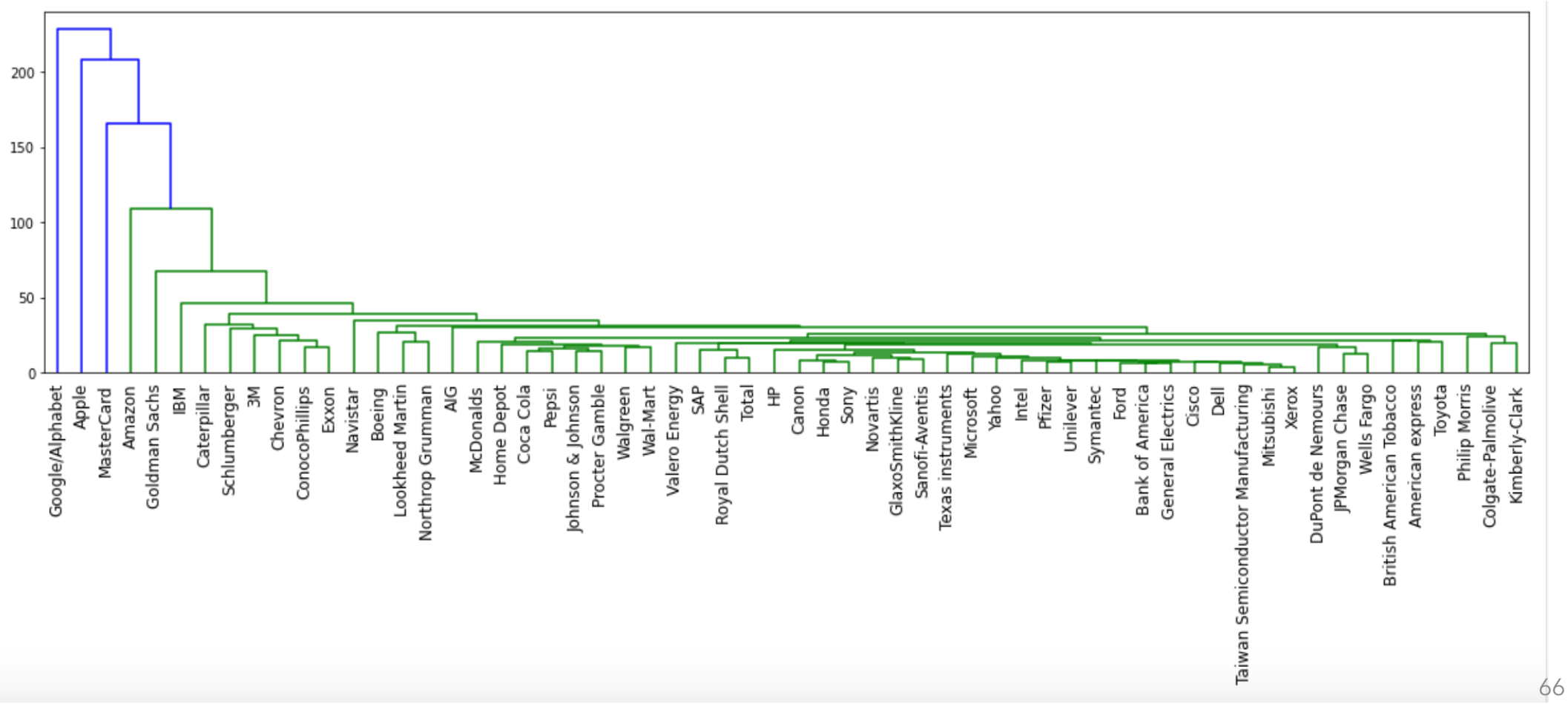
	A	B	C	D	E
A	0	1	2	3	4
B		0	3	4	5
C			0	4	5
D				0	1
E					0

Perform agglomerative hierarchical clustering

```
from sklearn.cluster import AgglomerativeClustering
from sklearn.preprocessing import normalize

model = AgglomerativeClustering(linkage='complete',
                                distance_threshold=0, # to calculate the full tree
                                n_clusters=None)
model = model.fit(X)

# get linkages
links = get_linkage(model)
# plot the dendrogram
fig, ax = plt.subplots(figsize=(20, 5))
dendrogram(links,
           labels=list(df['companies']),
           leaf_rotation=90,
           leaf_font_size=12,
           ax=ax);
```

https://scikit-learn.org/stable/modules/generated/sklearn.cluster.AgglomerativeClustering.html

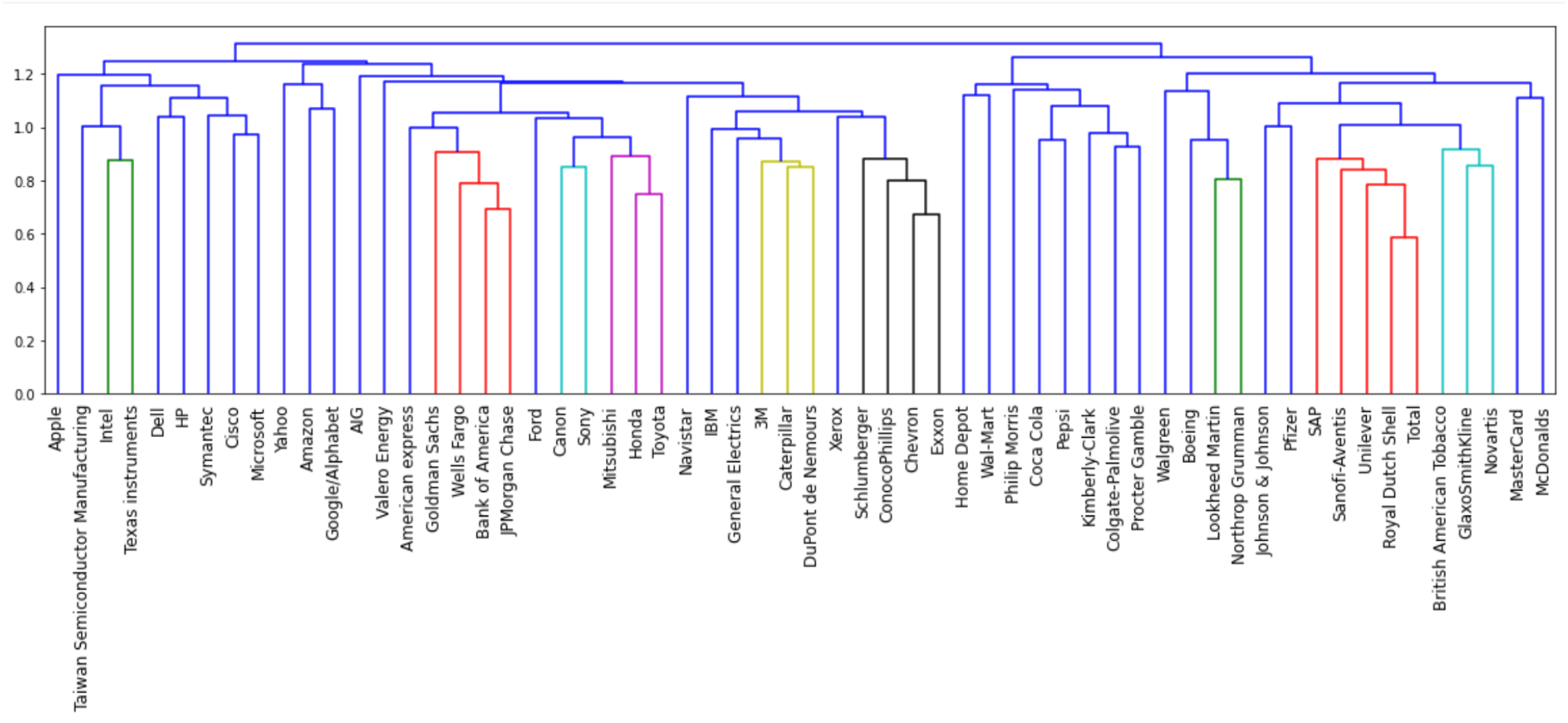
Clustering result



Normalize the movement

```
from sklearn.cluster import AgglomerativeClustering
from sklearn.preprocessing import normalize

# normalize each row to be a unit vector
X_norm = normalize(X)

model = AgglomerativeClustering(linkage='complete',
                                distance_threshold=0,
                                n_clusters=None)
model = model.fit(X_norm)

links = get_linkage(model)

# Plot the dendrogram
fig, ax = plt.subplots(figsize=(20, 5))
dendrogram(links,
           labels=list(df['companies']),
           leaf_rotation=90,
           leaf_font_size=12,
           ax=ax);
```

Clustering result w/ normalized data



Perform agglomerative hierarchical clustering w/ different linkage

Distance metrics

Linkage

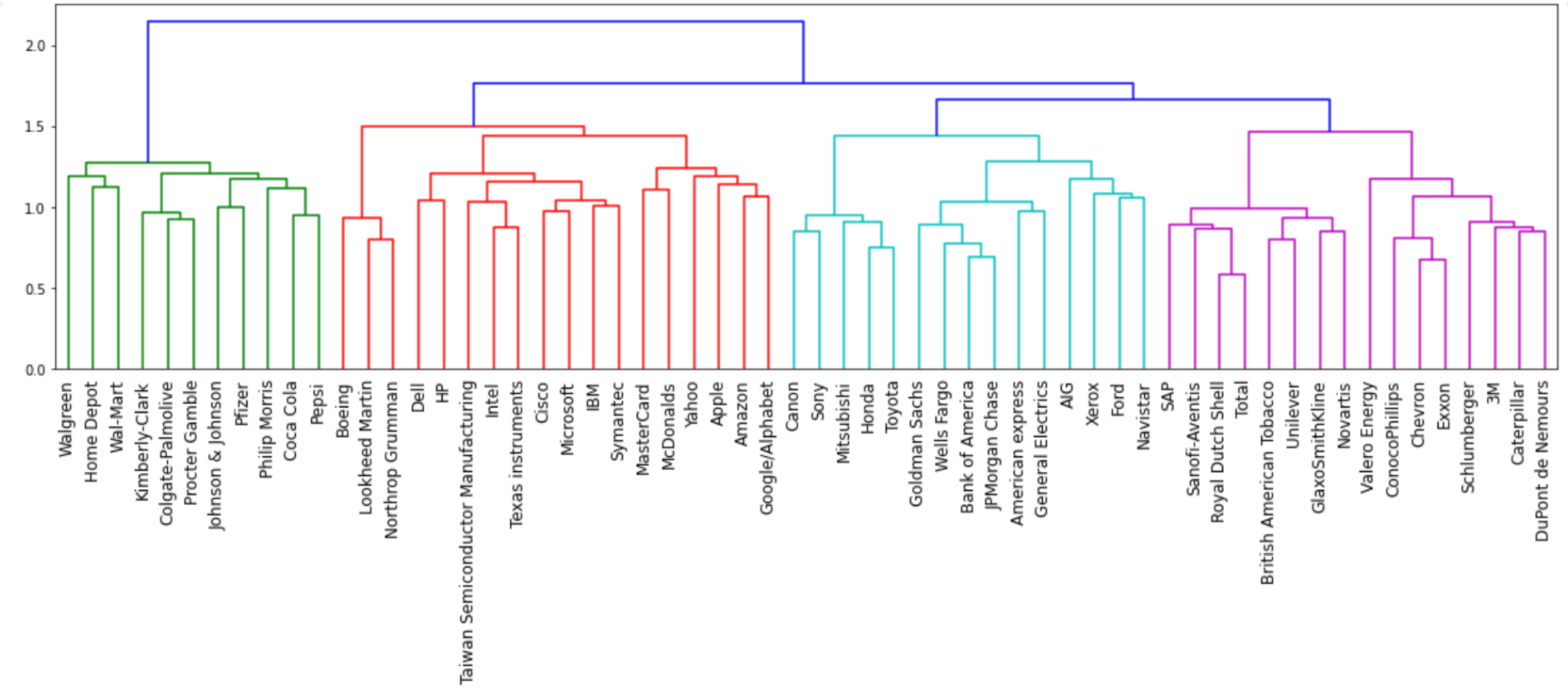
The linkage criterion determines which distance to use between sets of observation. The algorithm will merge the pairs of cluster that minimize this criterion.

- *ward* minimizes the variance of the clusters being merged.
- *average* uses the average of the distances of each observation of the two sets.
- *complete* or maximum linkage uses the maximum distances between all observations of the two sets.
- *single* uses the minimum of the distances between all observations of the two sets.

Perform agglomerative hierarchical clustering w/ different linkage

```
from sklearn.preprocessing import normalize
# normalize each row to be a unit vector
X_norm = normalize(X)

# try some other linkages
model = AgglomerativeClustering(linkage="ward",
                                distance_threshold=0,
                                n_clusters=None)
model = model.fit(X_norm)

links = get_linkage(model)

# Plot the dendrogram
fig, ax = plt.subplots(figsize=(20, 5))
dendrogram(links,
           labels=list(df['companies']),
           leaf_rotation=90,
           leaf_font_size=12,
           ax=ax);
```

Clustering result w/ normalized data and "ward" linkage



Try prediction

```
from sklearn.cluster import AgglomerativeClustering
from sklearn.preprocessing import normalize

# normalize each row to be a unit vector
X_norm = normalize(X)

model = AgglomerativeClustering(n_clusters=3,
                                linkage='complete')
y_pred = model.fit_predict(X_norm)
print('pred:', y_pred)
```

```
pred: [0 0 0 0 1 0 1 0 0 2 0 0 0 0 0 0 0 0 0 1 2 0 0 0 0 1 0 2 2 1 1 1 0 0 0 0 1
 1 2 1 2 2 1 1 0 0 1 0 0 1 0 0 1 0 1 0 2 0 0 0]
```

Estimate the number of clusters with Elbow

"Ward" linkage (minimizing variance of a joined cluster) is identical inspirit with K-means in a hierarchical clustering form.

```
from sklearn.cluster import AgglomerativeClustering
from sklearn.preprocessing import normalize

# normalize each row to be a unit vector
X_norm = normalize(X)

model = AgglomerativeClustering(linkage='ward',
                                distance_threshold=0,
                                n_clusters=None)
model.fit(X_norm)
```

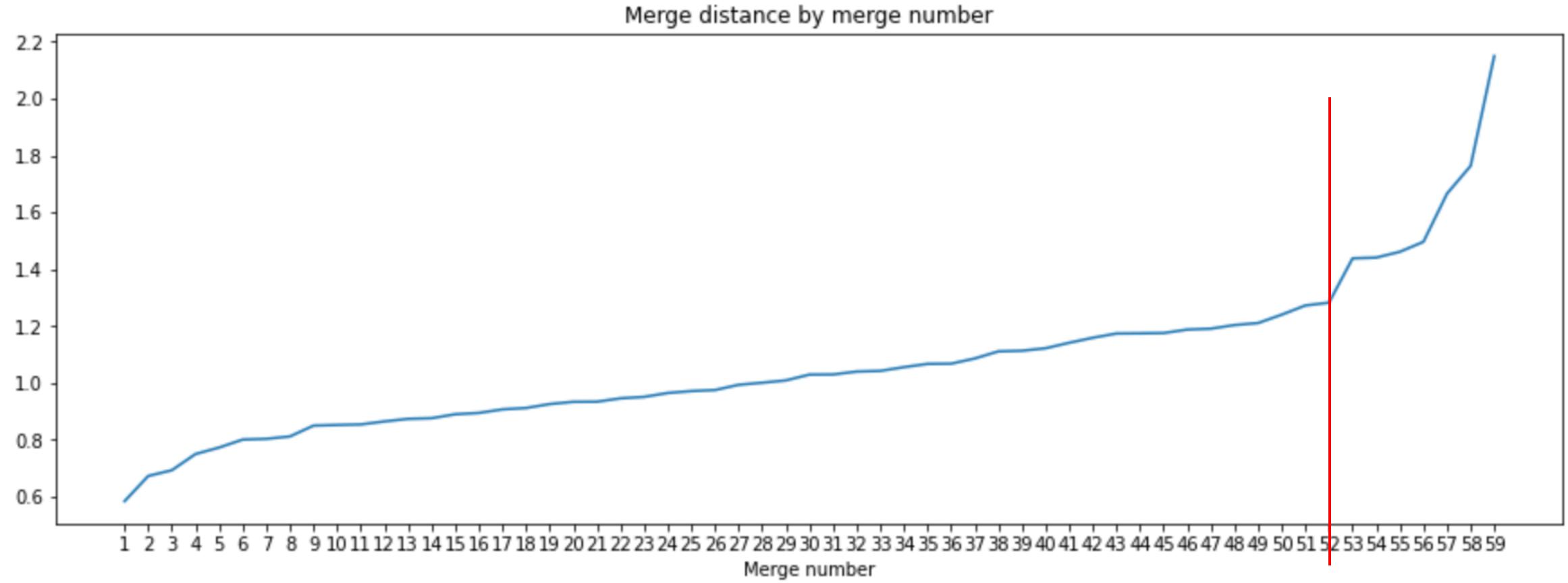
```
AgglomerativeClustering(affinity='euclidean', compute_full_tree='auto',
                        connectivity=None, distance_threshold=0, linkage='ward',
                        memory=None, n_clusters=None)
```

Plot the join distance

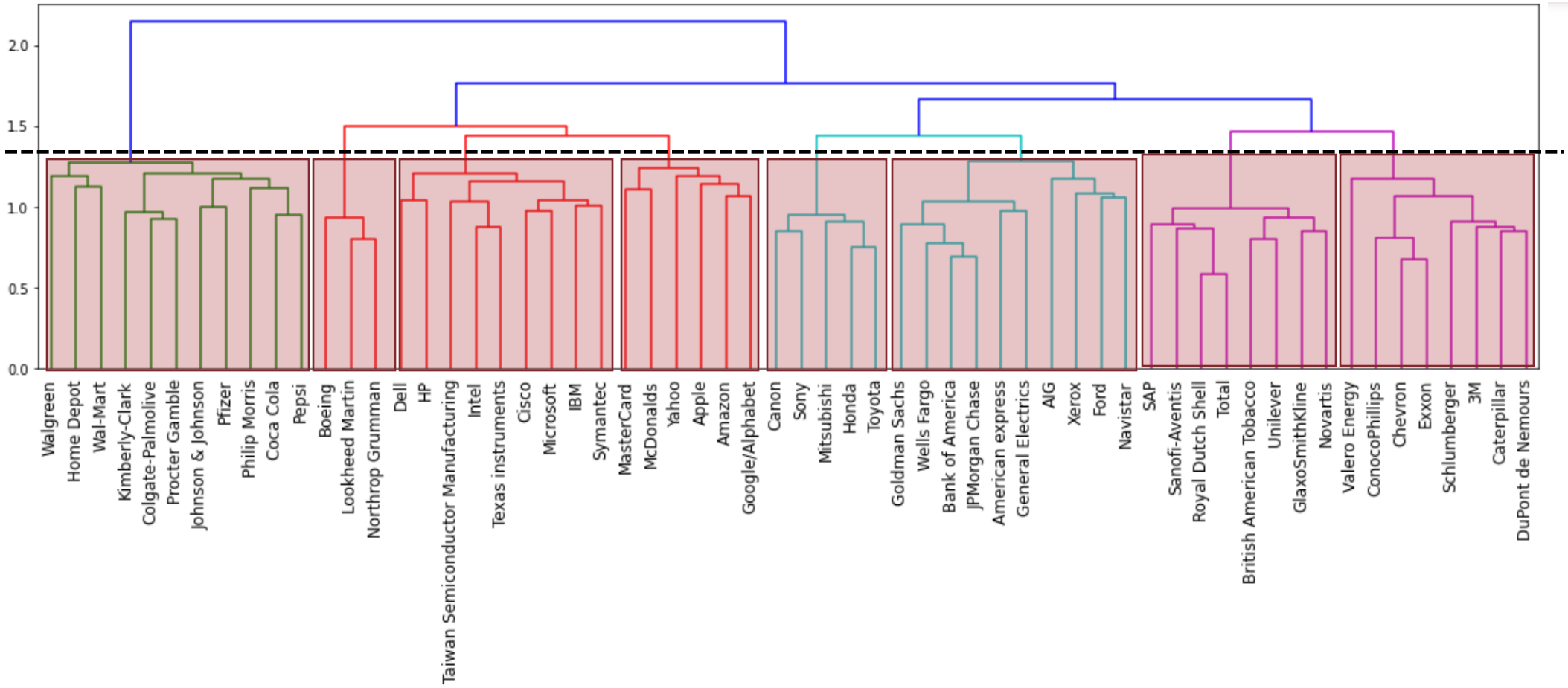
```
fig, ax = plt.subplots(figsize=(15, 5))
xticks = np.arange(1, 60)
ax.set_title('Merge distance by merge number')
ax.set_xticks(xticks)
ax.set_xlabel('Merge number')
ax.plot(xticks, model.distances_)
```

Plot the join distance

```
[<matplotlib.lines.Line2D at 0x7f6a53a758d0>]
```



Clustering result w/ normalized data and "ward" linkage



Try prediction

```
from sklearn.cluster import AgglomerativeClustering
from sklearn.preprocessing import normalize

# normalize each row to be a unit vector
X_norm = normalize(X)

# 8 seems to be the number of clusters (corresponds to 52 merges)
model = AgglomerativeClustering(linkage='ward',
                                n_clusters=8)
y_pred = model.fit_predict(X_norm)
print(y_pred)
```

```
[3 0 3 0 4 0 5 7 6 1 6 2 6 6 2 0 0 3 0 5 1 7 2 2 2 1 0 1 1 4 3 3 6 2 7 0 4
 5 1 1 1 1 5 5 6 7 5 2 7 5 2 2 5 6 1 0 1 6 0 3]
```

Explore prediction results

```
df2 = pd.DataFrame({
    'companies': df['companies'],
    'cluster': y_pred,
})
df2.head()
```

	companies	cluster
0	Apple	3
1	AIG	0
2	Amazon	3
3	American express	0
4	Boeing	4

Group by industries

	companies	cluster
15	Ford	0
1	AIG	0
3	American express	0
55	Wells Fargo	0
5	Bank of America	0
18	Goldman Sachs	0
16	General Electrics	0
26	JPMorgan Chase	0
35	Navistar	0
58	Xerox	0
20	Home Depot	1
38	Pepsi	1
39	Pfizer	1
25	Johnson & Johnson	1

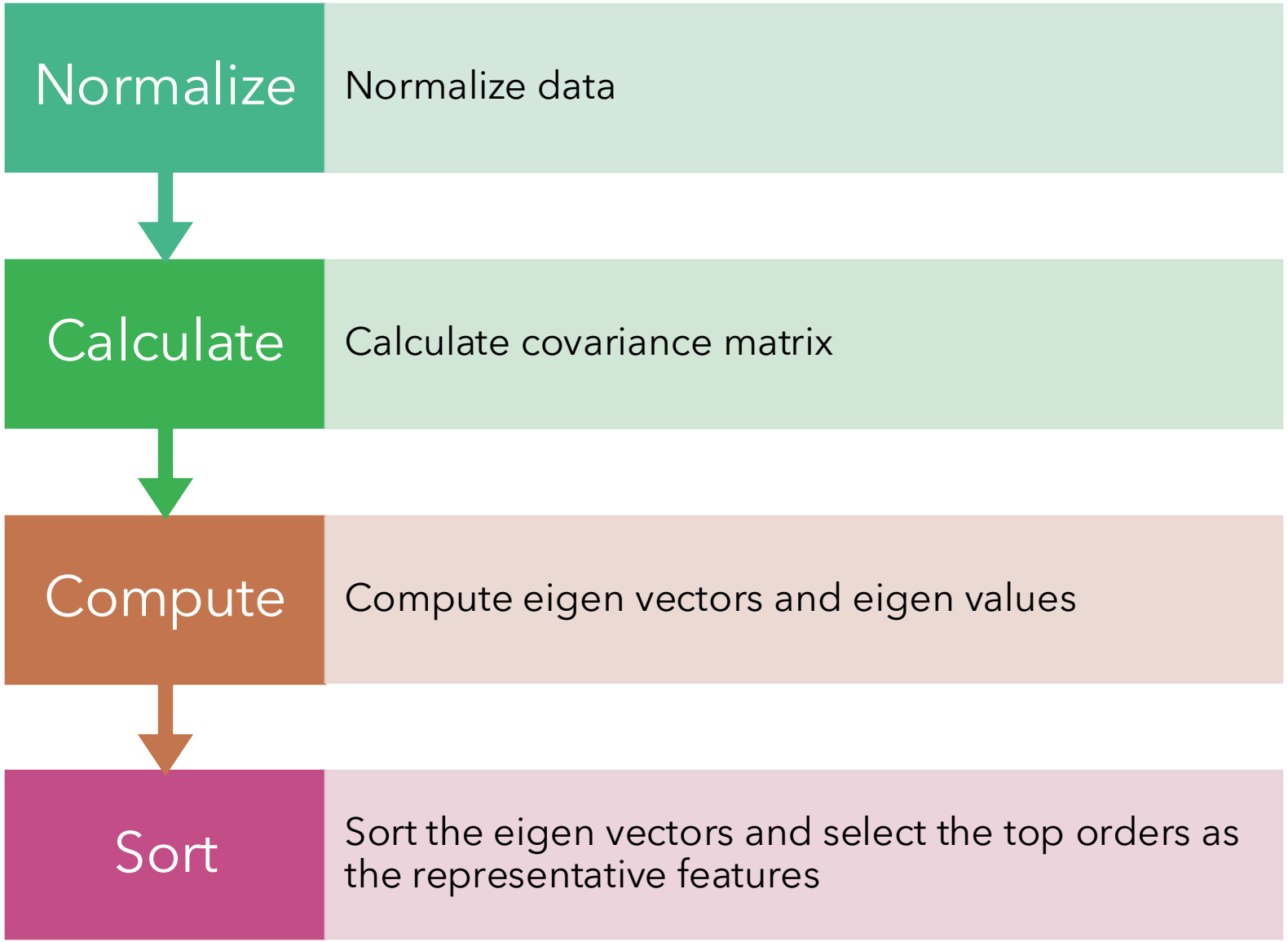
```
df2.sort_values('cluster')
```

Principle Component Analysis (PCA)

PCA

- PCA = “Principal Component Analysis”
- Fundamental *dimension reduction* technique to reduce feature redundancy and compress feature space

PCA standard steps



PCA aligns data with axes

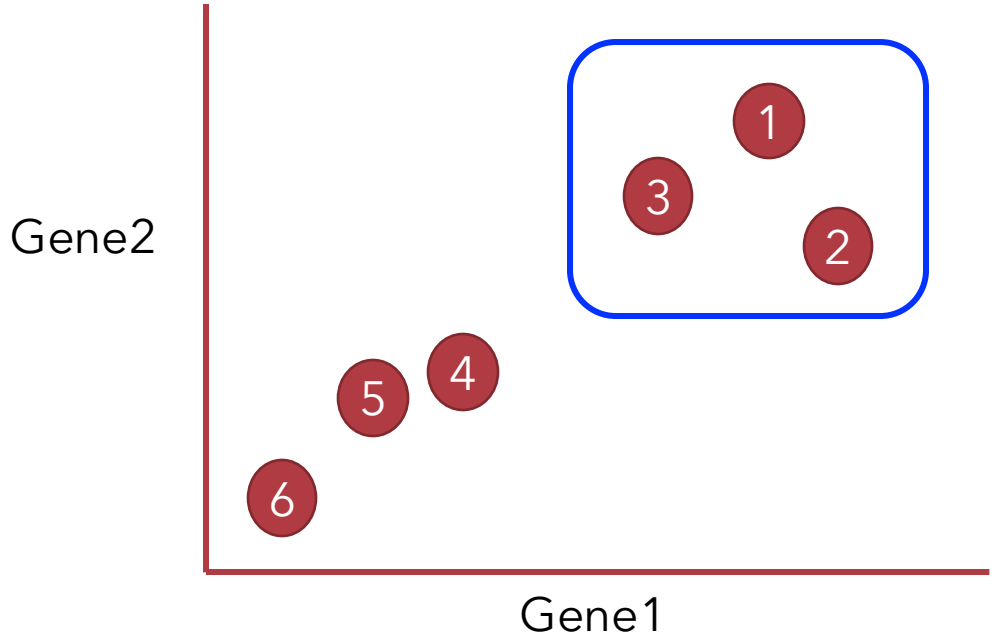
	Business1	Business2	Business3	…
Market cap	9.5 M	88 M	93 M	…
#Employees	960	79	9800	…

	Mouse 1	Mouse 2	Mouse 3	Mouse 4	Mouse 5	Mouse 6
Gene1	10	11	8	3	2	1

PCA aligns data with axes

If we measure two genes (2 variables)

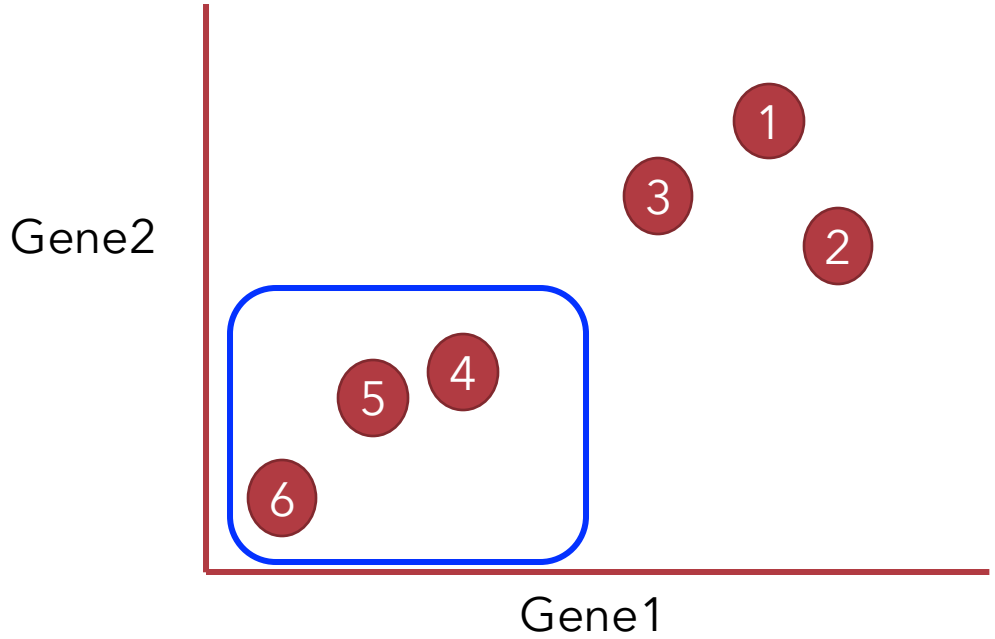
	Mouse 1	Mouse 2	Mouse 3	Mouse 4	Mouse 5	Mouse 6
Gene1	10	11	8	3	2	1
Gene2	6	4	5	3	2.8	1



PCA aligns data with axes

If we measure two genes (2 variables)

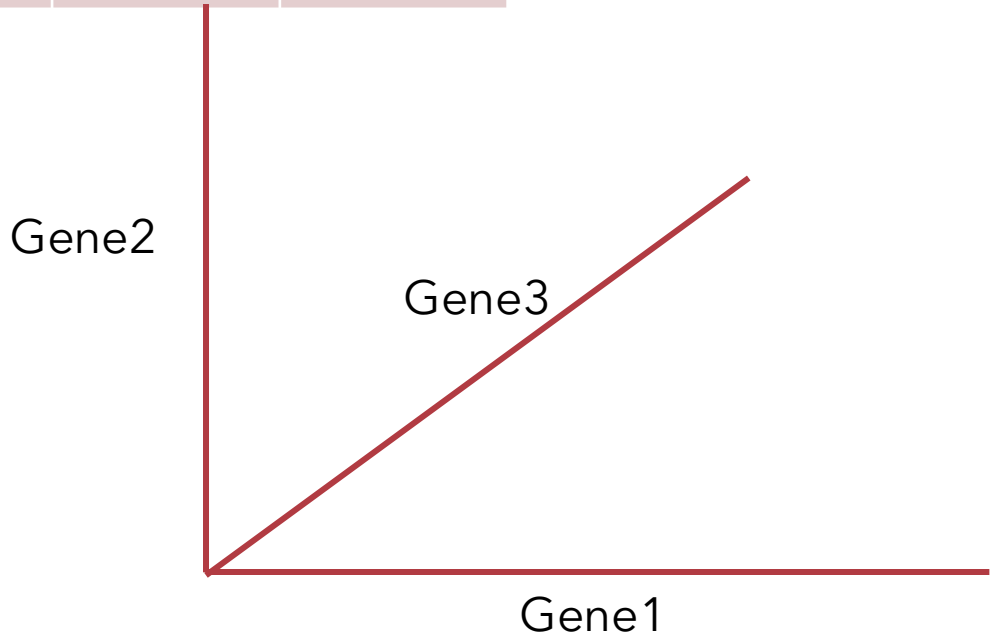
	Mouse 1	Mouse 2	Mouse 3	Mouse 4	Mouse 5	Mouse 6
Gene1	10	11	8	3	2	1
Gene2	6	4	5	3	2.8	1



PCA aligns data with axes

If we measure two genes (2 variables)

	Mouse 1	Mouse 2	Mouse 3	Mouse 4	Mouse 5	Mouse 6
Gene1	10	11	8	3	2	1
Gene2	6	4	5	3	2.8	1
Gene3	12	9	10	2.5	1.3	2



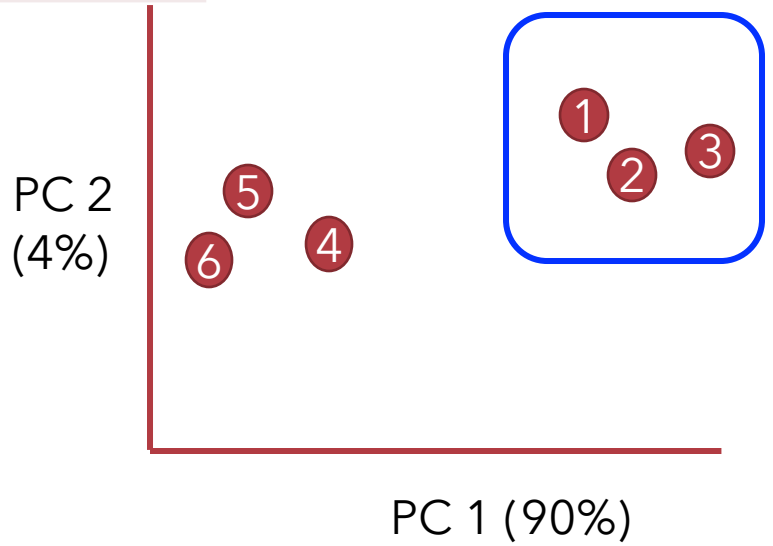
PCA aligns data with axes

If we measure two genes (2 variables)

	Mouse 1	Mouse 2	Mouse 3	Mouse 4	Mouse 5	Mouse 6
Gene1	10	11	8	3	2	1
Gene2	6	4	5	3	2.8	1
Gene3	12	9	10	2.5	1.3	2
Gene4	5	7	6	2	4	7

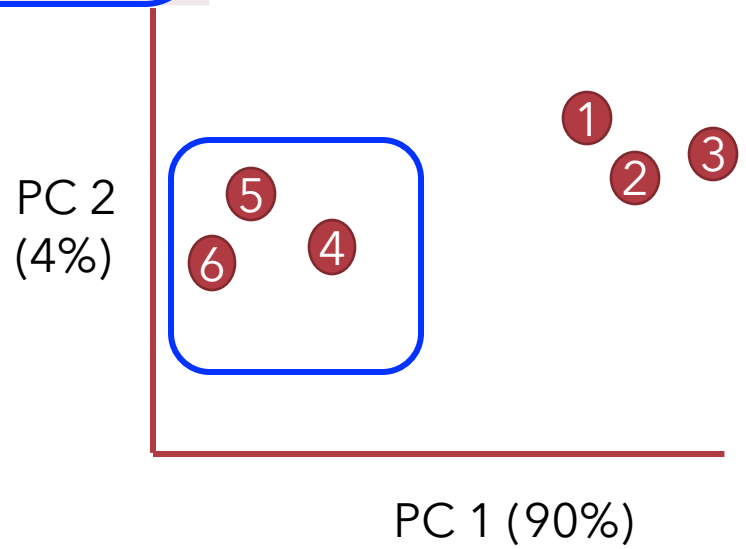
How PCA can take 4 or more feature measurements and make a 2D PCA plot

	Mouse 1	Mouse 2	Mouse 3	Mouse 4	Mouse 5	Mouse 6
Gene1	10	11	8	3	2	1
Gene2	6	4	5	3	2.8	1
Gene3	12	9	10	2.5	1.3	2
Gene4	5	7	6	2	4	7

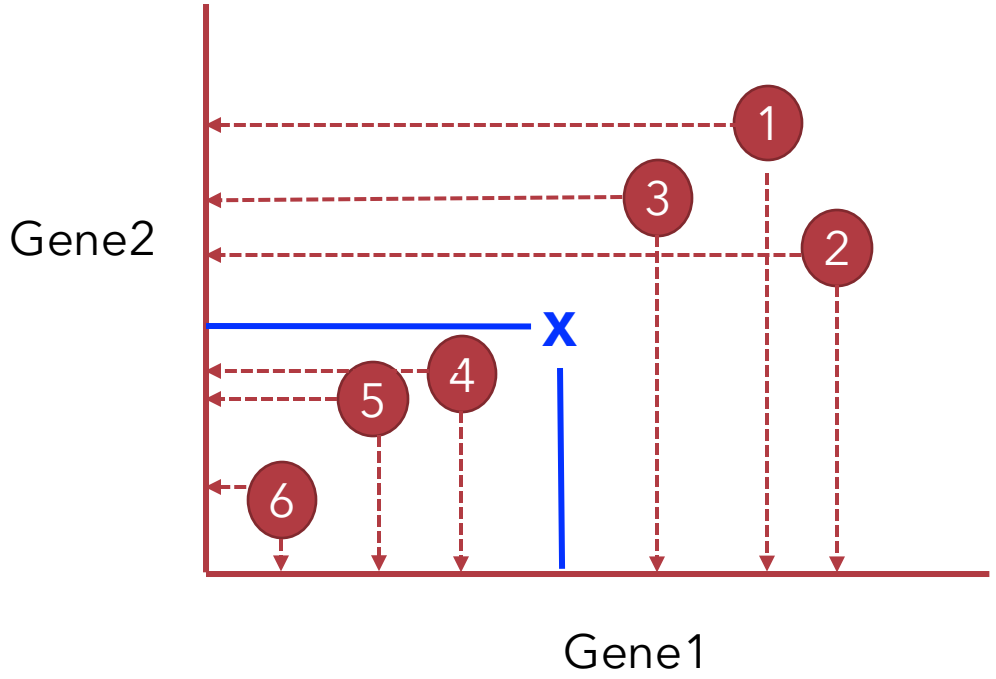


How PCA can take 4 or more feature measurements and make a 2D PCA plot

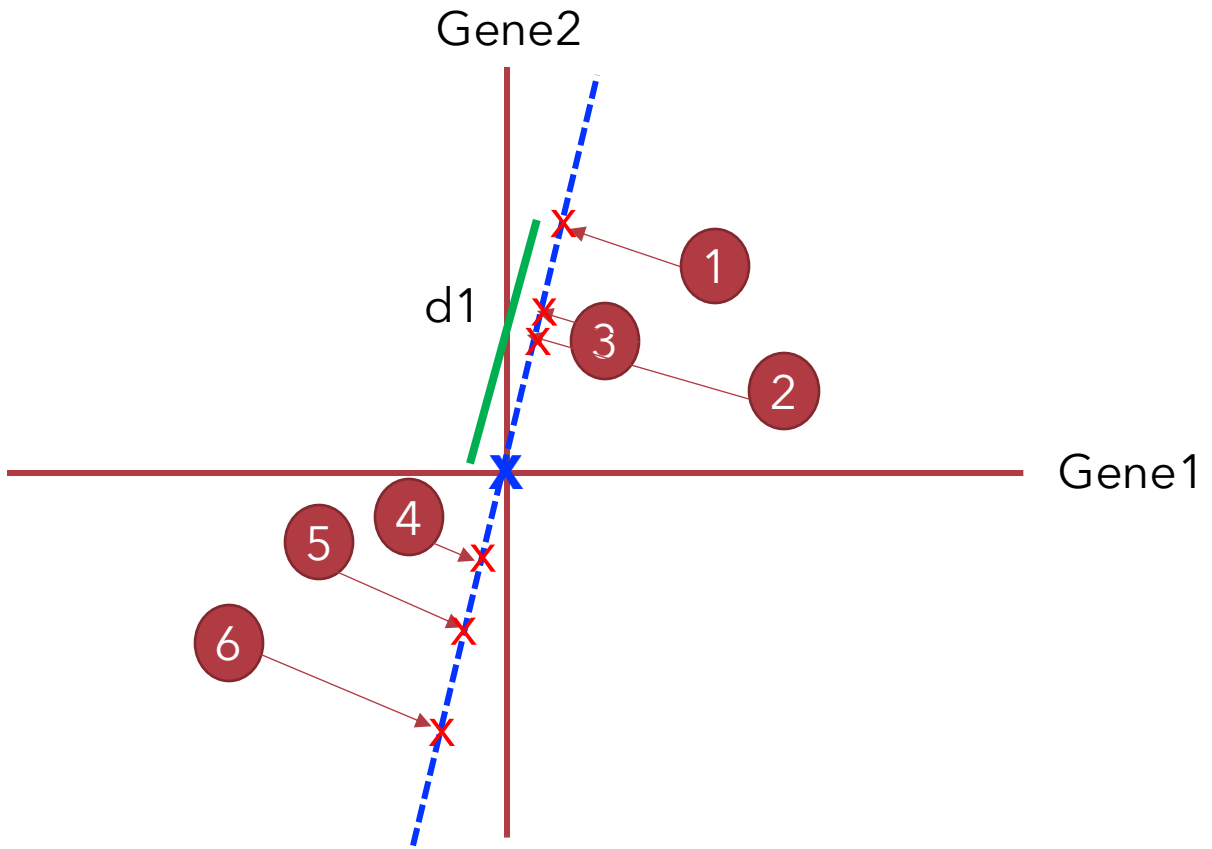
	Mouse 1	Mouse 2	Mouse 3	Mouse 4	Mouse 5	Mouse 6
Gene1	10	11	8	3	2	1
Gene2	6	4	5	3	2.8	1
Gene3	12	9	10	2.5	1.3	2
Gene4	5	7	6	2	4	7



PCA aligns data with axes
If we measure two genes (2 variables)

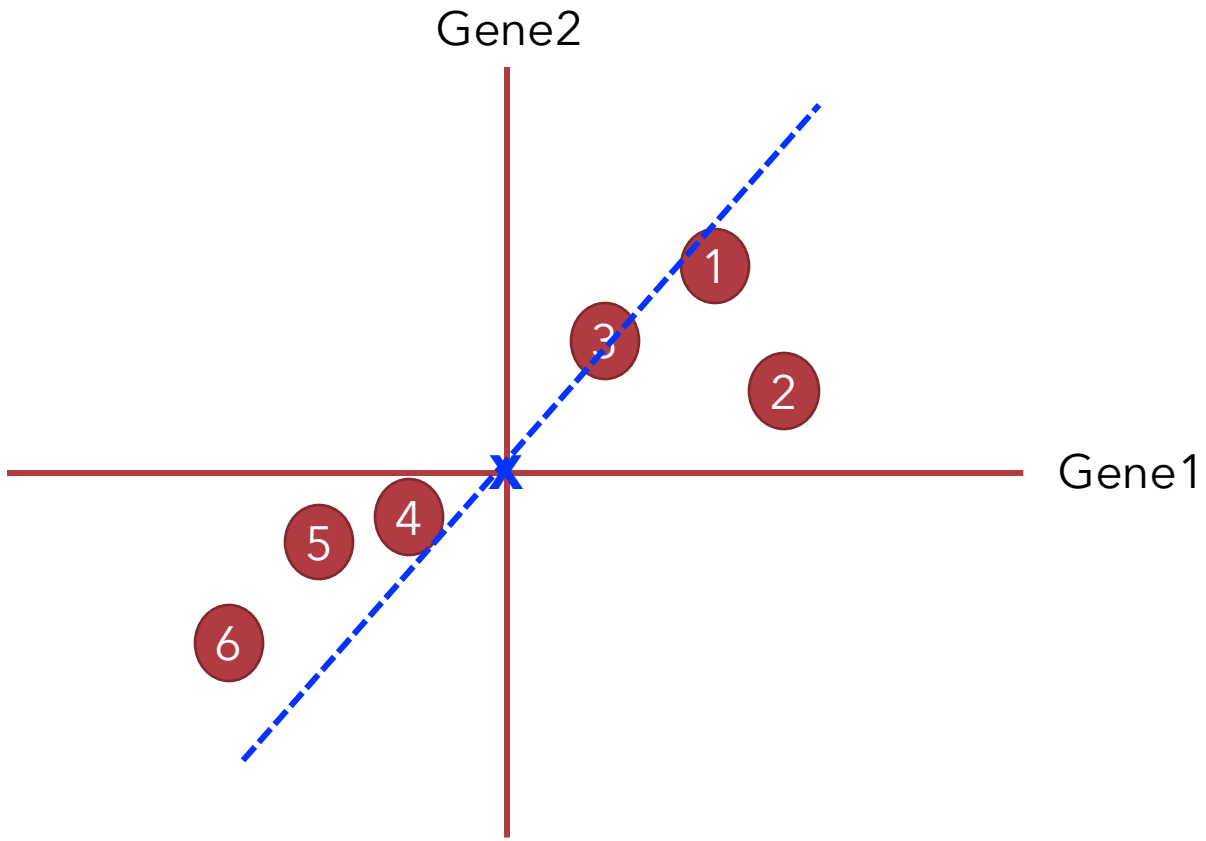


PCA aligns data with axes
If we measure two genes (2 variables)

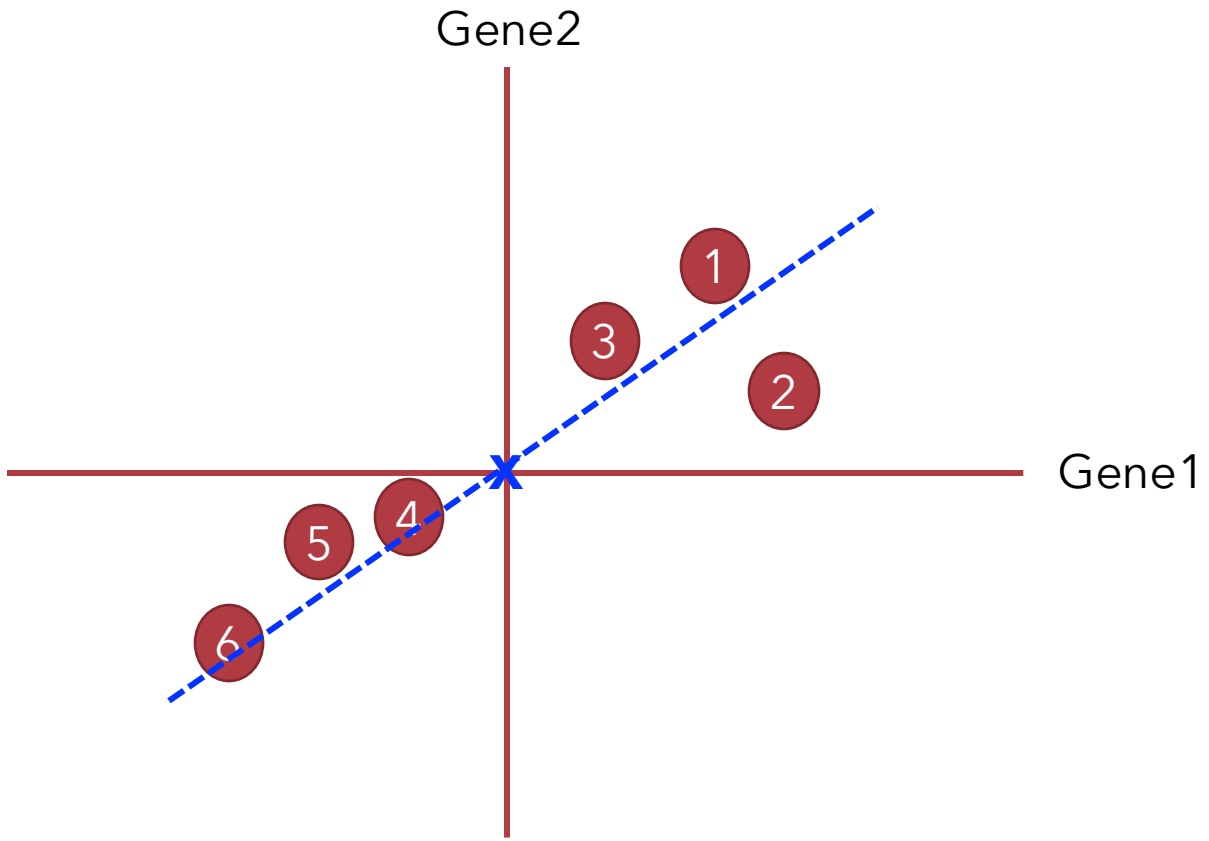


$d1^2 + d2^2 + d3^2 + d4^2 + d5^2 + d6^2$ = sum of squared distance = SS(distances)

PCA aligns data with axes
If we measure two genes (2 variables)

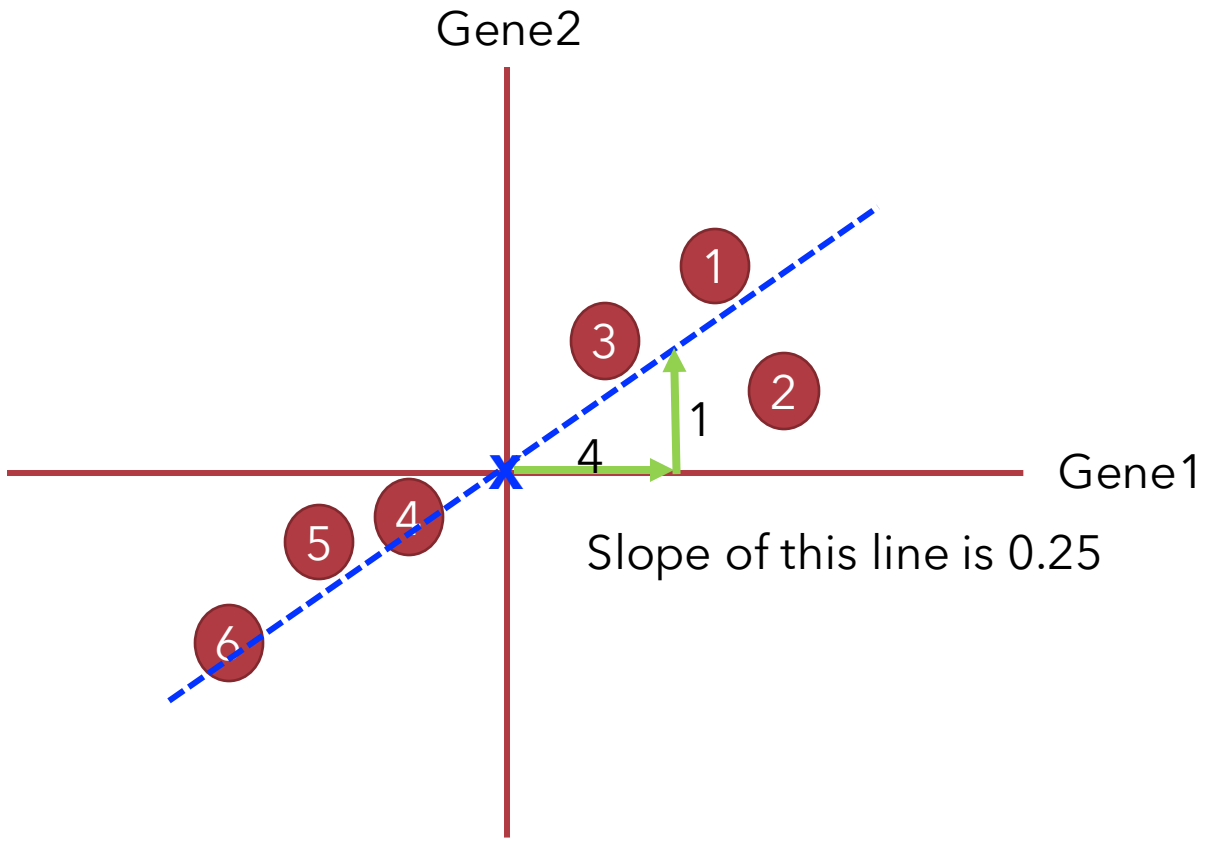


PCA aligns data with axes
If we measure two genes (2 variables)



Largest sum of squared distance = SS(distances) => Principal Component 1 (PC1)

PCA aligns data with axes
If we measure two genes (2 variables)



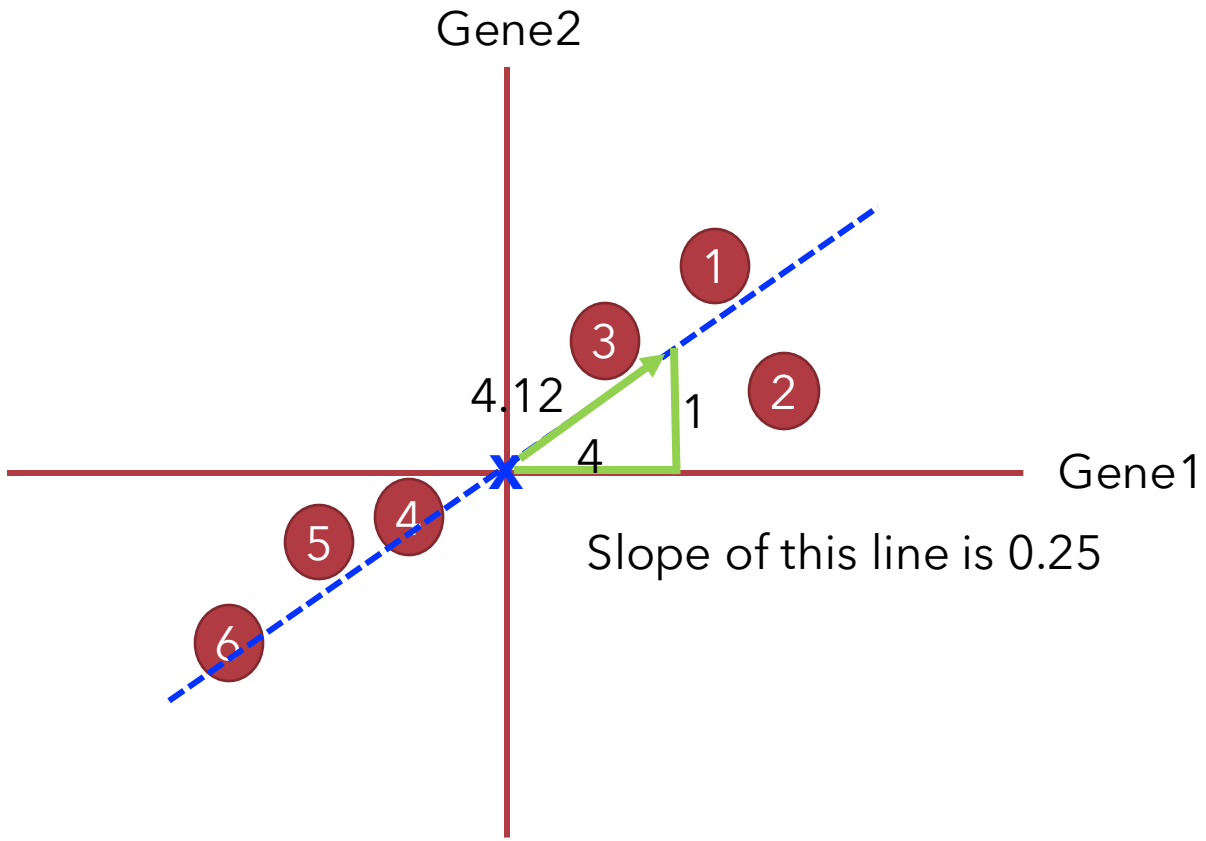
To make PC1
Mix 4 parts of Gene1
and 1 part of Gene 2

Linear combination between
Gene1 and Gene2

The ratio of Gene1
and Gene2 indicates that Gene1
is more important than Gene2

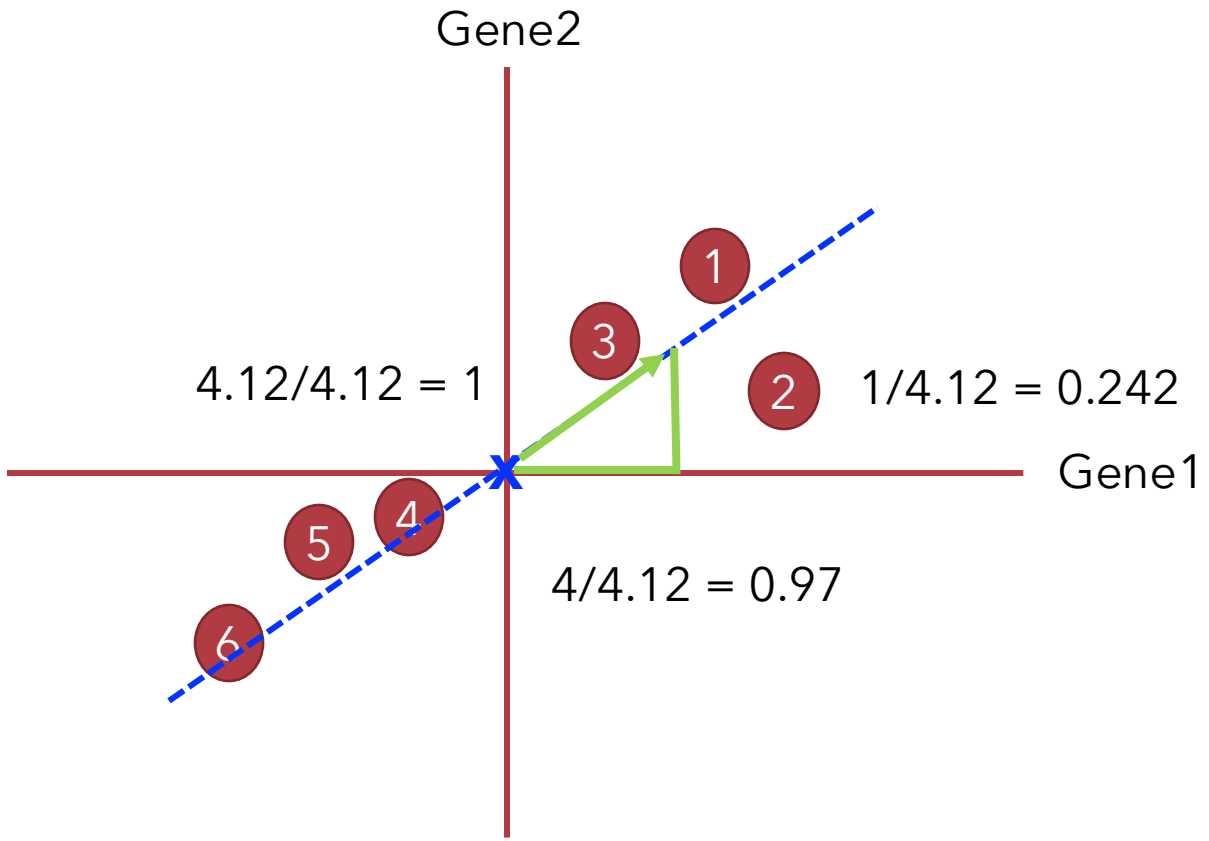
Largest sum of squared distance = SS(distances) => Principal Component 1 (PC1)

PCA aligns data with axes
If we measure two genes (2 variables)



Largest sum of squared distance = SS(distances) => Principal Component 1 (PC1)

PCA aligns data with axes
If we measure two genes (2 variables)



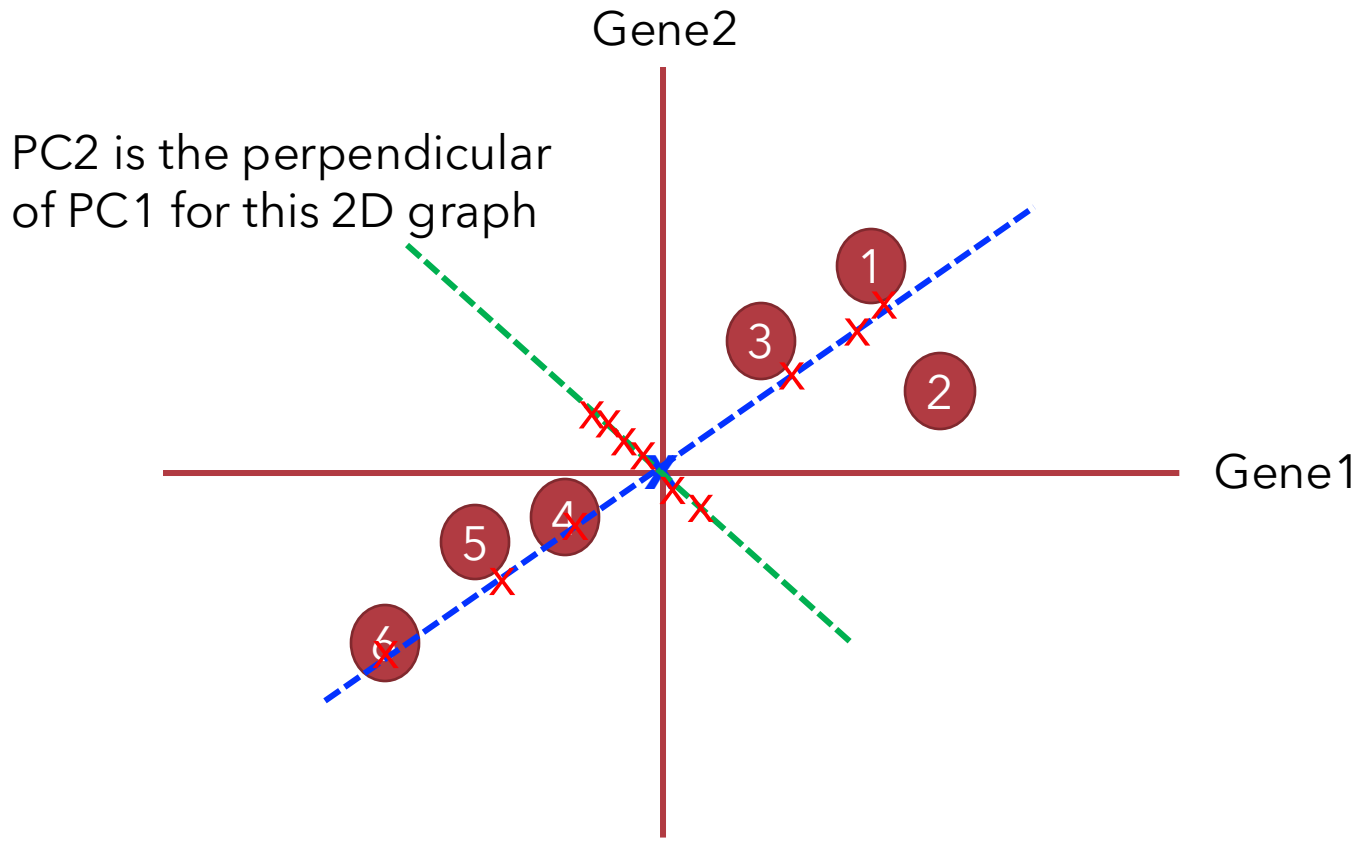
To make PC1
Mix 0.970 of Gene1
and 0.242 of Gene 2

1 unit vector of
0.970 parts Gene1 and
0.242 parts Gene2 is called
"Singular Vector" or the
"Eigenvector" for PC1

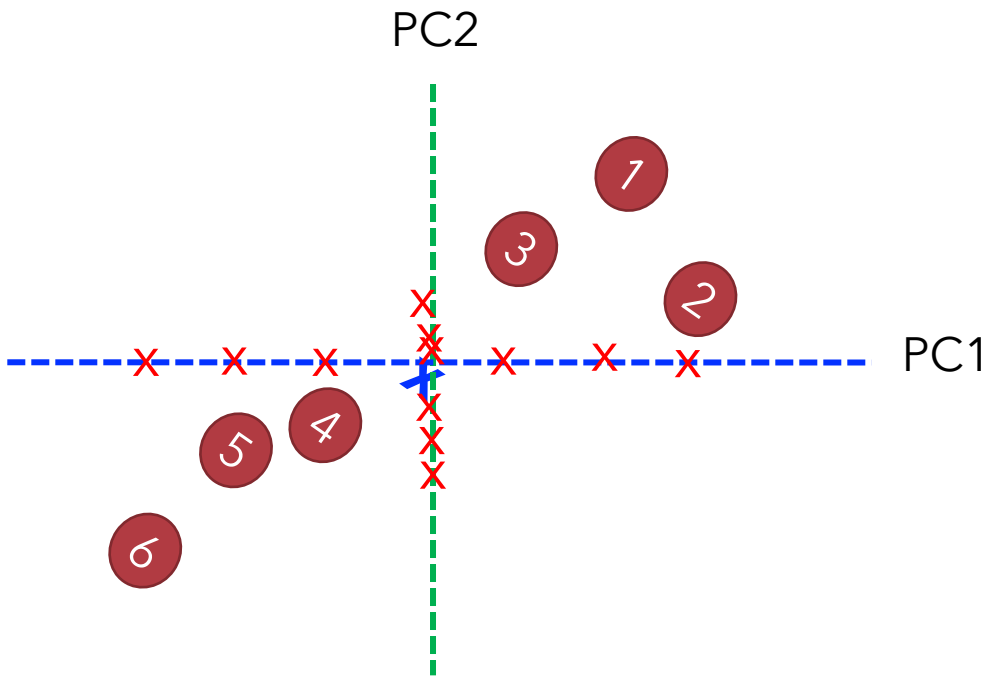
0.97 and 0.242 are
called loading scores

Largest sum of squared distance = SS(distances) => Principal Component 1 (PC1) = Eigenvalue for PC1

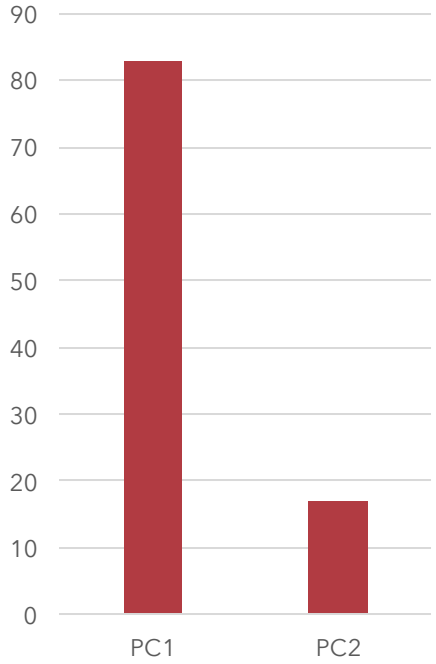
PCA aligns data with axes
If we measure two genes (2 variables)



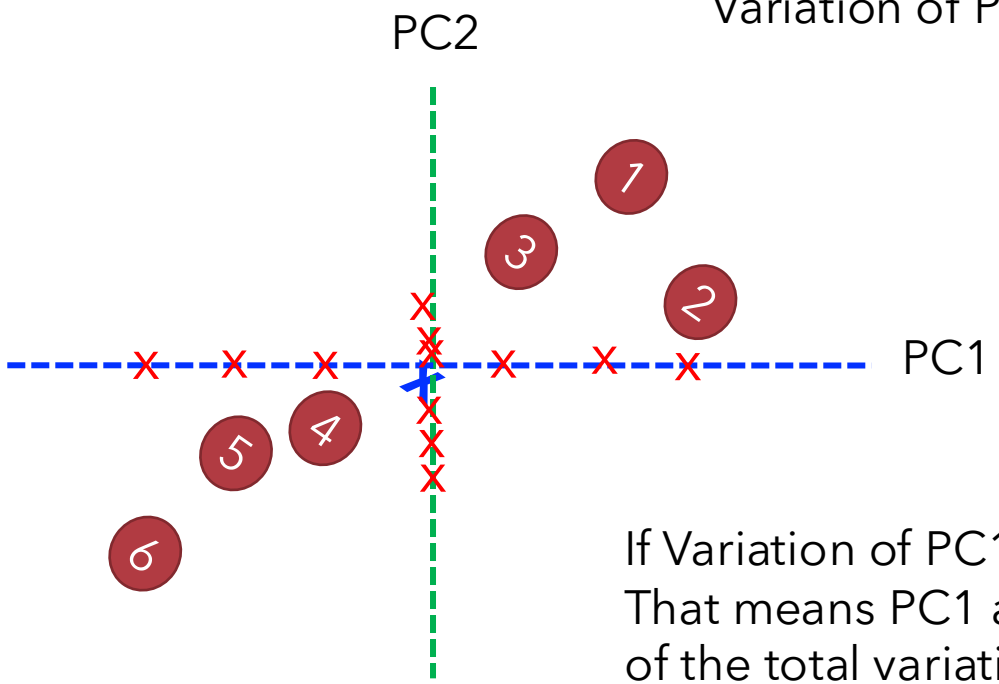
PCA aligns data with axes
If we measure two genes (2 variables)



Variation for the PCs



Scree plot

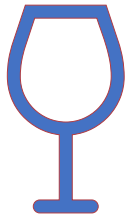


Variation of PC1 = SS(distance for PC1) / (n-1)
Variation of PC2 = SS(distance for PC2) / (n-1)

If Variation of PC1 = 15 and Variation of PC2 = 3
That means PC1 accounts for 15/18 = 0.83 or 83%
of the total variation around the PCs.
PC2 accounts for 3/18 = 0.17 or 17% in this example.

PCA for wine data dimension reduction

https://colab.research.google.com/drive/1rfjsRTTw5tT5qhfFpjBRfEqTR2WmcyRJ#scrollTo=pxyKOwajk3Xw



class 1 59
class 2 71
class 3 48

Wine dataset

The attributes are (dontated by Riccardo Leardi, riclea@anchem.unige.it)

1) Alcohol
2) Malic acid
3) Ash
4) Alcalinity of ash
5) Magnesium
6) Total phenols
7) Flavanoids
8) Nonflavanoid phenols
9) Proanthocyanins
10)Color intensity
11)Hue
12)OD280/OD315 of diluted wines
13)Proline

Data loading

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.cluster import KMeans

# Wine data
# 178 samples with 3 distinct varieties of red wine: Barolo, Grignolino and Barbera
# Features measure chemical composition e.g. alcohol content
# https://archive.ics.uci.edu/ml/datasets/Wine
url = 'https://raw.githubusercontent.com/wichadak/wine/master/wine.data'
df = pd.read_csv(url, header=None)
df.sample(frac=1).head()
```

Wine data

Features or dimensions

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
85	2	12.67	0.98	2.24	18.0	99	2.20	1.94	0.30	1.46	2.62	1.23	3.16	450
66	2	13.11	1.01	1.70	15.0	78	2.98	3.18	0.26	2.28	5.30	1.12	3.18	502
21	1	12.93	3.80	2.65	18.6	102	2.41	2.41	0.25	1.98	4.50	1.03	3.52	770
83	2	13.05	3.86	2.32	22.5	85	1.65	1.59	0.61	1.62	4.80	0.84	2.01	515
97	2	12.29	1.41	1.98	16.0	85	2.55	2.50	0.29	1.77	2.90	1.23	2.74	428

samples

Wine type

Data exploration

```
df.describe()
```

	0	1	2	3	4	5	6	7	8
count	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000
mean	1.938202	13.000618	2.336348	2.366517	19.494944	99.741573	2.295112	2.029270	0.361854
std	0.775035	0.811827	1.117146	0.274344	3.339564	14.282484	0.625851	0.998859	0.124453
min	1.000000	11.030000	0.740000	1.360000	10.600000	70.000000	0.980000	0.340000	0.130000
25%	1.000000	12.362500	1.602500	2.210000	17.200000	88.000000	1.742500	1.205000	0.270000
50%	2.000000	13.050000	1.865000	2.360000	19.500000	98.000000	2.355000	2.135000	0.340000
75%	3.000000	13.677500	3.082500	2.557500	21.500000	107.000000	2.800000	2.875000	0.437500
max	3.000000	14.830000	5.800000	3.230000	30.000000	162.000000	3.880000	5.080000	0.660000

Data preparation

```
X = df[df.columns[1:]].to_numpy(dtype=np.float32)
y = df[df.columns[0]].to_numpy()
print('X:', X.shape, X.dtype)
print('y:', y.shape, y.dtype)
```

```
X: (178, 13) float32
y: (178,) int64
```

PCA to decorrelate features

```
from sklearn.decomposition import PCA

# Create PCA instance: model
model = PCA()

# Apply the fit_transform method of model to grains: pca_features
X_pca = model.fit_transform(X)
print('X_pca:', X_pca.shape)

# describe
pd.DataFrame(X_pca).describe()
```

PCA to decorrelate features

X_pca: (178, 13)

	0	1	2	3	4	5	6	
count	178.000000	178.000000	1.780000e+02	1.780000e+02	1.780000e+02	1.780000e+02	1.780000e+02	1.780000e
mean	-0.000021	0.000003	1.151910e-07	1.446585e-07	2.817303e-07	-2.534872e-07	8.394879e-07	3.458241e
std	314.963104	13.135269	3.072152e+00	2.234093e+00	1.108533e+00	9.170942e-01	5.281796e-01	3.890775e
min	-469.059296	-27.504026	-9.005075e+00	-5.999351e+00	-3.319481e+00	-1.954749e+00	-1.568975e+00	-1.269572e
25%	-246.139977	-8.330378	-1.852034e+00	-1.339166e+00	-6.417685e-01	-6.234645e-01	-3.426479e-01	-2.776329e
50%	-73.525394	-3.269235	1.496602e-02	3.110407e-01	8.229639e-02	-9.193974e-02	-4.381787e-03	3.975998e
75%	238.565544	5.449198	2.152860e+00	1.481671e+00	6.736645e-01	5.145356e-01	3.724663e-01	2.548769e
max	933.118286	58.793758	9.947130e+00	6.461738e+00	3.811527e+00	3.596516e+00	1.843119e+00	1.403566e

Visualize the correlation (before and after)

Define two functions for visualization

```
# https://towardsdatascience.com/better-heatmaps-and-correlation-matrix-plots-in-python-41445d0f2bec
def heatmap(x, y, size, ax=None):
    if ax is None:
      fig, ax = plt.subplots()

    # Mapping from column names to integer coordinates
    x_labels = [v for v in sorted(x.unique())]
    y_labels = [v for v in sorted(y.unique())]
    x_to_num = {p[1]:p[0] for p in enumerate(x_labels)}
    y_to_num = {p[1]:p[0] for p in enumerate(y_labels)}

    size_scale = 500
    ax.scatter(
        x=x.map(x_to_num), # Use mapping for x
        y=y.map(y_to_num), # Use mapping for y
        s=size * size_scale, # Vector of square sizes, proportional to size parameter
        marker='s' # Use square as scatterplot marker
    )

    # Show column labels on the axes
    ax.set_xticks([x_to_num[v] for v in x_labels])
    ax.set_xticklabels(x_labels, rotation=45, horizontalalignment='right')
    ax.set_yticks([y_to_num[v] for v in y_labels])
    ax.set_yticklabels(y_labels)
    return ax
```

Define function for visualizing correlation map

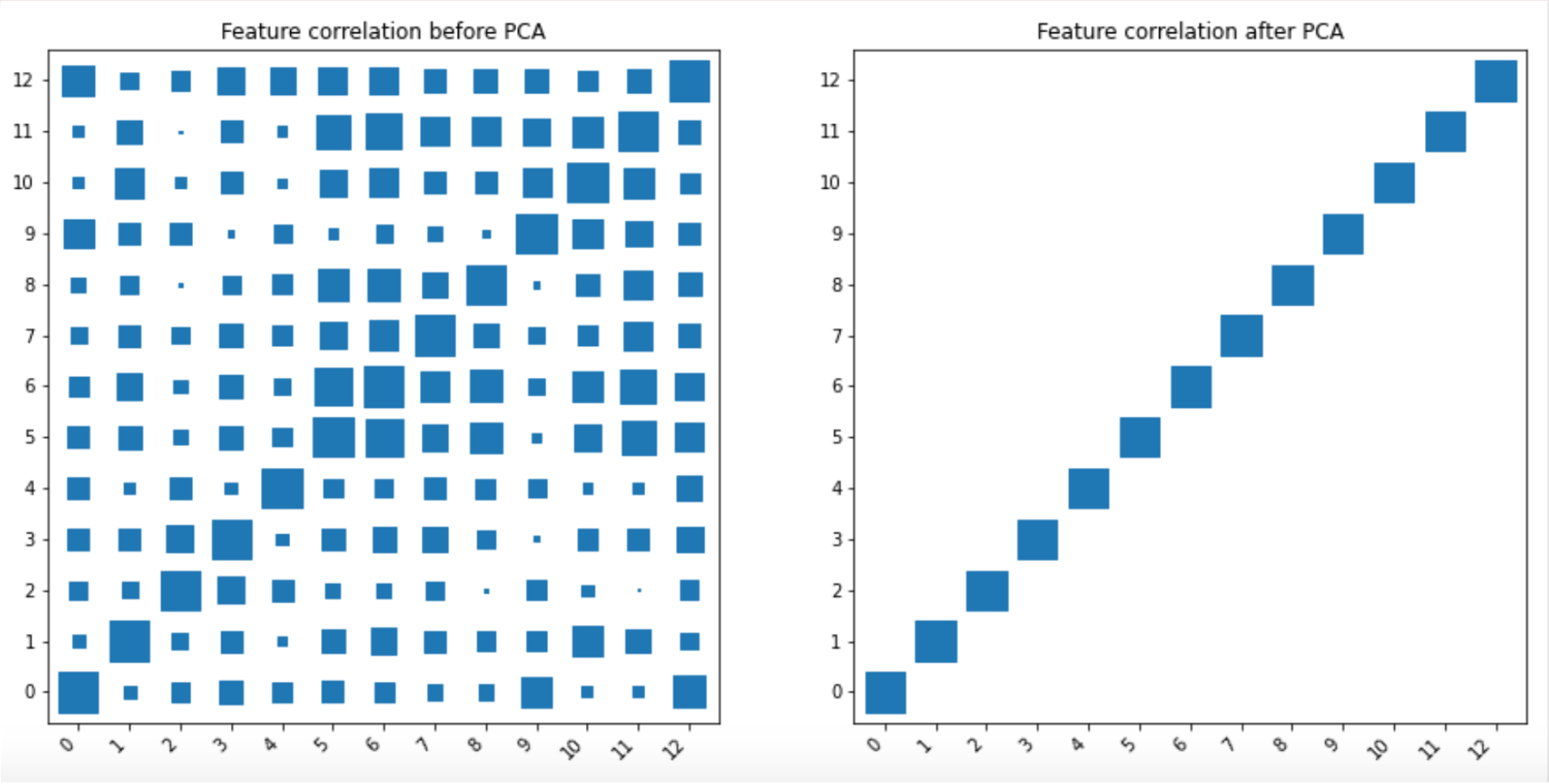
Define two functions for visualization

```
def correlation_map(X, ax=None):
  df = pd.DataFrame(X)
  corr = df.corr()
  corr = pd.melt(corr.reset_index(), id_vars='index') # Unpivot the dataframe, so we can get pair of
  corr.columns = ['x', 'y', 'value']
  heatmap(
      x=corr['x'],
      y=corr['y'],
      size=corr['value'].abs(),
      ax=ax,
  )
```

Visualize the correlation (before and after)

```
fig, ax = plt.subplots(ncols=2, figsize=(15, 7))
ax[0].set_title('Feature correlation before PCA')
ax[1].set_title('Feature correlation after PCA')
correlation_map(X, ax[0])
correlation_map(X_pca, ax[1])
```

Visualize the correlation (before and after PCA)



PCA to reduce dimension of the data

```
# reduce to 2 dimensions
model = PCA(n_components=2)
# Fit model to points
X_pca = model.fit_transform(X)
print('X_pca:', X_pca.shape)
```

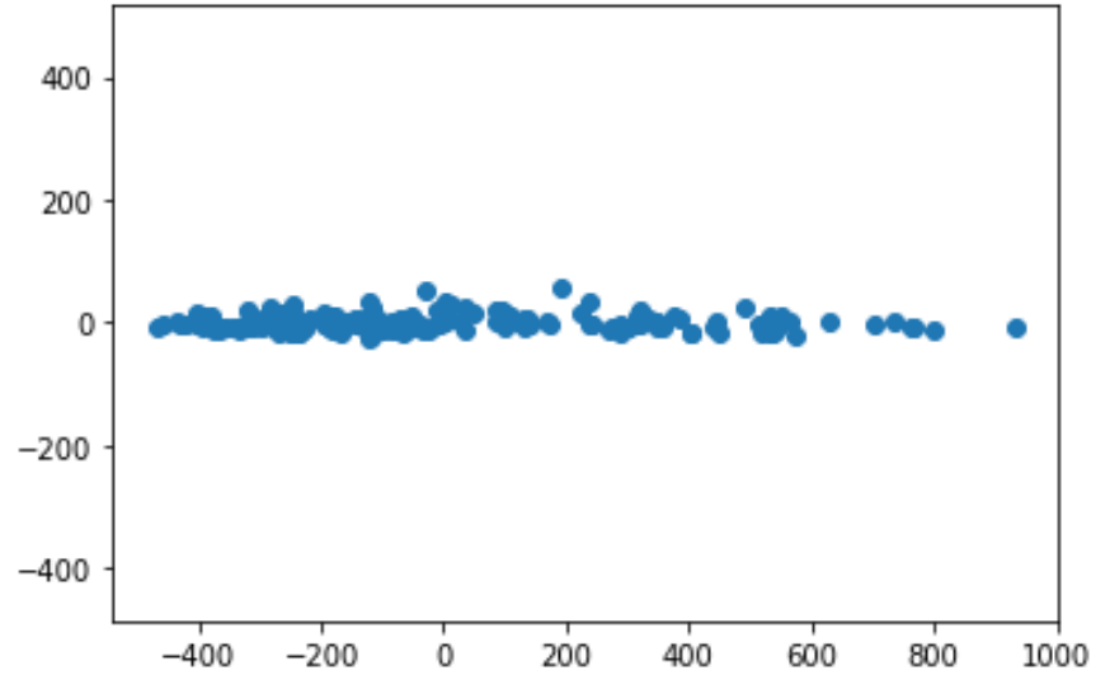
```
X_pca: (178, 2)
```

13 features were reduced to 2 principal components

2D scatter plot

```
plt.axis('equal')
plt.scatter(X_pca[:,0], X_pca[:,1])
```

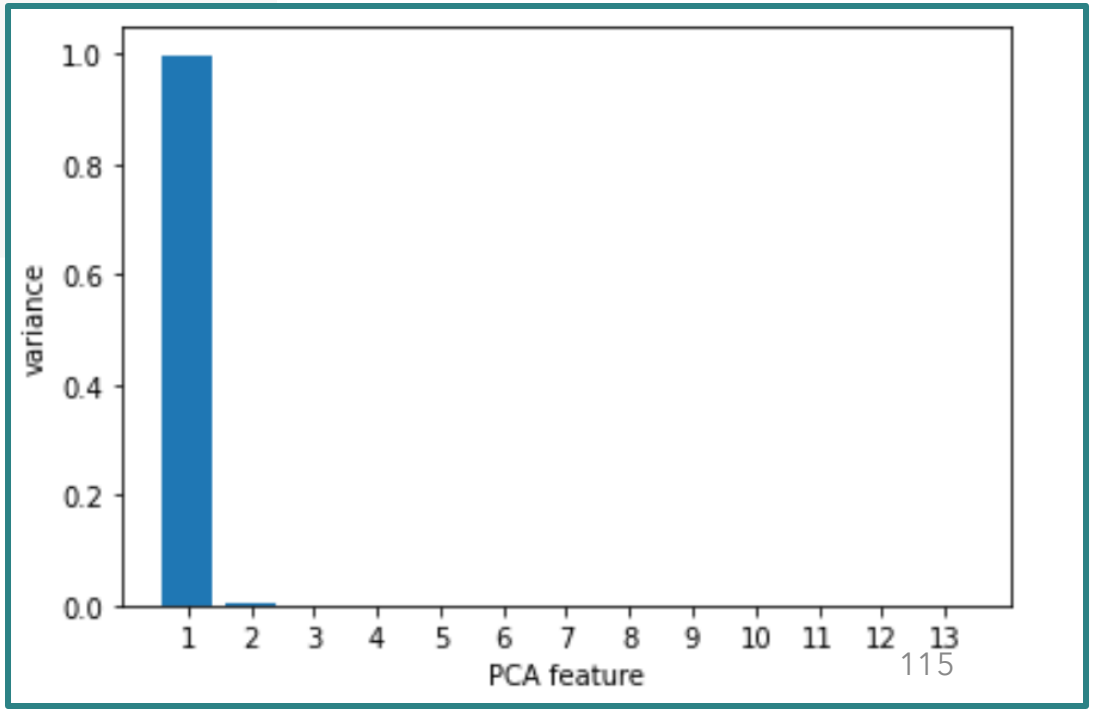
```
<matplotlib.collections.PathCollection at 0x7f26b59dcb38>
```



How good are the principal components?

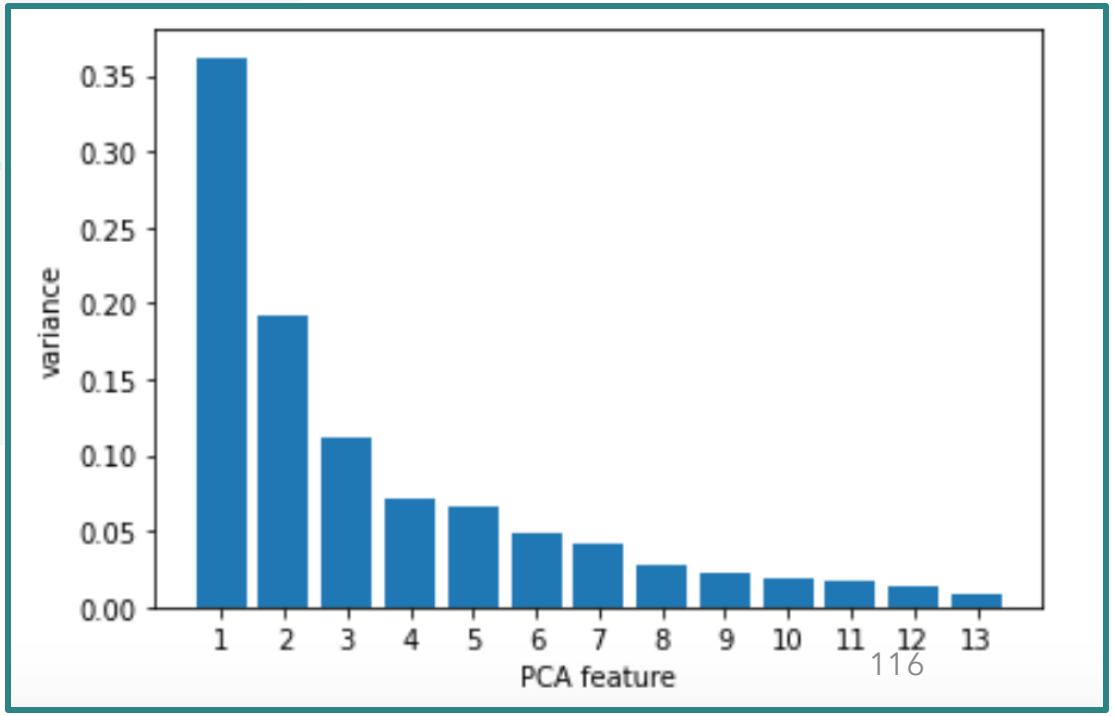
```
model = PCA()
X_pca = model.fit_transform(X)
cols = list(df.columns[1:])
plt.bar(cols, model.explained_variance_ratio_)
plt.xlabel('PCA feature')
plt.ylabel('variance')
plt.xticks(cols)
plt.show()
```

This is unrealistic



Normalize the data before reduce dimension

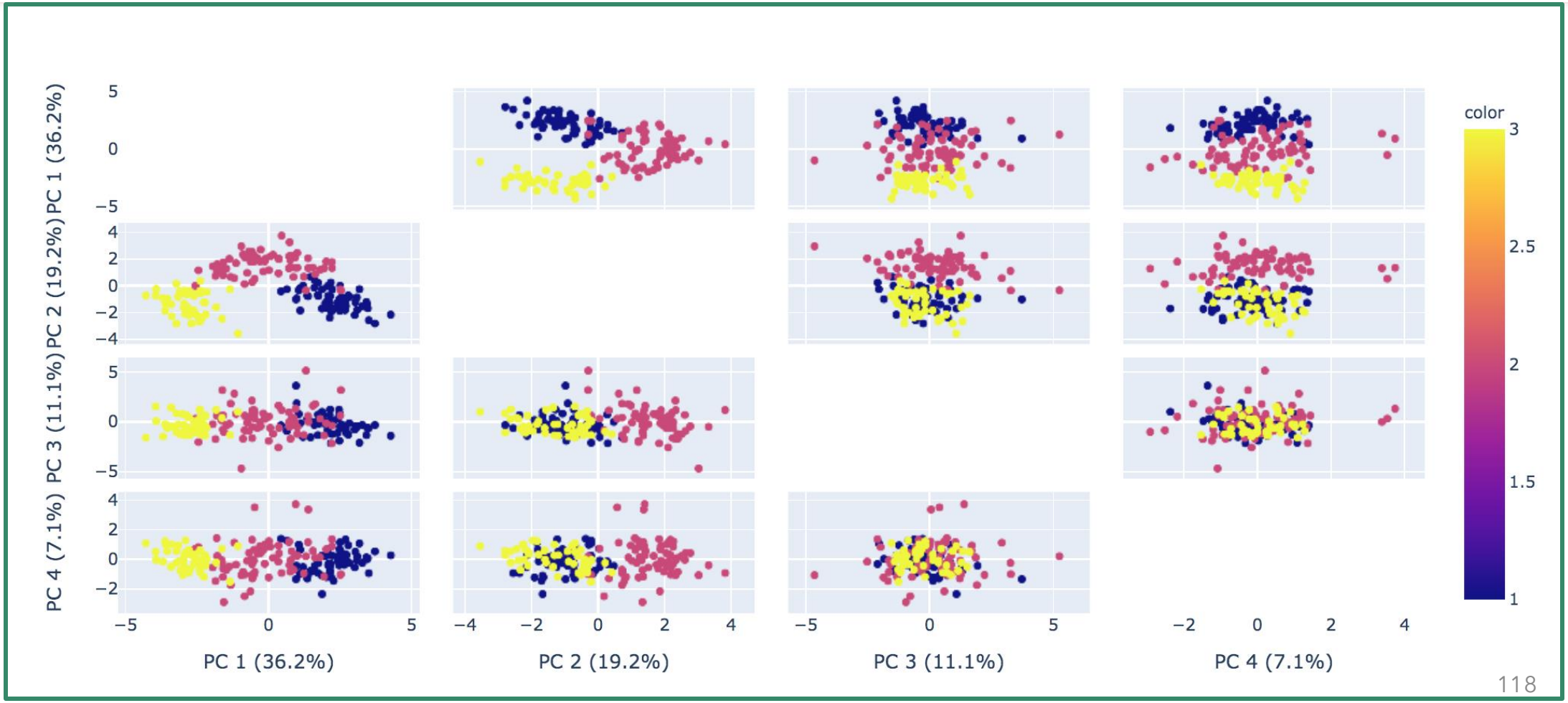
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
_X = scaler.fit_transform(X)
model = PCA()
X_pca = model.fit_transform(_X)

cols = list(df.columns[1:])
plt.bar(cols, model.explained_variance_ratio_)
plt.xlabel('PCA feature')
plt.ylabel('variance')
plt.xticks(cols)
plt.show()
```



Visualize 2D projection

```
import plotly.express as px

labels = {
    str(i): f"PC {i+1} ({var:.1f}%)"
    for i, var in enumerate(model.explained_variance_ratio_ * 100)
}

fig = px.scatter_matrix(
    X_pca,
    labels=labels,
    dimensions=range(4),
    color=df[0]
)
fig.update_traces(diagonal_visible=False)
fig.show()
```

Visualize 2D projection across pairs of PCs



Eigenfaces

If we treat images as vectors, a dataset of persons could reveal the "principal" faces.

Eigenfaces

sklearn.datasets.fetch_lfw_people

```
sklearn.datasets.fetch_lfw_people(*, data_home=None, funneled=True, resize=0.5, min_faces_per_person=0,
color=False, slice_=(slice(70, 195, None), slice(78, 172, None)), download_if_missing=True, return_X_y=False)
```
[source]

Load the Labeled Faces in the Wild (LFW) people dataset (classification).

Download it if necessary.

Classes	5749
Samples total	13233
Dimensionality	5828
Features	real, between 0 and 255

Data preparation

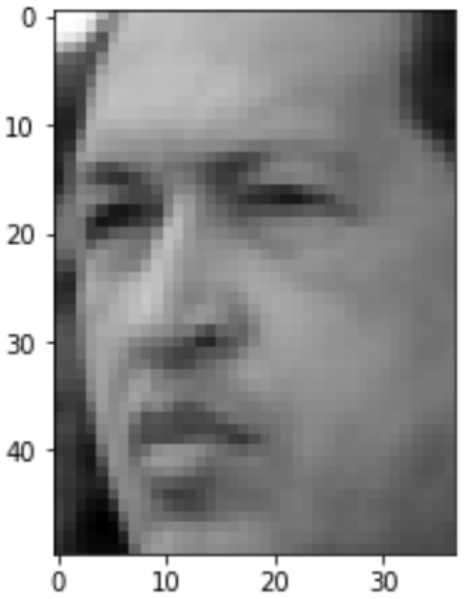
```
from sklearn.datasets import fetch_lfw_people

data = fetch_lfw_people(min_faces_per_person=70, resize=0.4)
print('dataset:', data.images.shape)
```

```
Downloading LFW metadata: https://ndownloader.figshare.com/files/5976012
Downloading LFW metadata: https://ndownloader.figshare.com/files/5976009
Downloading LFW metadata: https://ndownloader.figshare.com/files/5976006
Downloading LFW data (~200MB): https://ndownloader.figshare.com/files/5976015
dataset: (1288, 50, 37)
```

Data exploration

```
# example
plt.imshow(data.images[0], cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7f7b6c91e160>
```



Transform images into vectors

```
# flatten the image into vector
X = data.images.reshape(1288, -1)
```

Try PCA to get 5 principles face representations

```
from sklearn.decomposition import PCA
# find 5 faces
pca = PCA(n_components=5)
pca.fit(X)
```

```
PCA(copy=True, iterated_power='auto', n_components=5, random_state=None,
    svd_solver='auto', tol=0.0, whiten=False)
```

```
# these are the eigenvectors
eivec = pca.components_
print('eigenvec:', eivec.shape)
```

```
eigenvec: (5, 1850)
```

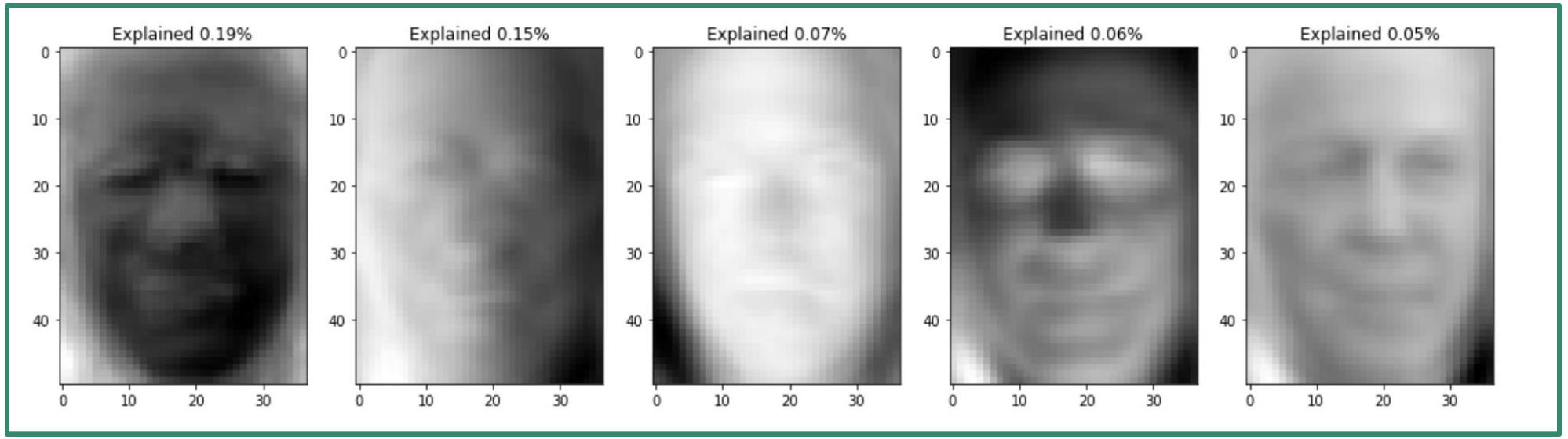
Explain the variances

```
print('each explain the variance of the dataset by:', pca.explained_variance_ratio_)
```

```
each explain the variance of the dataset by: [0.18831733 0.15066975 0.07268997 0.0603476  0.05140407]
```

Visualize the variances

```
fig, ax = plt.subplots(ncols=len(eivec), figsize=(3.7*(len(eivec)), 5))
for i in range(len(eivec)):
  v = eivec[i]
  ratio = pca.explained_variance_ratio_[i]
  ax[i].set_title(f'Explained {ratio:.2f}%')
  ax[i].imshow(v.reshape(50, 37), cmap='gray')
```




t-SNE
t-distributed stochastic neighbor embedding

t-SNE

- It is a dimension reduction for visualization.
- t-SNE takes a set of points in high-dimensional space and try to find the representation of these points in the lower dimensional space, typically 2D.
- The algorithm is non-linear projection.

Source: https://distill.pub/2016/misread-tsne/

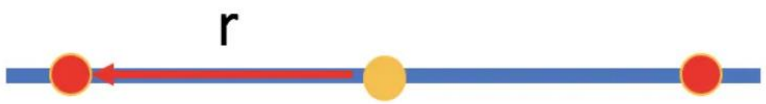
Wattenberg, et al., "How to Use t-SNE Effectively", Distill, 2016. http://doi.org/10.23915/distill.00002

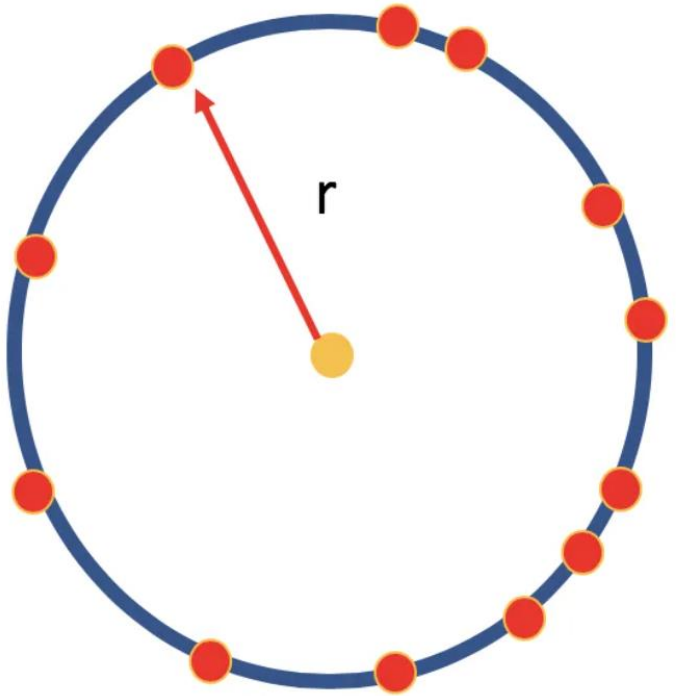
t-SNE

- It creates Gaussian distribution to represent the relationship between points in high-dimensional space.
- It recreates the student t-distribution in low dimensional space. This avoids the crowding problem.

https://mlexplained.com/2018/09/14/paper-dissected-visualizing-data-using-t-sne-explained/

t-SNE tries to optimize the spread-out the medium distance points





More points are having the same medium distance from a point in the high dimensional space

https://mlexplained.com/2018/09/14/paper-dissected-visualizing-data-using-t-sne-explained/

Stochastic neighbors

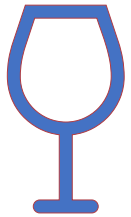
- No clear line of which points are neighbors of the other points
- This allows t-SNE to take both global and local structure into account.
- t-SNE defines a probability to pick another point as a neighbor, which is proportionate to Gaussian distribution.
- Gradient descent is used to optimize the t-distribution in the lower dimensional space.

Perplexity

- Perplexity is a hyperparameter.
- It is the effective number of neighbors of any point. It tends to be small for points in densely populated areas and larger for sparse areas.
- Larger perplexities will take more global structure whereas smaller perplexities will make the embeddings more locally focused.

t-SNE for wine data dimension reduction

https://colab.research.google.com/drive/1hPukLQt5N9ymsaqFut1Rq6ULHWxBtJRw



class 1 59
class 2 71
class 3 48

Wine dataset

The attributes are (dontated by Riccardo Leardi, riclea@anchem.unige.it)

1) Alcohol
2) Malic acid
3) Ash
4) Alcalinity of ash
5) Magnesium
6) Total phenols
7) Flavanoids
8) Nonflavanoid phenols
9) Proanthocyanins
10)Color intensity
11)Hue
12)OD280/OD315 of diluted wines
13)Proline

Data loading

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.cluster import KMeans

# Wine data
# 178 samples with 3 distinct varieties of red wine: Barolo, Grignolino and Barbera
# Features measure chemical composition e.g. alcohol content
# https://archive.ics.uci.edu/ml/datasets/Wine
url = 'https://raw.githubusercontent.com/wichadak/wine/master/wine.data'
df = pd.read_csv(url, header=None)
df.sample(frac=1).head()
```

Wine data

Features or dimensions

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
85	2	12.67	0.98	2.24	18.0	99	2.20	1.94	0.30	1.46	2.62	1.23	3.16	450
66	2	13.11	1.01	1.70	15.0	78	2.98	3.18	0.26	2.28	5.30	1.12	3.18	502
21	1	12.93	3.80	2.65	18.6	102	2.41	2.41	0.25	1.98	4.50	1.03	3.52	770
83	2	13.05	3.86	2.32	22.5	85	1.65	1.59	0.61	1.62	4.80	0.84	2.01	515
97	2	12.29	1.41	1.98	16.0	85	2.55	2.50	0.29	1.77	2.90	1.23	2.74	428

samples

Wine type

Data exploration

```
df.describe()
```

	0	1	2	3	4	5	6	7	8
count	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000
mean	1.938202	13.000618	2.336348	2.366517	19.494944	99.741573	2.295112	2.029270	0.361854
std	0.775035	0.811827	1.117146	0.274344	3.339564	14.282484	0.625851	0.998859	0.124453
min	1.000000	11.030000	0.740000	1.360000	10.600000	70.000000	0.980000	0.340000	0.130000
25%	1.000000	12.362500	1.602500	2.210000	17.200000	88.000000	1.742500	1.205000	0.270000
50%	2.000000	13.050000	1.865000	2.360000	19.500000	98.000000	2.355000	2.135000	0.340000
75%	3.000000	13.677500	3.082500	2.557500	21.500000	107.000000	2.800000	2.875000	0.437500
max	3.000000	14.830000	5.800000	3.230000	30.000000	162.000000	3.880000	5.080000	0.660000

Data preparation

```
X = df[df.columns[1:]].to_numpy(dtype=np.float32)
y = df[df.columns[0]].to_numpy()
print('X:', X.shape, X.dtype)
print('y:', y.shape, y.dtype)
```

```
X: (178, 13) float32
y: (178,) int64
```

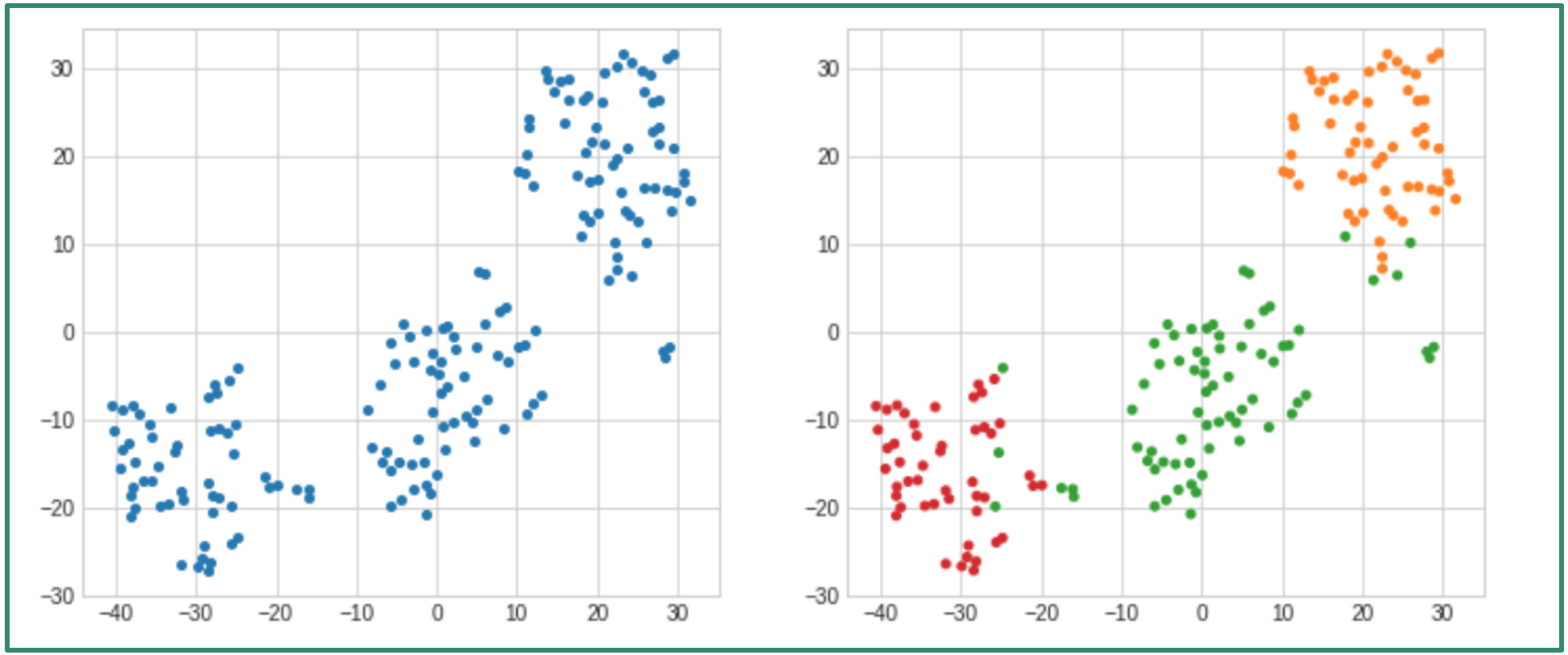
Define functions for visualization

```
plt.style.use('seaborn-whitegrid')
colors = np.array(plt.rcParams['axes.prop_cycle'].by_key()['color'])

def scatter(X, y):
  # scatter plot
  plt.scatter(X[:,0], X[:,1], c=colors[y], s=4)

def scatter_2sides(X, y):
  # visualize the cluster and color it with ground truth
  fig, ax = plt.subplots(ncols=2, figsize=(12, 5))
  ax[0].scatter(_X[:,0], _X[:,1], s=16)
  ax[1].scatter(_X[:,0], _X[:,1], c=colors[y], s=16)
```

Try t-SNE for dimension reduction

```
from sklearn.manifold import TSNE
from sklearn.preprocessing import StandardScaler

# standardize the data first
scaler = StandardScaler()
_X = scaler.fit_transform(X)

model = TSNE(n_components=2,
             perplexity=10,
             learning_rate=100,
             random_state=42)
_X = model.fit_transform(_X)

# visualize the cluster and color it with ground truth
scatter_2sides(_X, y)
```

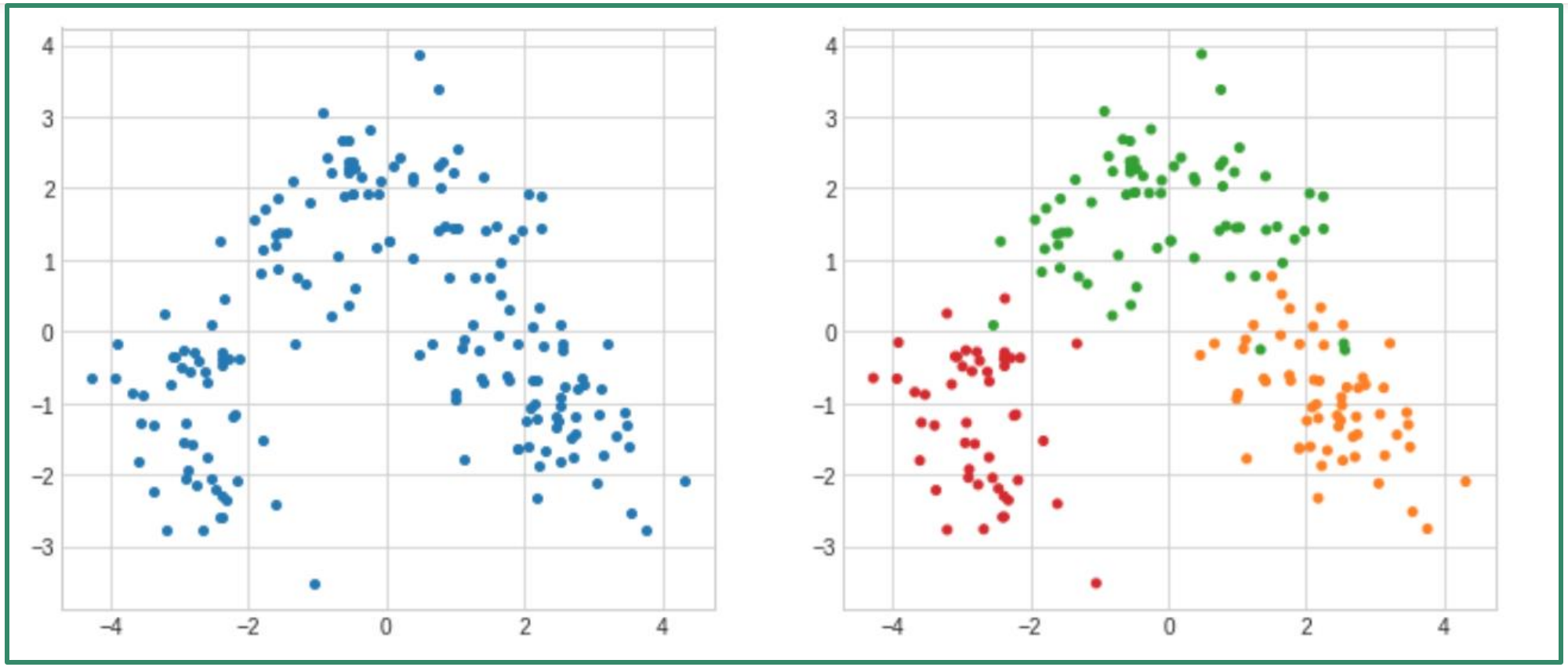
Visualized t-SNE results



Compare to PCA

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
_X = scaler.fit_transform(X)
model = PCA(n_components=2)
_X = model.fit_transform(_X)

# visualize the cluster and color it with ground truth
scatter_2sides(_X, y)
```

Visualize PCA results

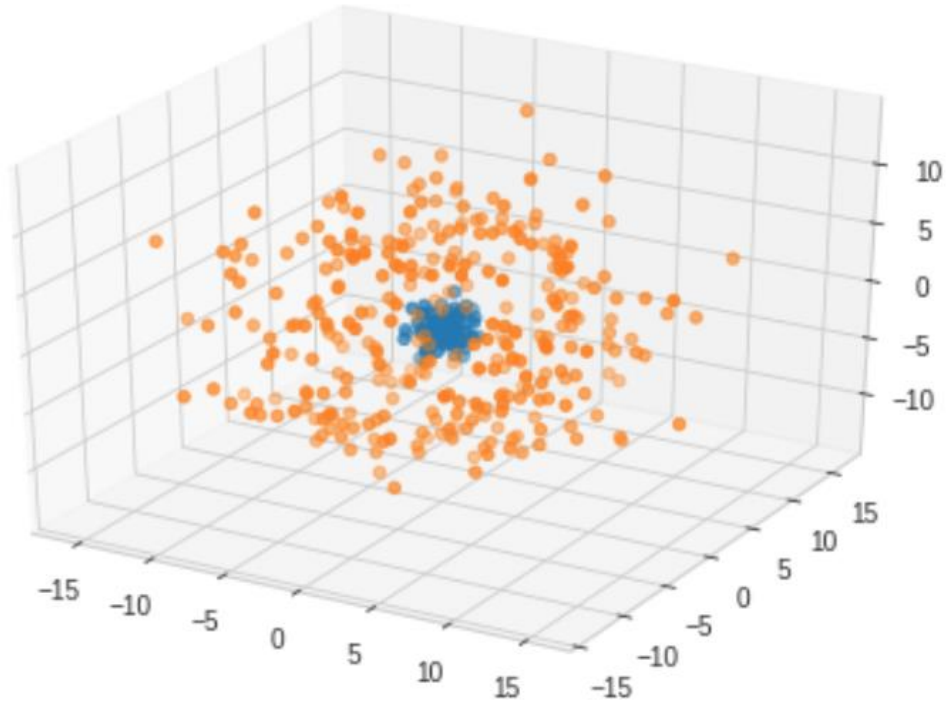


Dataset#2: randomly generated points

```
# Another example for compring PCA and t-SNE
# Source https://mlexplained.com/2018/09/14/paper-dissected-visualizing-data-using-t-sne-explained/
# Generate a synthetic data

import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline

A = np.random.normal(scale=1, size=(100,3))
B = np.array([x for x in np.random.normal(scale=5, size=(500,3)) if np.linalg.norm(x) > 7])

from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure()
ax = Axes3D(fig)
ax.scatter(A[:,0], A[:,1], A[:, 2])
ax.scatter(B[:,0], B[:,1], B[:, 2])
```

Visualization of the randomly generated points

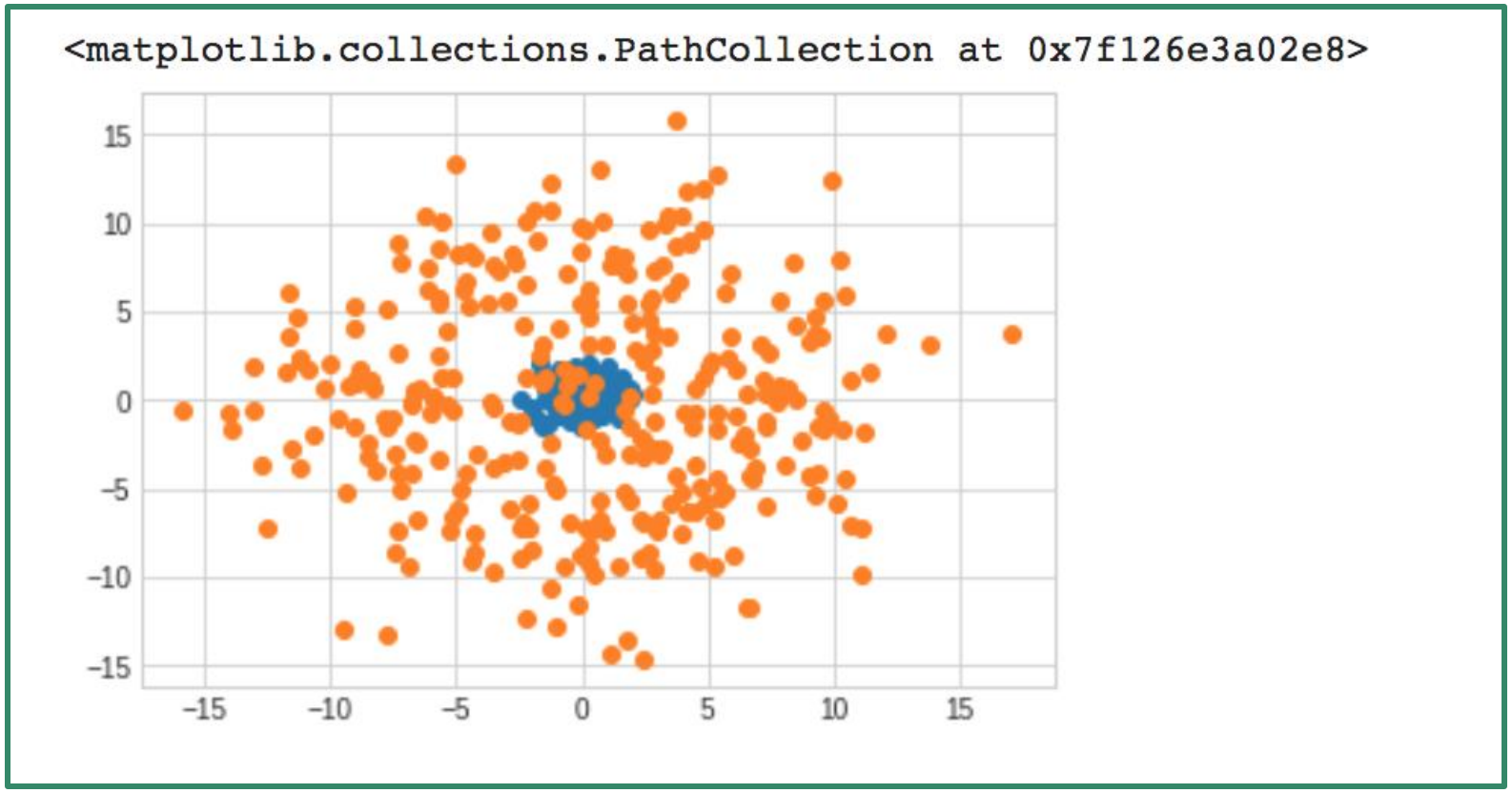
```
<mpl_toolkits.mplot3d.art3d.Path3DCollection at 0x7f1261d85a20>
```



Apply PCA for these random points

```
from sklearn.decomposition import PCA
X = np.r_[A,B]
X2 = PCA(n_components=2).fit_transform(X)
A2 = X2[:A.shape[0],:]
B2 = X2[A.shape[0]:,:]
plt.scatter(A2[:,0], A2[:,1])
plt.scatter(B2[:,0], B2[:,1])
```

PCA result for the random points



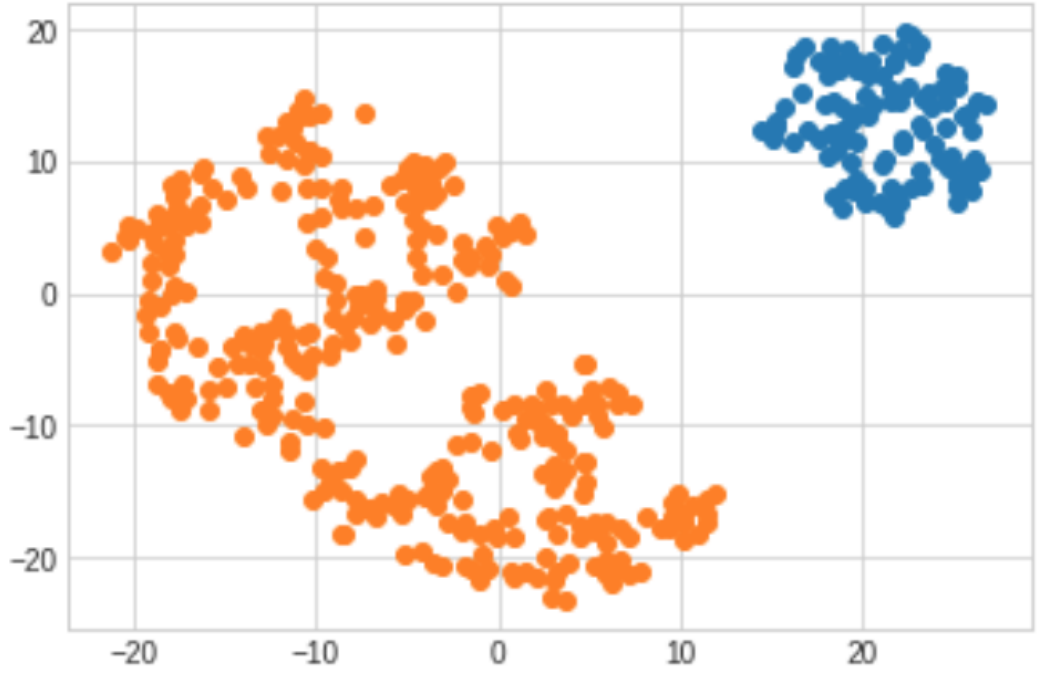
Apply t-SNE for these random points

```
from sklearn.manifold import TSNE
X3 = TSNE(n_components=2).fit_transform(X)
A3 = X3[:A.shape[0],:]
B3 = X3[A.shape[0]:,:]
plt.scatter(A3[:,0], A3[:,1])
plt.scatter(B3[:,0], B3[:,1])
```

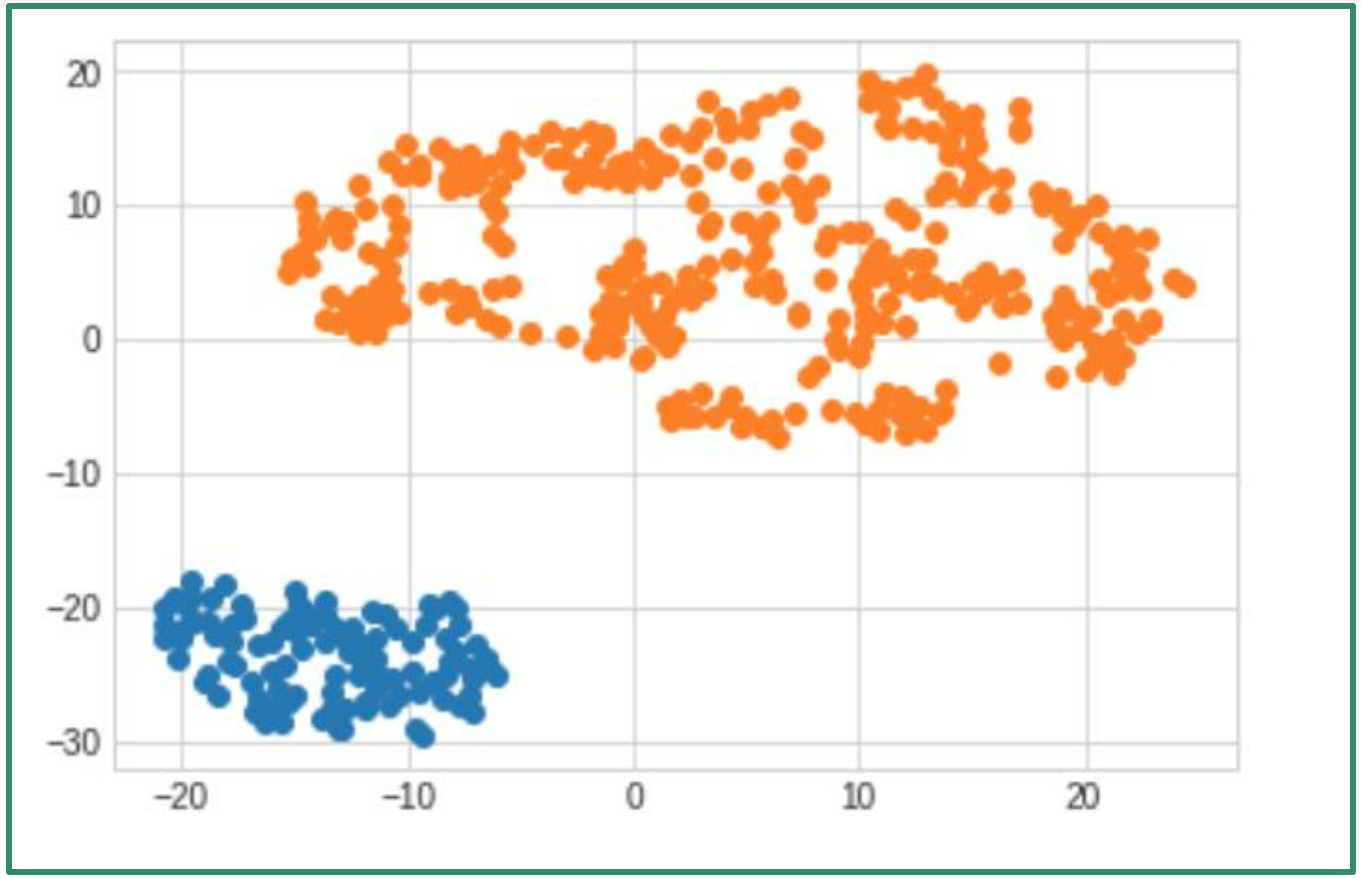
If we run this code for multiple times, what will be the results??

t-SNE result for the random points

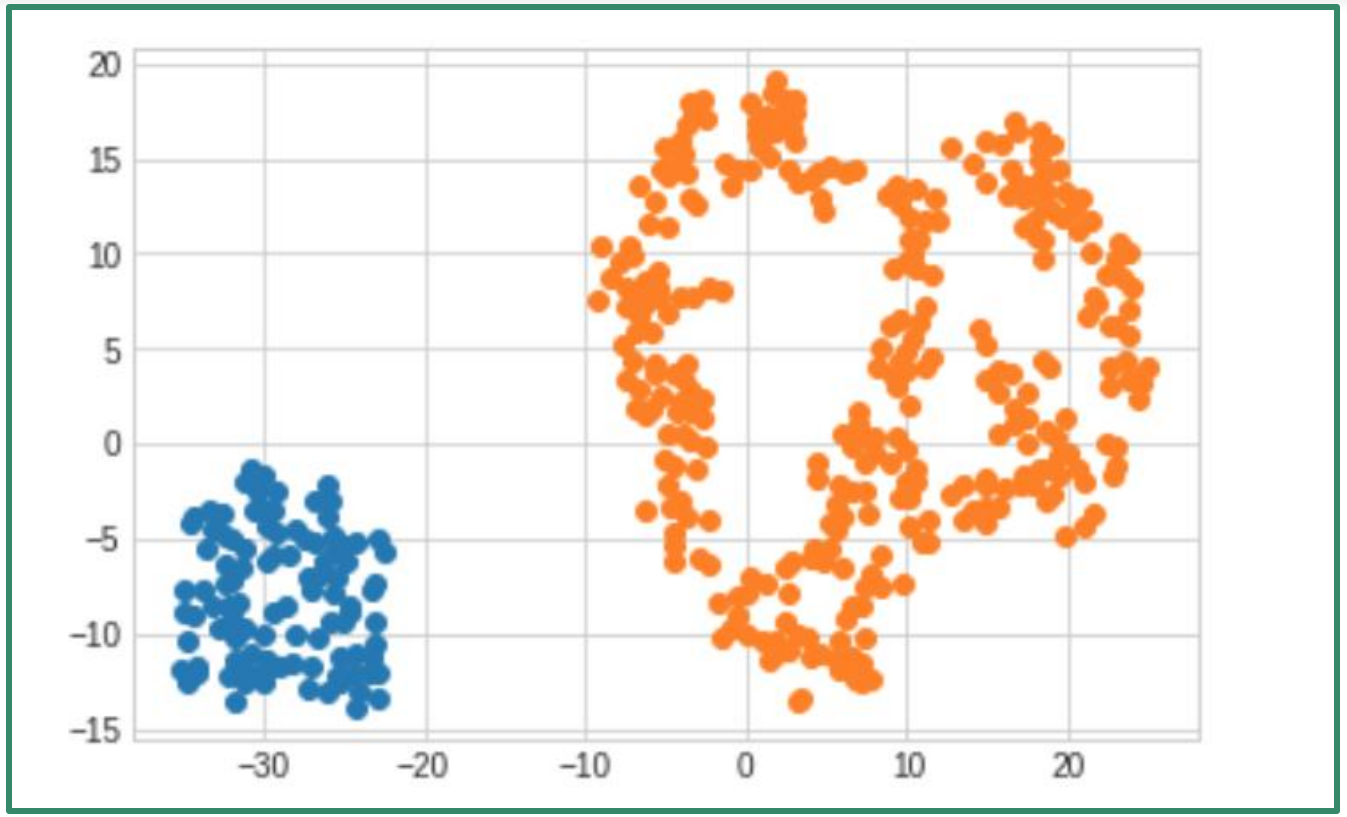
```
<matplotlib.collections.PathCollection at 0x7f127f114cc0>
```



t-SNE result for the random points



t-SNE result for the random points

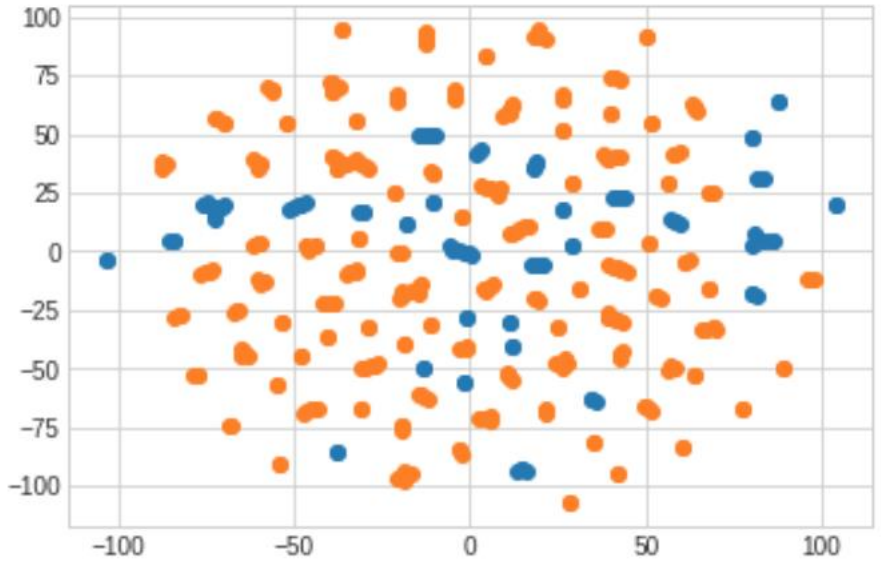


How about using different perplexity

```
# Try different perplexity
from sklearn.manifold import TSNE
X3 = TSNE(n_components=2, perplexity=1).fit_transform(X)
A3 = X3[:A.shape[0],:]
B3 = X3[A.shape[0]:,:]
plt.scatter(A3[:,0], A3[:,1])
plt.scatter(B3[:,0], B3[:,1])
```

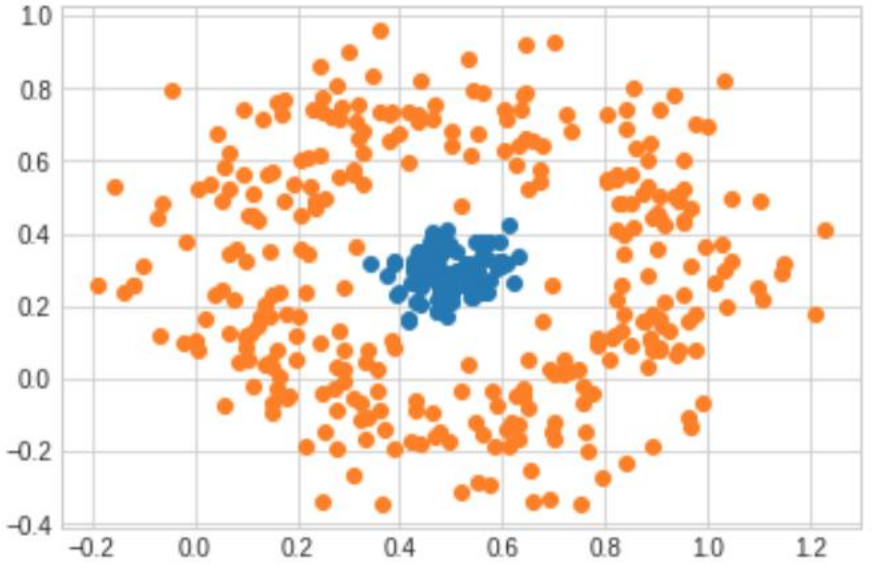
Different perplexity results

<matplotlib.collections.PathCollection at 0x7f1



Perplexity = 1

<matplotlib.collections.PathCollection at 0x7f126e891828>



Perplexity = 400

MNIST dataset

Modified National Institute of Standards and Technology database

https://commons.wikimedia.org/wiki/File:MnistExamples.png

mnist_784

active ARFF Publicly available Visibility: public Uploaded 29-09-2014 by Joaquin Vanschoren

4 likes downloaded by 68 people , 98 total downloads 0 issues 0 downvotes

AzurePilot OpenML-CC18 OpenML100 study_1 study_123 study_41 study_99 vision study_14 study_225 + Add tag

Help us complete this description → Edit

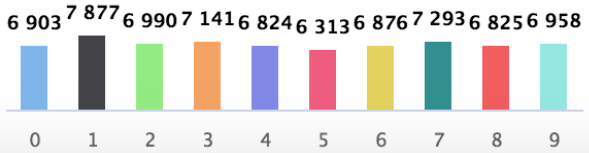
Author: Yann LeCun, Corinna Cortes, Christopher J.C. Burges
Source: MNIST Website - Date unknown
Please cite:

The MNIST database of handwritten digits with 784 features, raw data available at: http://yann.lecun.com/exdb/mnist/. It can be split in a training set of the first 60,000 examples, and a test set of 10,000 examples

It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image. It is a

▾ Show all

785 features

class **(target)**	nominal	10 unique values 0 missing	

https://www.openml.org/d/554

Data loading

```
import time
import matplotlib.pyplot as plt
import numpy as np

from sklearn.datasets import fetch_openml
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.utils import check_random_state
```

```
X, y = fetch_openml('mnist_784', version=1, return_X_y=True)
```

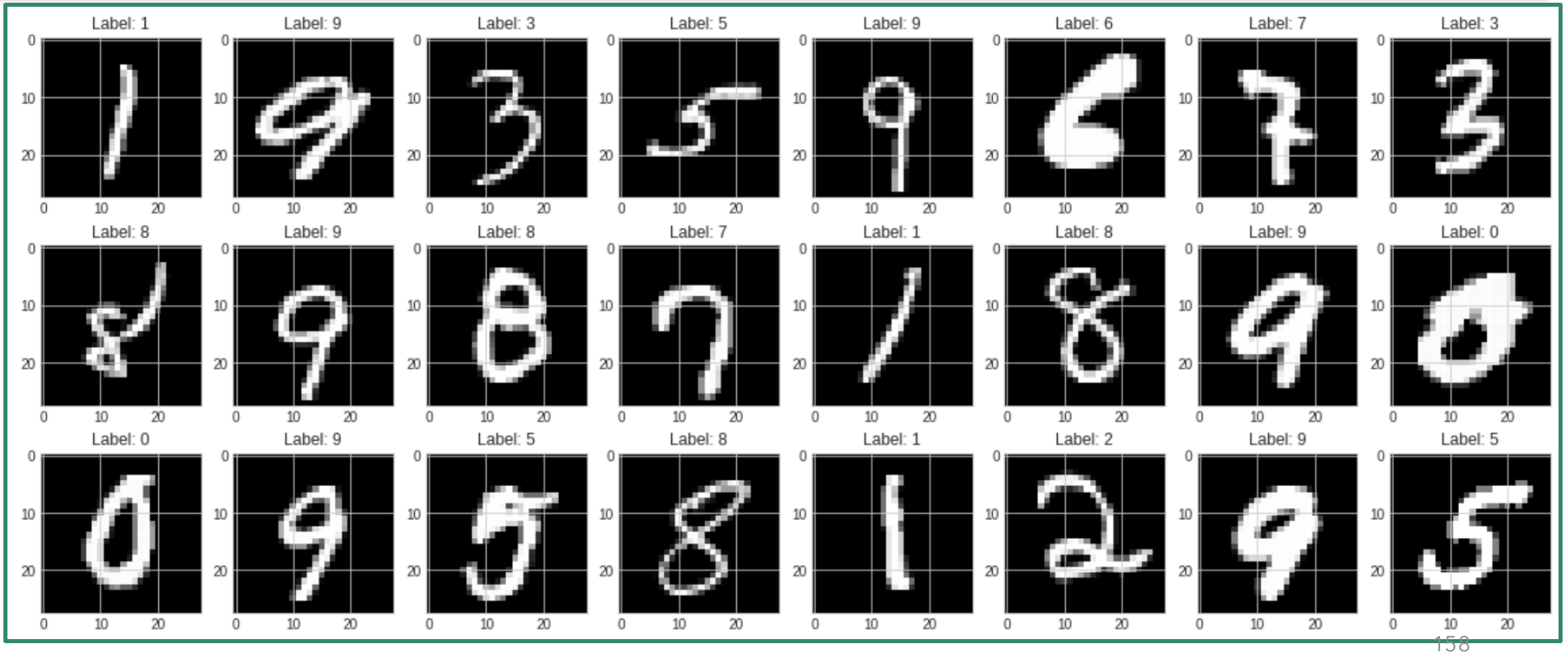
```
print('X:', X.shape)
print('y:', y.shape)
```

```
X: (70000, 784)
y: (70000,)
```

Data exploration

```
idx = np.random.choice(len(X), 24)
fig, ax = plt.subplots(3, 8, figsize=(20, 8))
ax = [*ax[0], *ax[1], *ax[2]]
for i in range(len(idx)):
  img = X[idx[i]].reshape(28, 28)
  ax[i].set_title(f'Label: {y[idx[i]]}')
  ax[i].imshow(img, cmap='gray')
```

Data exploration



Sampling only some data

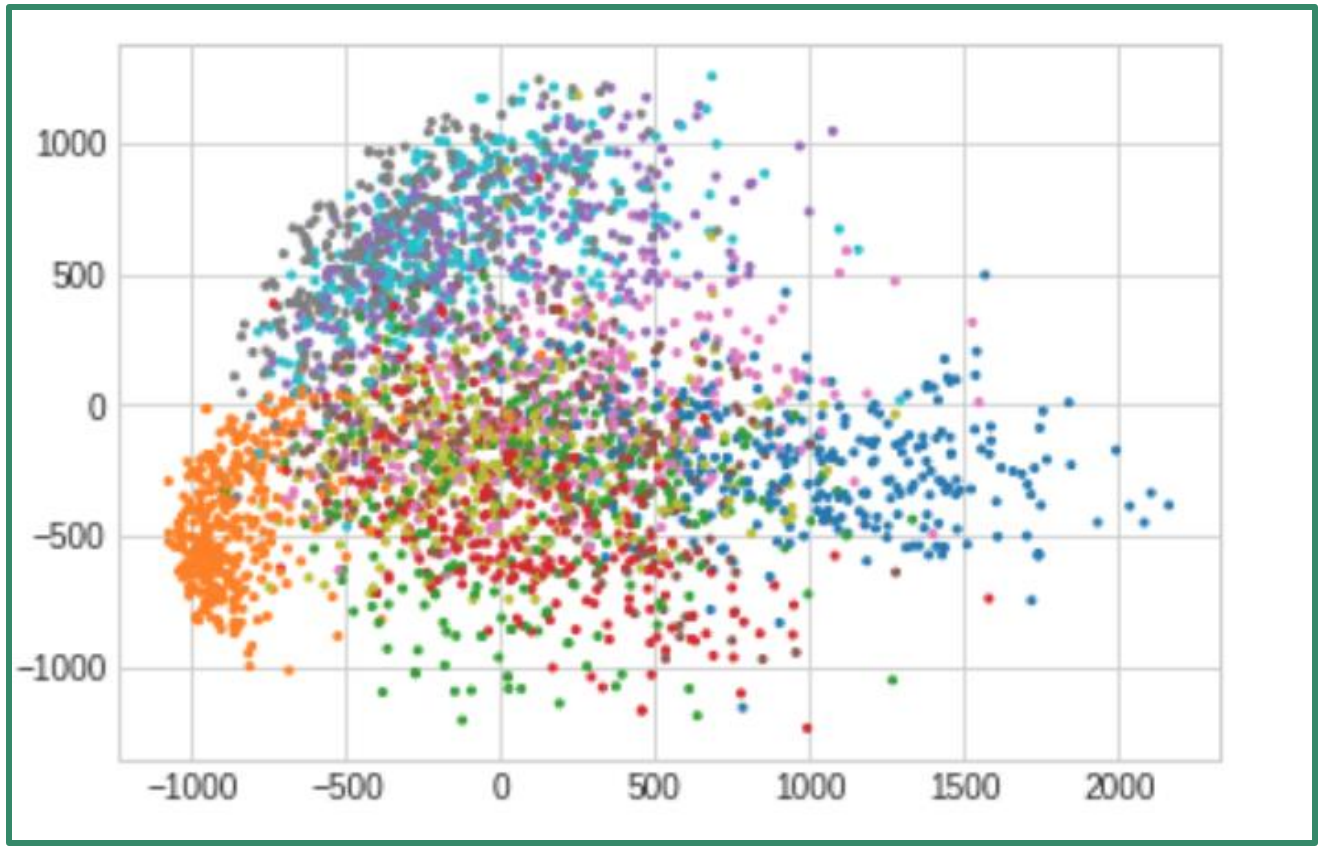
```
n = 3000
np.random.seed(42)
idx = np.random.choice(len(X), n)
Xs = X[idx]
ys = np.array([int(i) for i in y[idx]])
print('X:', Xs.shape)
print('Y:', ys.shape)
print(ys[:3])
```

```
X: (3000, 784)
Y: (3000,)
[0 0 1]
```

Natively using PCA

```
from sklearn.decomposition import PCA

# Create PCA instance: model
model = PCA(n_components=2)
# Apply the fit_transform method of model to grains: pca_features
_X = model.fit_transform(Xs)

scatter(_X, ys)
```

PCA result of MNIST



How about t-SNE for MNIST (assignment)

References

- StatQuest: Principal Component Analysis (PCA), Step-by-Step (https://www.youtube.com/watch?v=FgakZw6K1QQ)
- StatQuest: t-SNE clearly explained https://www.youtube.com/watch?v=NEaUSP4YerM
- https://towardsdatascience.com/an-approach-to-choosing-the-number-of-components-in-a-principal-component-analysis-pca-3b9f3d6e73fe
- https://www.datacamp.com/community/tutorials/introduction-t-sne
- https://towardsdatascience.com/k-means-clustering-from-a-to-z-f6242a314e9a
- https://stanford.edu/~cpiech/cs221/handouts/kmeans.html
- http://www.mit.edu/~9.54/fall14/slides/Class13.pdf