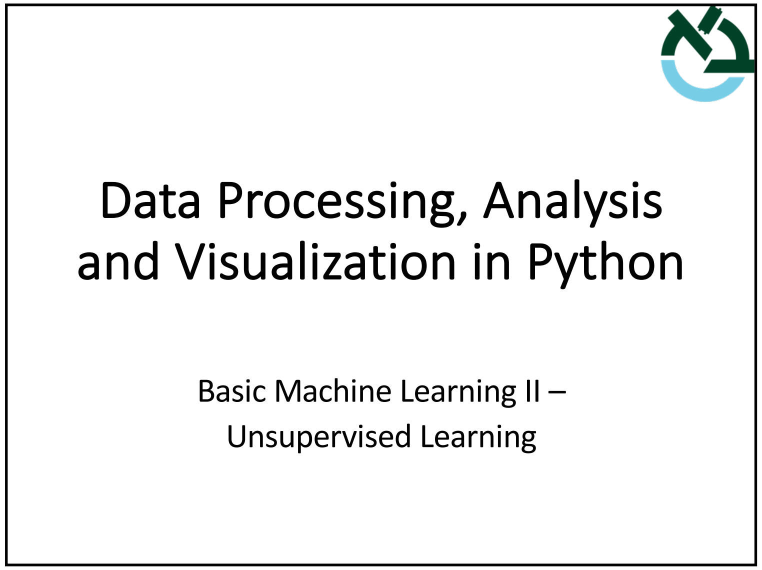
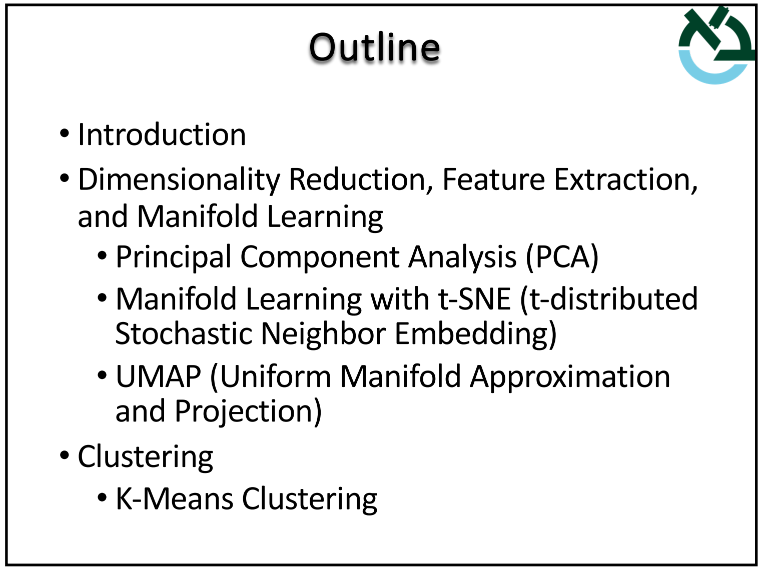


Data Processing, Analysis and Visualization in Python

Basic Machine Learning II –
Unsupervised Learning

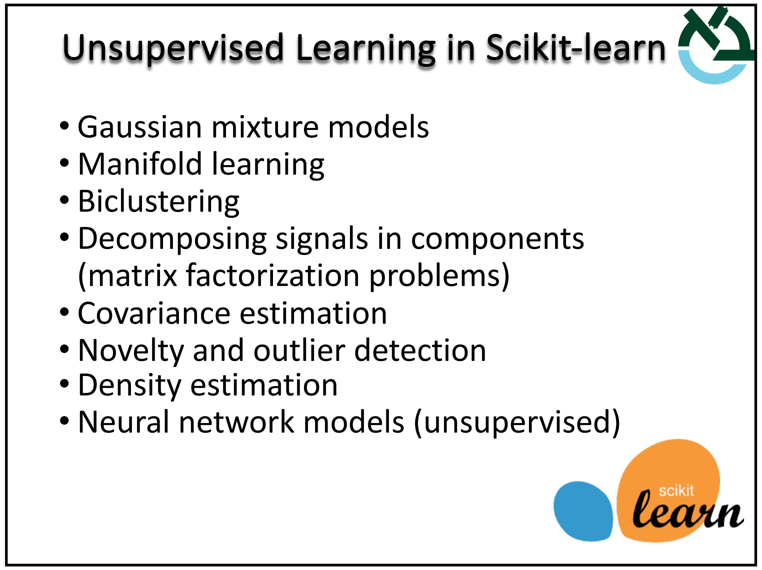
1



Outline

- Introduction
- Dimensionality Reduction, Feature Extraction, and Manifold Learning
  - Principal Component Analysis (PCA)
  - Manifold Learning with t-SNE (t-distributed Stochastic Neighbor Embedding)
  - UMAP (Uniform Manifold Approximation and Projection)
- Clustering
  - K-Means Clustering

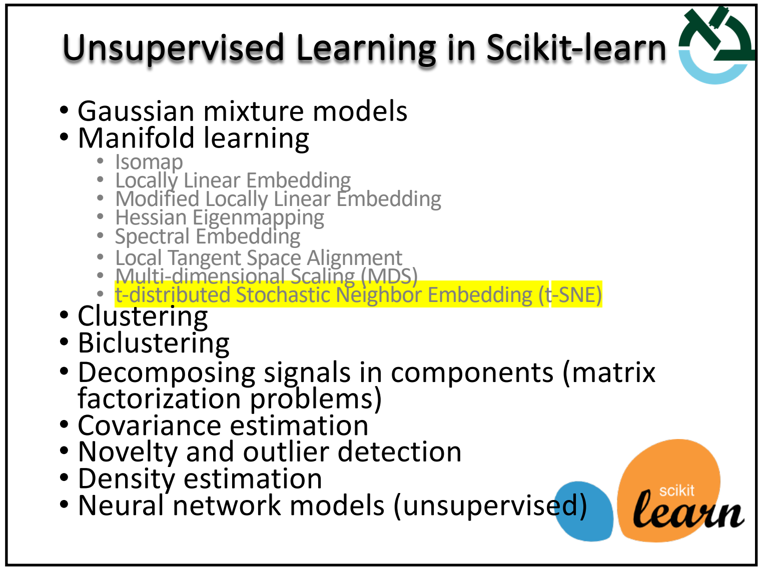
2



Unsupervised Learning in Scikit-learn

- Gaussian mixture models
- Manifold learning
- Biclustering
- Decomposing signals in components (matrix factorization problems)
- Covariance estimation
- Novelty and outlier detection
- Density estimation
- Neural network models (unsupervised)

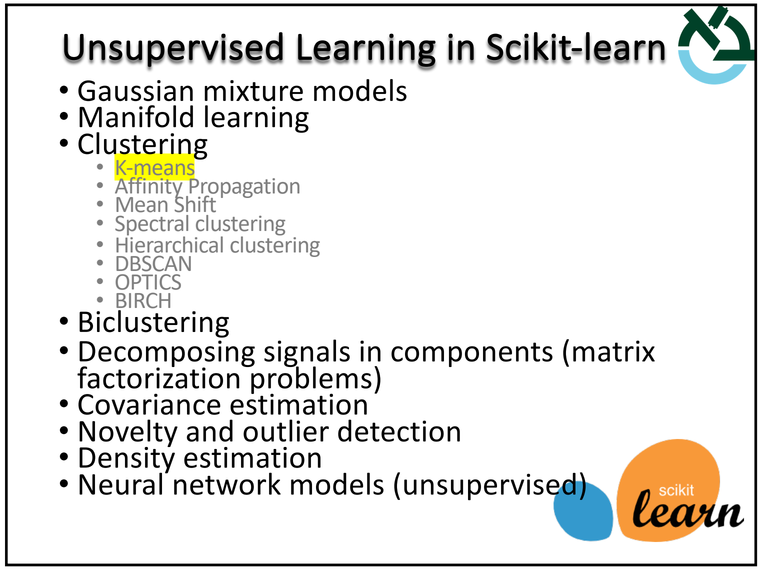
3



Unsupervised Learning in Scikit-learn

- Gaussian mixture models
- Manifold learning
  - Isomap
  - Locally Linear Embedding
  - Modified Locally Linear Embedding
  - Hessian Eigenmapping
  - Spectral Embedding
  - Local Tangent Space Alignment
  - Multi-dimensional Scaling (MDS)
  - t-distributed Stochastic Neighbor Embedding (t-SNE)
- Clustering
- Biclustering
- Decomposing signals in components (matrix factorization problems)
- Covariance estimation
- Novelty and outlier detection
- Density estimation
- Neural network models (unsupervised)

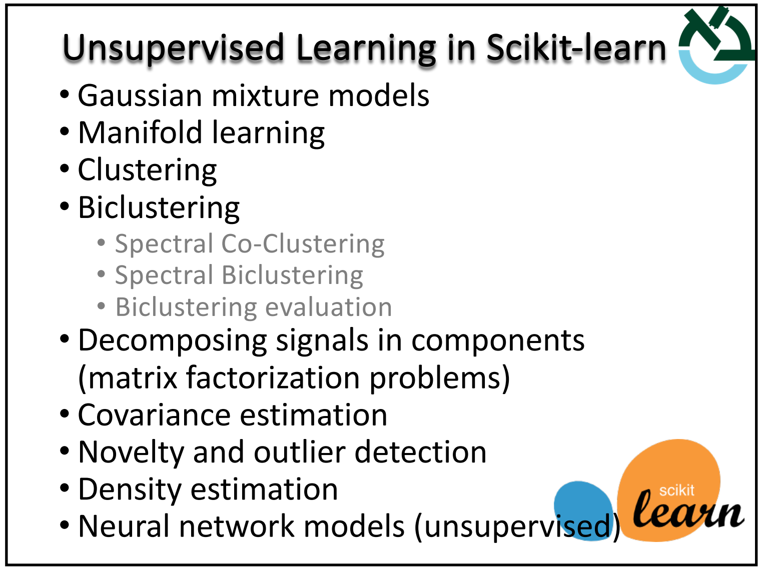
4



Unsupervised Learning in Scikit-learn

- Gaussian mixture models
- Manifold learning
- Clustering
  - K-means
  - Affinity Propagation
  - Mean Shift
  - Spectral clustering
  - Hierarchical clustering
  - DBSCAN
  - OPTICS
  - BIRCH
- Biclustering
- Decomposing signals in components (matrix factorization problems)
- Covariance estimation
- Novelty and outlier detection
- Density estimation
- Neural network models (unsupervised)

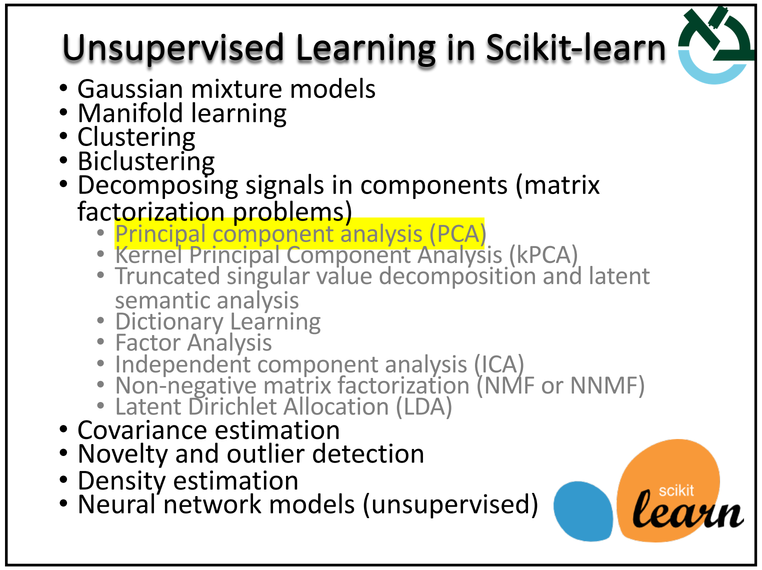
5



Unsupervised Learning in Scikit-learn

- Gaussian mixture models
- Manifold learning
- Clustering
- Biclustering
  - Spectral Co-Clustering
  - Spectral Biclustering
  - Biclustering evaluation
- Decomposing signals in components (matrix factorization problems)
- Covariance estimation
- Novelty and outlier detection
- Density estimation
- Neural network models (unsupervised)

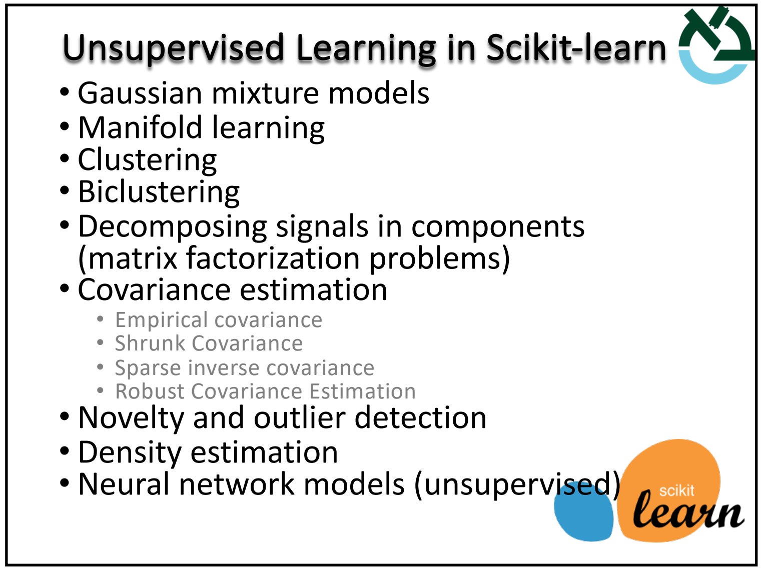
6



Unsupervised Learning in Scikit-learn

- Gaussian mixture models
- Manifold learning
- Clustering
- Biclustering
- Decomposing signals in components (matrix factorization problems)
  - Principal component analysis (PCA)
  - Kernel Principal Component Analysis (kPCA)
  - Truncated singular value decomposition and latent semantic analysis
  - Dictionary Learning
  - Factor Analysis
  - Independent component analysis (ICA)
  - Non-negative matrix factorization (NMF or NNMF)
  - Latent Dirichlet Allocation (LDA)
- Covariance estimation
- Novelty and outlier detection
- Density estimation
- Neural network models (unsupervised)

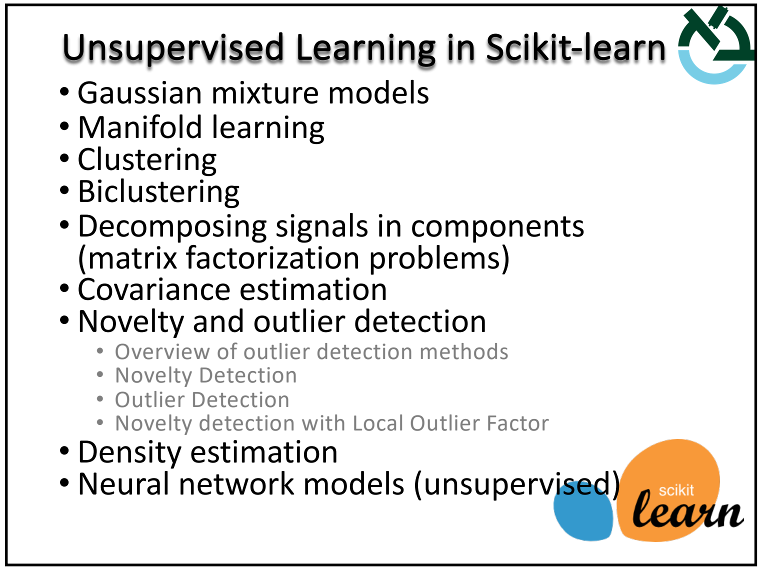
7



Unsupervised Learning in Scikit-learn

- Gaussian mixture models
- Manifold learning
- Clustering
- Biclustering
- Decomposing signals in components (matrix factorization problems)
- Covariance estimation
  - Empirical covariance
  - Shrunk Covariance
  - Sparse inverse covariance
  - Robust Covariance Estimation
- Novelty and outlier detection
- Density estimation
- Neural network models (unsupervised)

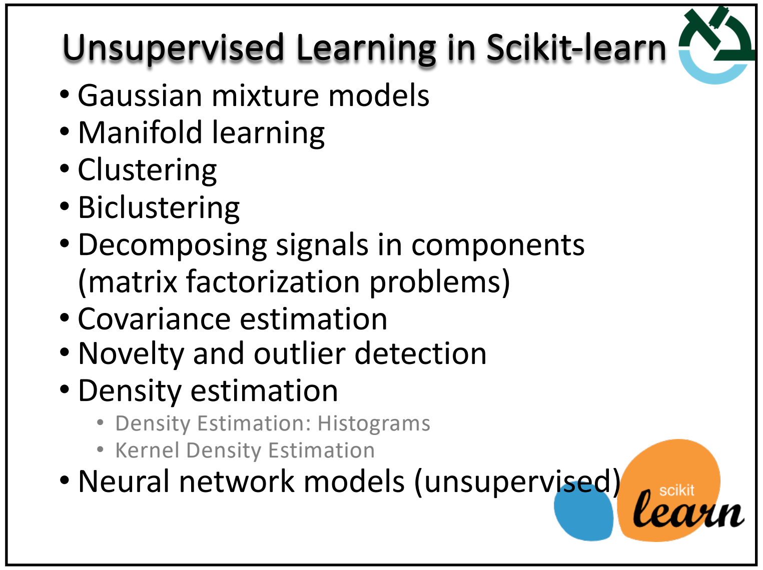
8



Unsupervised Learning in Scikit-learn

- Gaussian mixture models
- Manifold learning
- Clustering
- Biclustering
- Decomposing signals in components (matrix factorization problems)
- Covariance estimation
- Novelty and outlier detection
  - Overview of outlier detection methods
  - Novelty Detection
  - Outlier Detection
  - Novelty detection with Local Outlier Factor
- Density estimation
- Neural network models (unsupervised)

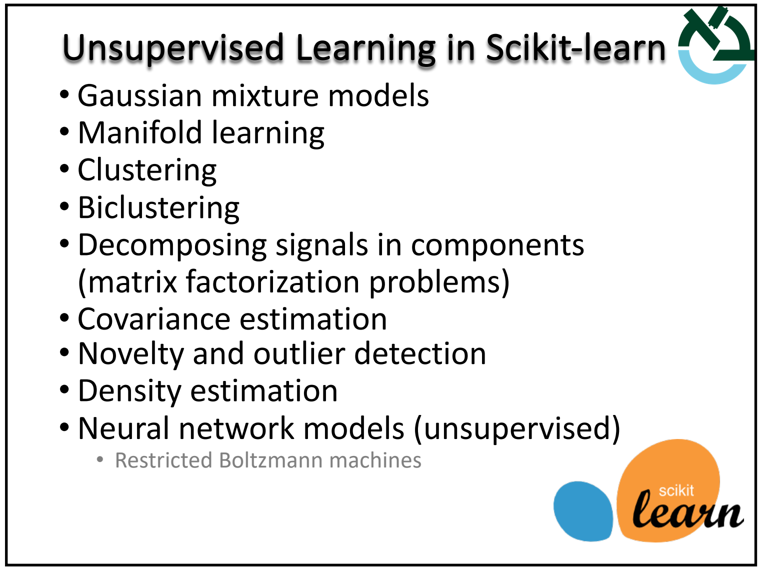
9



Unsupervised Learning in Scikit-learn

- Gaussian mixture models
- Manifold learning
- Clustering
- Biclustering
- Decomposing signals in components (matrix factorization problems)
- Covariance estimation
- Novelty and outlier detection
- Density estimation
  - Density Estimation: Histograms
  - Kernel Density Estimation
- Neural network models (unsupervised)

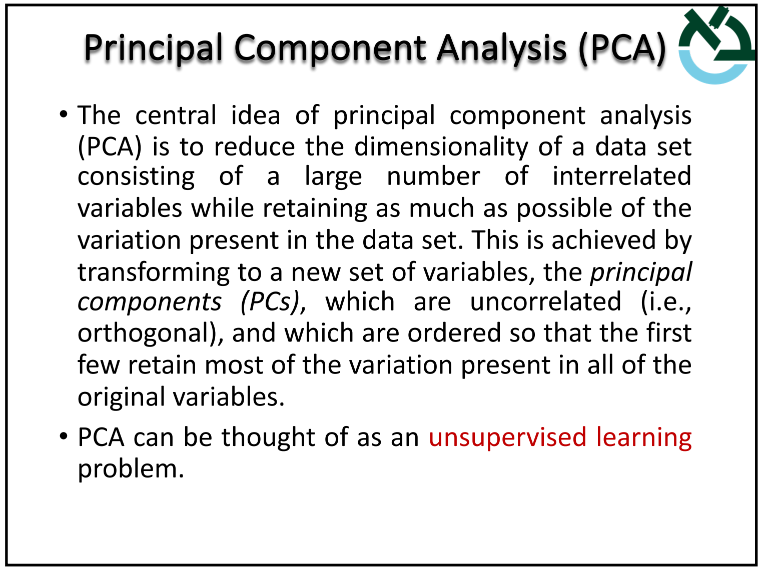
10



Unsupervised Learning in Scikit-learn

- Gaussian mixture models
- Manifold learning
- Clustering
- Biclustering
- Decomposing signals in components (matrix factorization problems)
- Covariance estimation
- Novelty and outlier detection
- Density estimation
- Neural network models (unsupervised)
  - Restricted Boltzmann machines

11



Principal Component Analysis (PCA)

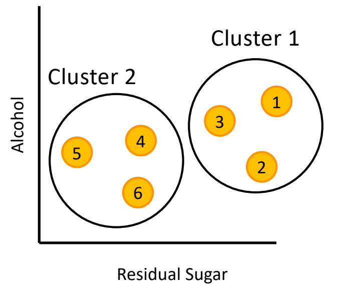
- The central idea of principal component analysis (PCA) is to reduce the dimensionality of a data set consisting of a large number of interrelated variables while retaining as much as possible of the variation present in the data set. This is achieved by transforming to a new set of variables, the *principal components (PCs)*, which are uncorrelated (i.e., orthogonal), and which are ordered so that the first few retain most of the variation present in all of the original variables.
- PCA can be thought of as an unsupervised learning problem.

12

PCA in a nutshell

- Let's assume we have 6 datapoints and 2 variables

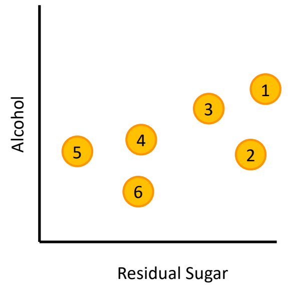
	residual sugar	alcohol
1	2.6	9.8
2	2.3	9.8
3	1.9	9.8
4	1.9	9.4
5	1.8	9.4
6	1.6	9.4



13

PCA in a nutshell

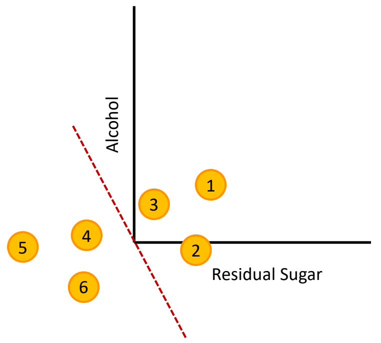
1. Calculate the mean of each variable
2. Calculate the center of the data for each variable and shift to origin



14

PCA in a nutshell

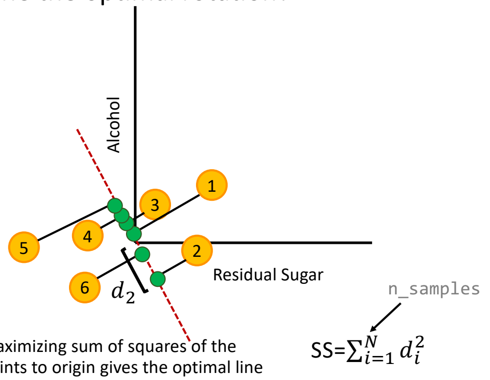
1. Draw a line through the origin and then rotate until it fits the points optimally



15

PCA in a nutshell

1. Draw a line through the origin and then rotate until it fits the points optimally. How do we determine the optimal rotation?



Maximizing sum of squares of the points to origin gives the optimal line

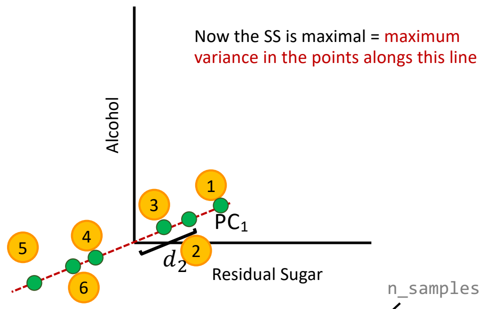
$SS = \sum_{i=1}^{N} d_i^2$

16

PCA in a nutshell

Draw a line through the origin and then rotate until it fits the points optimally. This is PC1.

Now the SS is maximal = maximum variance in the points alongs this line



The value of the sum of squares is called the **eigenvalue** $SS = \sum_{i=1}^{N} d_i^2$

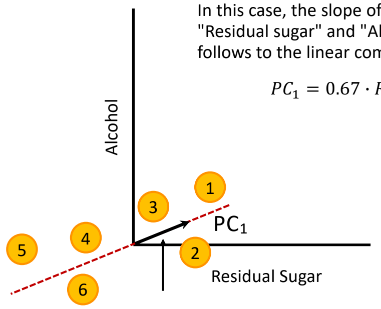
17

PCA in a nutshell

1. Draw a line through the origin and then rotate until it fits the points optimally. This is PC_1.

In this case, the slope of the line is ca. 0.5, so "Residual sugar" and "Alcohol" contribute as follows to the linear combination of PC_1:

$$PC_1 = 0.67 \cdot RS + 0.33 \cdot A$$



Unit vector called **eigenvector**

18

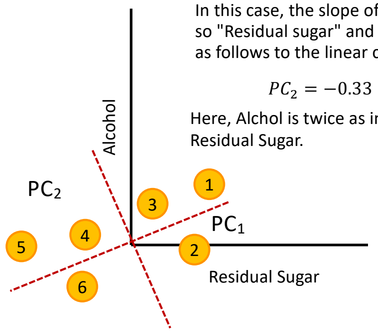
PCA in a nutshell

The line orthogonal to PC_1 through the origin is PC_2.

In this case, the slope of the new line is ca. -2, so "Residual sugar" and "Alcohol" contribute as follows to the linear combination of PC2:

$$PC_2 = -0.33 \cdot RS + 0.67 \cdot A$$

Here, Alchol is twice as important as Residual Sugar.



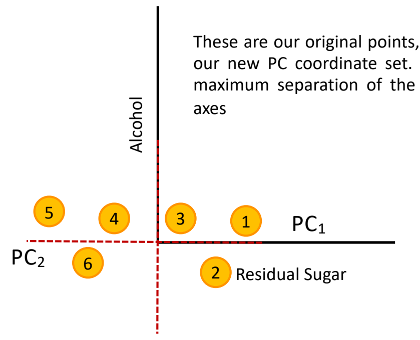
To get the projected points, we rotate back the PC axes to original coordinate set

19

PCA in a nutshell

To get the projected points, we rotate back the PC axes to original coordinate set

These are our original points, but translated to our new PC coordinate set. This set provides maximum separation of the points along the axes



20

PCA in a nutshell

How important is each PC?

SS_1 (distances for PC_1) = eigenvalue for PC_1
SS_2 (distances for PC_2) = eigenvalue for PC_2

$\downarrow$ Convert the eigenvalues to variation for each PC:

$$\frac{SS_1}{N-1} = Variance\ ratio\ PC_1 = \sigma_1 \quad \Rightarrow \quad \frac{\sigma_1}{\sigma_1+\sigma_2}$$

$$\frac{SS_2}{N-1} = Variance\ ratio\ PC_2 = \sigma_2 \quad \Rightarrow \quad \frac{\sigma_2}{\sigma_1+\sigma_2}$$

How much does each PC contribute to the variation in the data

21

The mathematics of PCA

- **Compute the *covariance matrix* $\mathbf{cov}(\boldsymbol{X},\boldsymbol{X}) = \boldsymbol{C}$ of the whole dataset.**

n_samples

$$C = \frac{\sum_{i=1}^{N}(\mathrm{x}_i - \mathrm{x})\sum_{i=1}^{N}(\mathrm{x}_i - \mathrm{x})}{N-1}$$

vector of size n_features

- **Solve eigen-equation**

Covariance matrix (n_features × n_features)

$$Cv = \lambda v$$

Eigenvector of C (n_features × n_features)

Eigenvalue of v (n_features)

by solving determinant determine eigenvalues

$$\det(C - \lambda I)$$

The eigenvalues are the roots of C

- **Determine eigenvectors by substituting ν into the eigen–equation**
- **Form a matrix W with the k highest eigenvectors and use W as follows:**

Transformed samples onto new subspace

$$x' = W^T \cdot \mathrm{x}$$

22

Principal Component Analysis (PCA)

- The process of obtaining principal components from a raw dataset can be simplified in 5 parts :

1. Take the whole dataset consisting of **n_features+1** *dimensions* and ignore the labels such that our new dataset becomes **n_features** *dimensional.* Compute the *mean* for every dimension of the whole dataset.
2. Compute the *covariance matrix* of the whole dataset.
3. Compute *eigenvalues* and the corresponding *eigenvectors*.
4. Sort the eigenvectors by decreasing eigenvalues and choose *k* eigenvectors with the largest eigenvalues to form a **n_features** × *k dimensional* matrix **W.**
5. Use this **n_features** × *k eigenvector matrix* to transform the samples onto the new subspace.

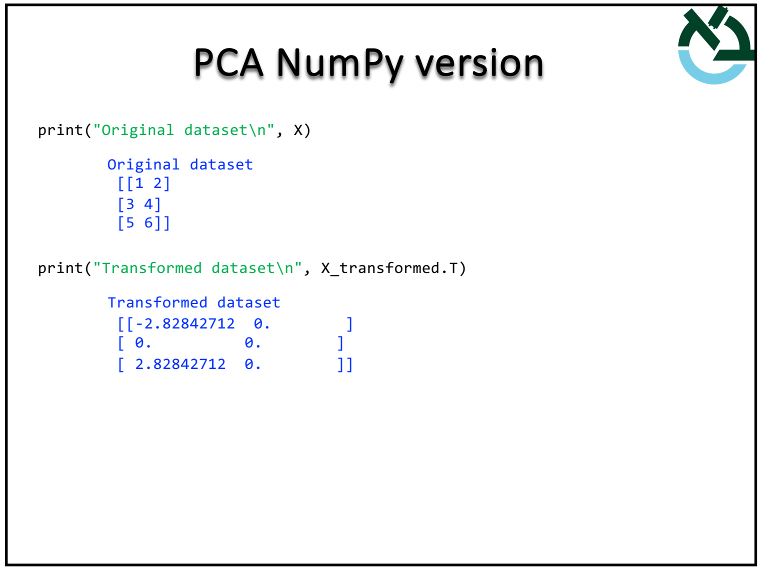
23

PCA NumPy version

```
import numpy as np
import numpy.linalg as linalg
from sklearn.datasets import make_classification

n_features = 2
X = np.array([[1, 2], [3, 4], [5, 6]])
n_samples = X.shape[0]
# We center the data and compute the sample covariance matrix
X_centered = (X - np.mean(X, axis=0))
# Compute covariance matrix
cov_matrix = np.dot(X_centered.T, X_centered) / (n_samples - 1)
# Eigendecomposition of covariance matrix
eigenvalues, eigenvectors = linalg.eig(cov_matrix)
# Sort eigenvalues and associated eigenvectors using index-based sorting
idx = eigenvalues.argsort()[::-1]
eigenvalues = eigenvalues[idx]
eigenvectors = eigenvectors[:,idx]
# Eigenvectors corresponding to the k maximum eigenvalues
W = eigenvectors[:,0:2]
# Transform the samples onto the new subspace
X_transformed = np.dot(W.T, X_centered.T)
```

pca_numpy.ipynb

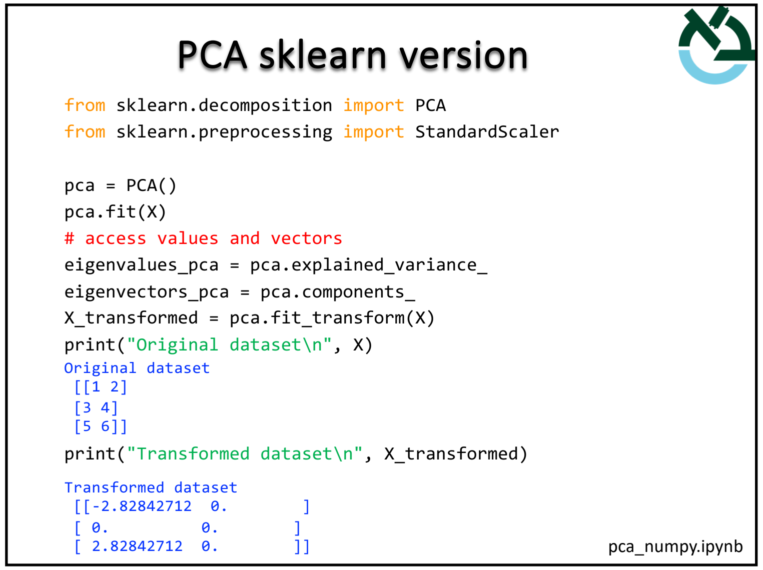
24



PCA NumPy version

```
print("Original dataset\n", X)

       Original dataset
        [[1 2]
        [3 4]
        [5 6]]

print("Transformed dataset\n", X_transformed.T)

       Transformed dataset
        [[-2.82842712  0.        ]
        [ 0.          0.        ]
        [ 2.82842712  0.        ]]
```

25

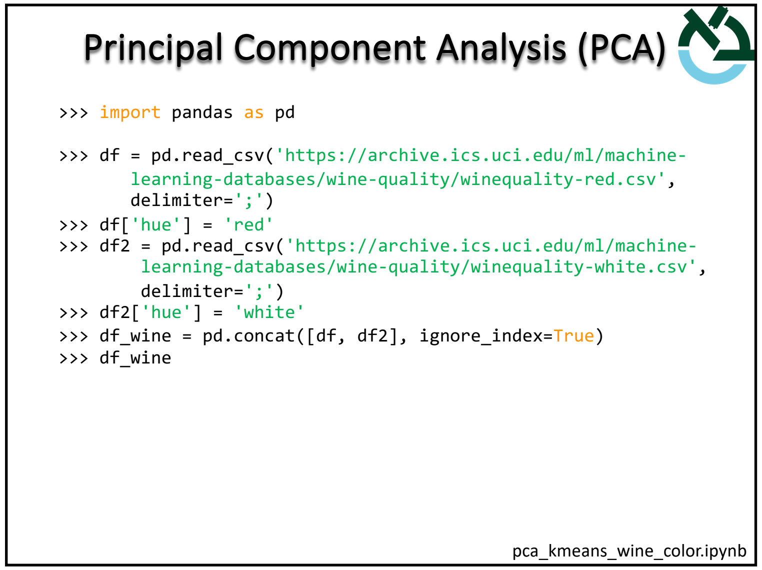


PCA sklearn version

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

pca = PCA()
pca.fit(X)
# access values and vectors
eigenvalues_pca = pca.explained_variance_
eigenvectors_pca = pca.components_
X_transformed = pca.fit_transform(X)
print("Original dataset\n", X)
Original dataset
 [[1 2]
 [3 4]
 [5 6]]
print("Transformed dataset\n", X_transformed)
Transformed dataset
 [[-2.82842712  0.        ]
 [ 0.          0.        ]
 [ 2.82842712  0.        ]]
```

pca_numpy.ipynb

26

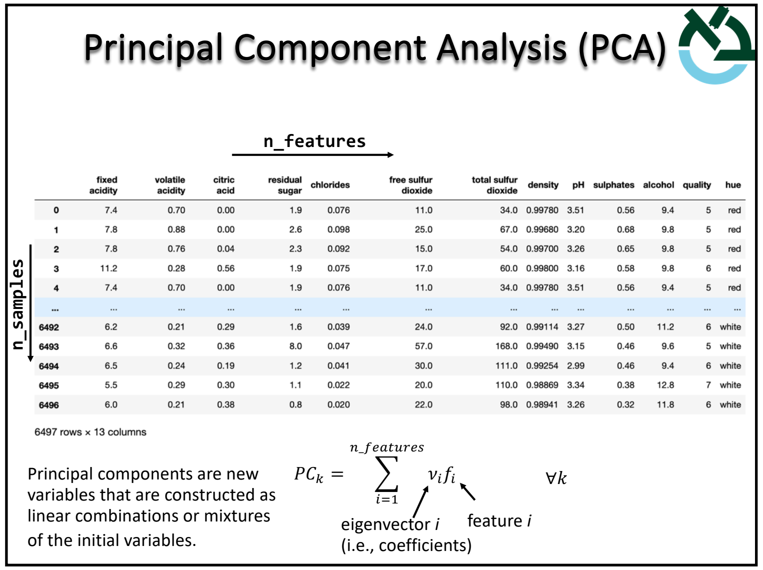


Principal Component Analysis (PCA)

```
>>> import pandas as pd

>>> df = pd.read_csv('https://archive.ics.uci.edu/ml/machine-
       learning-databases/wine-quality/winequality-red.csv',
       delimiter=';')
>>> df['hue'] = 'red'
>>> df2 = pd.read_csv('https://archive.ics.uci.edu/ml/machine-
        learning-databases/wine-quality/winequality-white.csv',
        delimiter=';')
>>> df2['hue'] = 'white'
>>> df_wine = pd.concat([df, df2], ignore_index=True)
>>> df_wine
```

pca_kmeans_wine_color.ipynb

27



Principal Component Analysis (PCA)

n_features
n_samples

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality	hue
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5	red
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	9.8	5	red
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65	9.8	5	red
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58	9.8	6	red
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5	red
...	...	...	...	...	...	...	...	...	...	...	...	...	...
6492	6.2	0.21	0.29	1.6	0.039	24.0	92.0	0.99114	3.27	0.50	11.2	6	white
6493	6.6	0.32	0.36	8.0	0.047	57.0	168.0	0.99490	3.15	0.46	9.6	5	white
6494	6.5	0.24	0.19	1.2	0.041	30.0	111.0	0.99254	2.99	0.46	9.4	6	white
6495	5.5	0.29	0.30	1.1	0.022	20.0	110.0	0.98869	3.34	0.38	12.8	7	white
6496	6.0	0.21	0.38	0.8	0.020	22.0	98.0	0.98941	3.26	0.32	11.8	6	white

6497 rows × 13 columns

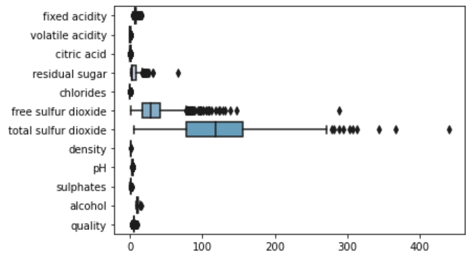
Principal components are new variables that are constructed as linear combinations or mixtures of the initial variables.

$$PC_k = \sum_{i=1}^{n\_features} \underbrace{v_i}_{\text{eigenvector } i \text{ (i.e., coefficients)}} \underbrace{f_i}_{\text{feature } i} \qquad \forall k$$

28

Principal Component Analysis (PCA)

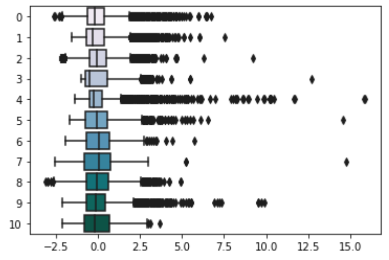
```
>>> import seaborn as sns

>>> ax = sns.boxplot(data=df_wine, orient='h', palette='PuBuGn')
```



29

Principal Component Analysis (PCA) Data standardization

```
>>> from sklearn.preprocessing import StandardScaler

>>> data_pca = df_wine.copy()
>>> data_pca = data_pca.drop(labels = ['quality', 'hue'],axis = 1)
>>> # StandardScaler() scales the data. The fit_transform() module fits these new
>>> # values to the data, and stores them, replacing the old values.
>>> data_pca = StandardScaler().fit_transform(data_pca)
>>> # Plot the transformed (scaled and centered) data:
>>> ax = sns.boxplot(data=data_pca, orient='h', palette='PuBuGn')
```



30

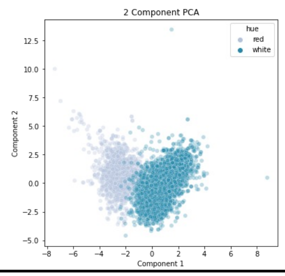
Principal Component Analysis (PCA)

```
>>> from sklearn.decomposition import PCA

>>> # Apply PCA on the transformed (scaled and centered) data:
>>> pca = PCA(n_components=2)
>>> print(type(pca))
<class 'sklearn.decomposition._pca.PCA'>
>>> X = pca.fit_transform(data_pca)
>>> X.shape
(6497, 2)
>>> X
array([[-3.20599617,  0.41652332],
       [-3.03905081,  1.10746213],
       [-3.07189347,  0.87896444],
       ...,
       [ 0.5711325 , -0.72266165],
       [ 0.09005243, -3.54577991],
       [ 0.51257566, -2.89104008]])
```

31

Principal Component Analysis (PCA)

```
>>> import matplotlib.pyplot as plt

>>> pca_dataset = pd.DataFrame(data = X, columns = ['component1', 'component2'])
>>> pca_dataset['hue'] = df_wine['hue']
>>> plt.figure()
>>> plt.figure(figsize=(6,6))
>>> plt.xlabel('Component 1')
>>> plt.ylabel('Component 2')
>>> plt.title('2 Component PCA')
>>> sns.scatterplot(x = pca_dataset['component1'], y = pca_dataset['component2'],
                    hue=pca_dataset['hue'], alpha=0.3,palette='PuBuGn')
```



Similar datapoints are closer together, forming a cluster. For this dataset we see that white wine and red wine form two separate clusters.

32

Principal Component Analysis (PCA)

```
>>> # Trying to decipher the meaning of the principal components
>>> print("Meaning of the 2 components:")
>>> eigenvectors = pca.components_ # The eigenvectors
>>> print(eigenvectors.shape)
(2, 11)
>>> pd.DataFrame(eigenvectors,
       columns=df_wine.keys().drop(labels = ['quality', 'hue']))
```

n_features

n_components

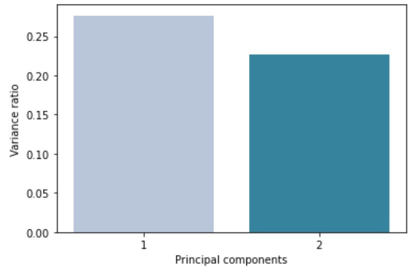
	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
0	-0.239850	-0.378601	0.147806	0.344812	-0.289520	0.433571	0.488749	-0.045997	-0.215840	-0.295267	-0.108741
1	0.335908	0.120606	0.183417	0.334376	0.311061	0.068348	0.086109	0.583883	-0.160638	0.190101	-0.464181

```
>>> eigenvalues = pca.explained_variance_
>>> print(pd.Series(eigenvalues))

            0    3.043073
            1    2.494307
            dtype: float64
```

33

Principal Component Analysis (PCA)

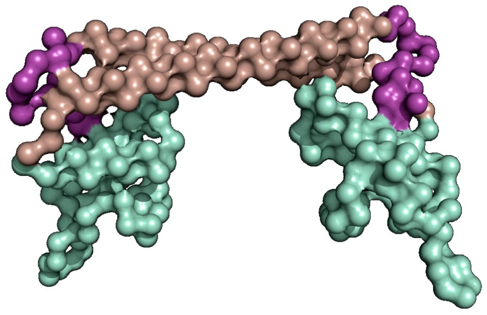
```
>>> # How much does each component explain the original dataset.
>>> string = [s+1 for s in range(len(pca.explained_variance_ratio_))]
>>> variance_ratio = pd.DataFrame(pca.explained_variance_ratio_,
                    columns=['variance_ratio'], index=string)
>>> ax = sns.barplot(data=variance_ratio, y='variance_ratio',
                    x=variance_ratio.index, palette='PuBuGn')
>>> ax.set(xlabel='Principal components', ylabel='Variance ratio')
```



Try with different number of PCs

34

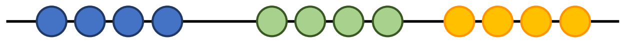
PCA in the Cu-binding protein CueR

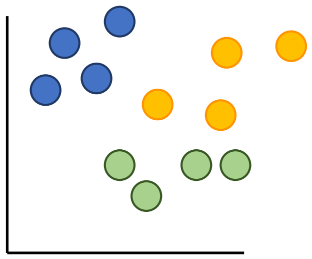


35

K-means Clustering

How do we cluster points in space?

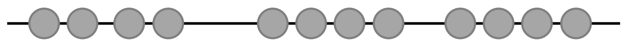




36

K-means Clustering

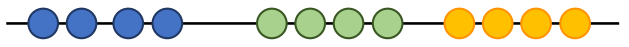
How do we cluster points in space?



37

K-means Clustering

How do we cluster points in space?



38

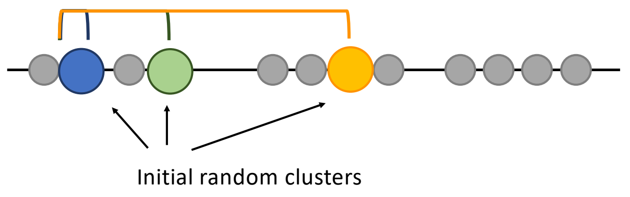
K-means Clustering

Step 1: Select the number of clusters. Let's try 3.

Step 2: Select 3 random clusters

Step 3: Measure distance between all points and all clusters

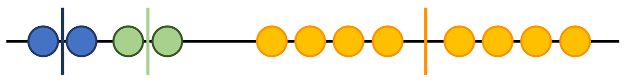
Step 4: Assign each point to the nearest cluster



39

K-means Clustering

In this case we get the following cluster:



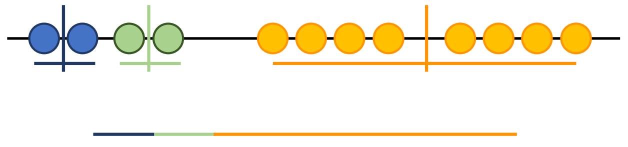
Step 5: Calculate the centroid of each cluster

Step 6: Calculate the distance of all points relative to the centroids and re-cluster. If assignment of points doesn't change, we're done.

40

K-means Clustering

Step 7: Now we calculate the variation within each cluster:



Total variation within the clusters

This clearly isn't an optimal cluster, but K-means doesn't know this. Solution?

41

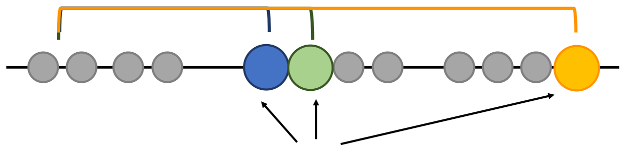
K-means Clustering

Step 1: Select a new set of clusters. Let's try 3.

Step 2: Select 3 random clusters

Step 3: Measure distance between all points and all clusters

Step 4: Assign each point to the nearest cluster

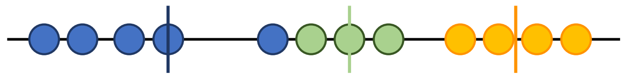


Initial random clusters

42

K-means Clustering

In this case we get the following cluster:



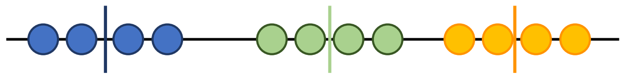
Step 5: Calculate the centroid of each cluster

Step 6: Calculate the distance of all points relative to the centroids and re-cluster. If assignment of points doesn't change, we're done.

43

K-means Clustering

In this case we get the following cluster:



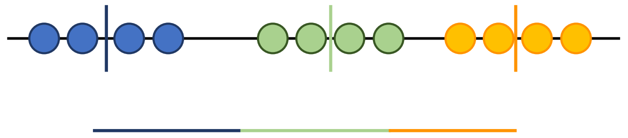
Step 5: Calculate the centroid of each cluster

Step 6: Calculate the distance of all points relative to the centroids and re-cluster. If assignment of points doesn't change, we're done.

44

K-means Clustering

Step 7: Now we calculate the variation within each cluster:

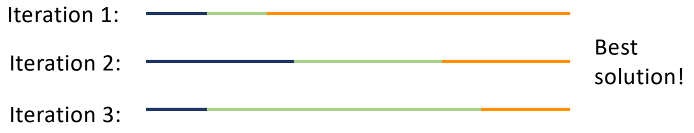


Total variation within the clusters

45

K-means Clustering

Since K-means doesn't know which is the best solution, it will do a certain number of clusters and determine which is the optimal (best distribution of variation within clusters). Let's say we tried 3 random initial guesses:



Total variation within the clusters

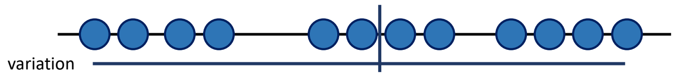
46

K-means Clustering

What's the optimal value for K?

Start with K=1 and then increase the number of K's.

K=1:



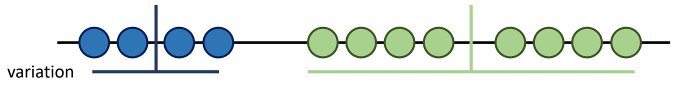
47

K-means Clustering

What's the optimal value for K?

Start with K=1 and then increase the number of K's.

K=2:



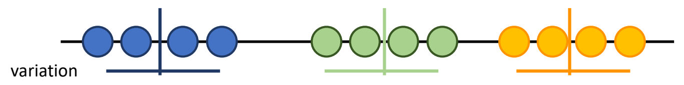
48

K-means Clustering

What's the optimal value for K?

Start with K=1 and then increase the number of K's.

K=3:



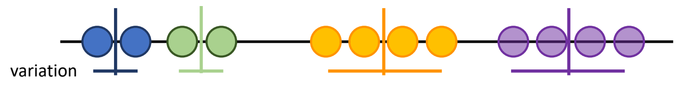
49

K-means Clustering

What's the optimal value for K?

Start with K=1 and then increase the number of K's.

K=4:

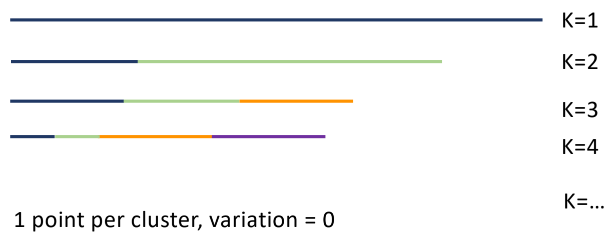


50

K-means Clustering

What's the optimal value for K?

Comparing the variation with K:



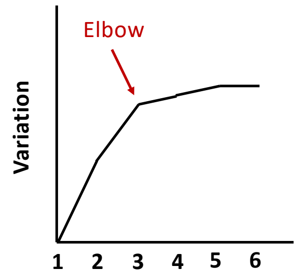
1 point per cluster, variation = 0

51

K-means Clustering

What's the optimal value for K?

Plotting the variation vs. K (elbow plot):



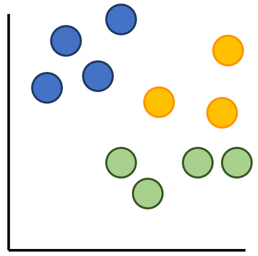
We'll see in a bit that we'll use something called inertia instead of variation

52

K-means Clustering

How do we cluster points in 2D, 3D, …, *N*D space?

Same idea!



53

K-means Clustering

We have to decide how to compute the distance between points.

The K-means algorithm aims to choose centroids that minimize the inertia, or within-cluster sum-of-squares criterion:

All points in cluster j

$$\sum_{i=0}^{N} \min_{\mu_j \in C} \left(\left\| x_i - \mu_j \right\|^2 \right)$$

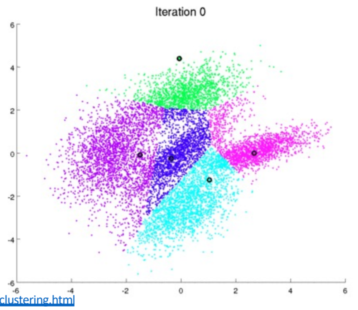
Centroid j

Sum over points in cluster, j

54

K-means Clustering

1. Guess number of clusters *K*
2. Form the initial clusters
3. Reiterate until convergence

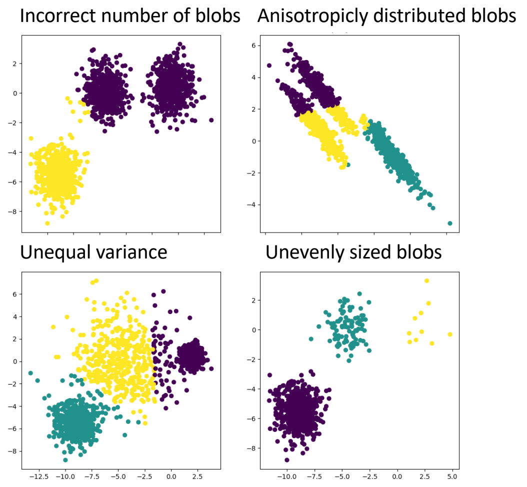


K-Means in action

Source: http://mcla.ug/blog/k-means-clustering.html

55

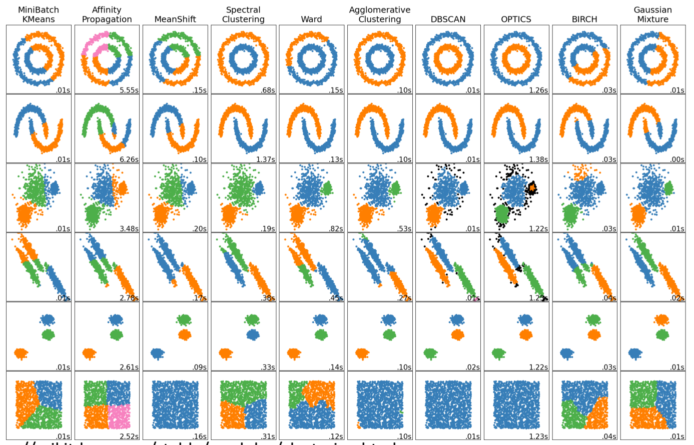
Problematic cases for K-means



https://scikit-learn.org/stable/auto_examples/cluster/plot_cluster_comparison.html

56

Behavior of clustering methods



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

57

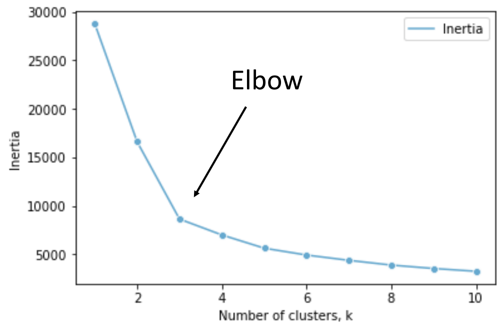
K-means example

```
from sklearn.cluster import KMeans
inertia = []
# Creating 10 K-Mean models while varying the number of clusters (k)
# An elbow in the graph indicates the right number of clusters
for k in range(1,11):
    model = KMeans(n_clusters=k, init='k-means++')
    model.fit(pca_dataset.iloc[:,:2]) # Fit model to samples
    inertia.append(model.inertia_) # Append the inertia to
                                   # the list of inertias
inertia = pd.DataFrame({'Inertia':inertia}, index=range(1,11))
ax = sns.lineplot(data=inertia, marker='o', palette='PuBuGn')
ax.set(xlabel='Number of clusters, k', ylabel='Inertia')
```

pca_kmeans_wine_color.ipynb

58

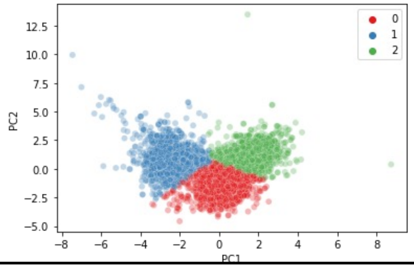
K-means example



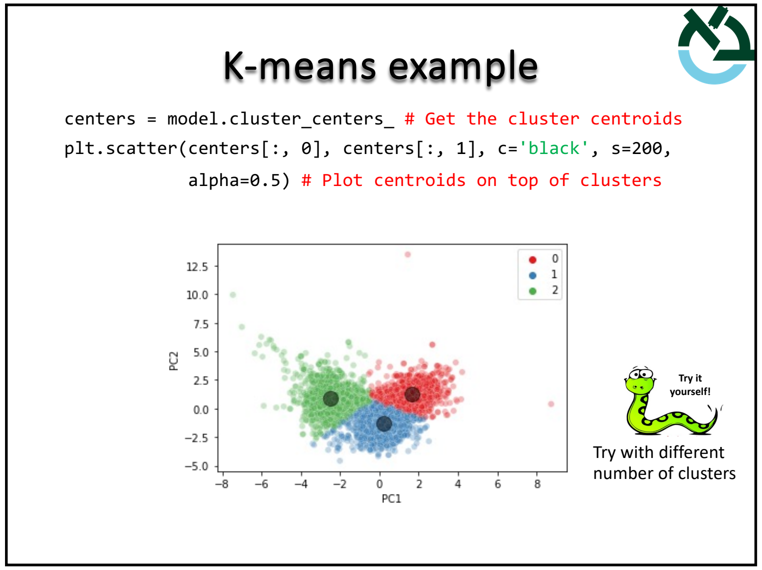
59

K-means example

```
model = KMeans(n_clusters=3, init='k-means++')
model.fit(pca_dataset.iloc[:,:2])
labels = model.predict(pca_dataset.iloc[:,:2])
sns.scatterplot(x = pca_dataset[0], y = pca_dataset[1],
                alpha=0.3, hue=labels, palette='Set1')
ax.set(xlabel='PC1', ylabel='PC2')
```

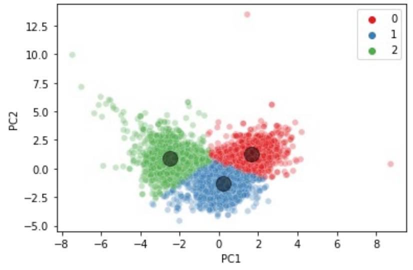


60



K-means example

```
centers = model.cluster_centers_ # Get the cluster centroids
plt.scatter(centers[:, 0], centers[:, 1], c='black', s=200,
            alpha=0.5) # Plot centroids on top of clusters
```



Try with different number of clusters

61