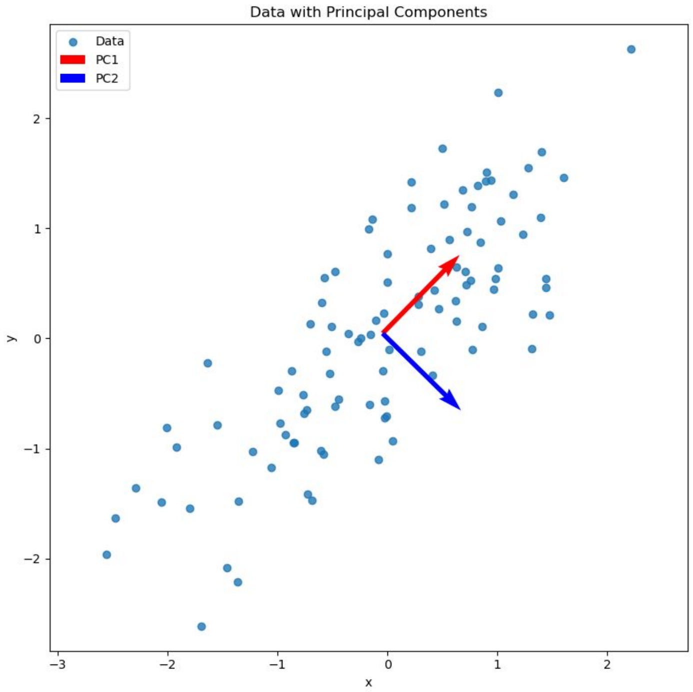
Principal Component Analysis & Linear Discriminant Analysis

Jolla Kullgren
Department of Chemistry - Ångström

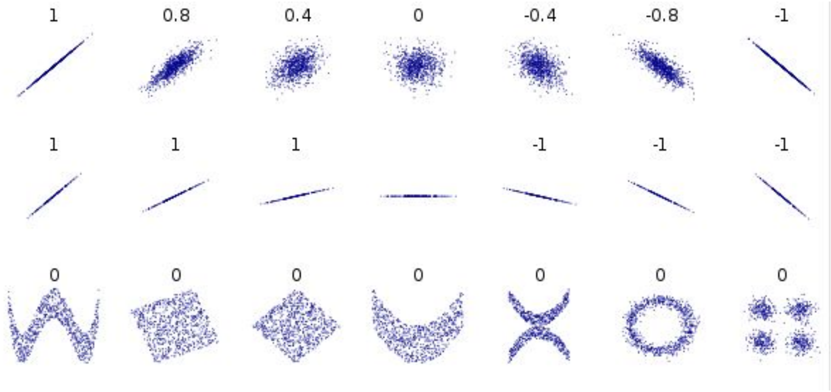
Principal Component Analysis (PCA)



- PCA is a way to reduce dimensionality of multivariate data.
- PCA identifies the principal components, which are *linear combinations* of the original variables that capture the maximum variance in the data.

UPPSALA UNIVERSITET

Principal Component Analysis (PCA) -Covariance, variance and correlation.



Example for wikipedia showing the Pearson correlation coefficient for a number of data-sets.

- Variance is a measure of spread in the data.
- Covariance is a measure of the joint variability of two random variables.
- Pearson correlation coefficient compare covariance and variance:

$$r = \frac{\mathrm{Cov}(X_i, X_j)}{\sqrt{\mathrm{Var}(X_i) \cdot \mathrm{Var}(X_j)}}$$

Principal Component Analysis (PCA) -Covariance, variance and correlation.

$$\mathbf{S} = \begin{bmatrix} \mathrm{Var}(X_1) & \mathrm{Cov}(X_1, X_2) & \dots & \mathrm{Cov}(X_1, X_n) \\ \mathrm{Cov}(X_2, X_1) & \mathrm{Var}(X_2) & \dots & \mathrm{Cov}(X_2, X_n) \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{Cov}(X_n, X_1) & \mathrm{Cov}(X_n, X_2) & \dots & \mathrm{Var}(X_n) \end{bmatrix}$$

$$\mathrm{Cov}(X_i, X_j) = \frac{\sum_{k=1}^{n}(X_{i,k}-\bar{X}_i)(X_{j,k}-\bar{X}_j)}{n-1}$$

- To make a PCA, we would like to minimize covariance and maximize variance.
- We can do this by diagonalizing S.

S using matrices

- With start with **X** whose columns are our independent variables.

$(k \times n)$

$$\mathbf{X} = [X_1, X_2, \cdots, X_n]$$

- We subtract the mean for each variable. I.e. each column is centered around zero.

$(k \times n)$

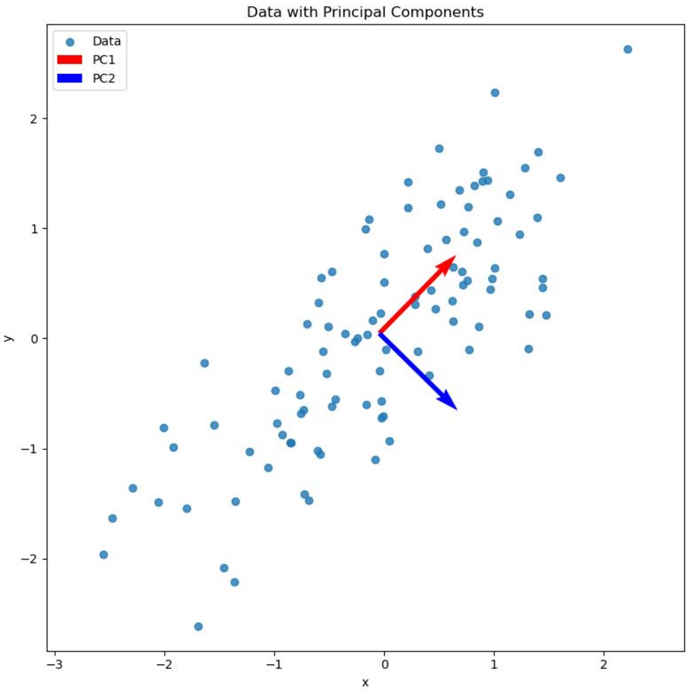
$$\bar{\mathbf{X}} = [X_1 - \mathbf{1}\bar{X}_1, X_2 - \mathbf{1}\bar{X}_2, \cdots, X_n - \mathbf{1}\bar{X}_n]$$

- We can now form **S** from a simple matrix multiplication.

$(n \times n)$

$$\mathbf{S} = \frac{1}{n-1}\bar{\mathbf{X}}^T\bar{\mathbf{X}}$$

Principal Component Analysis (PCA)



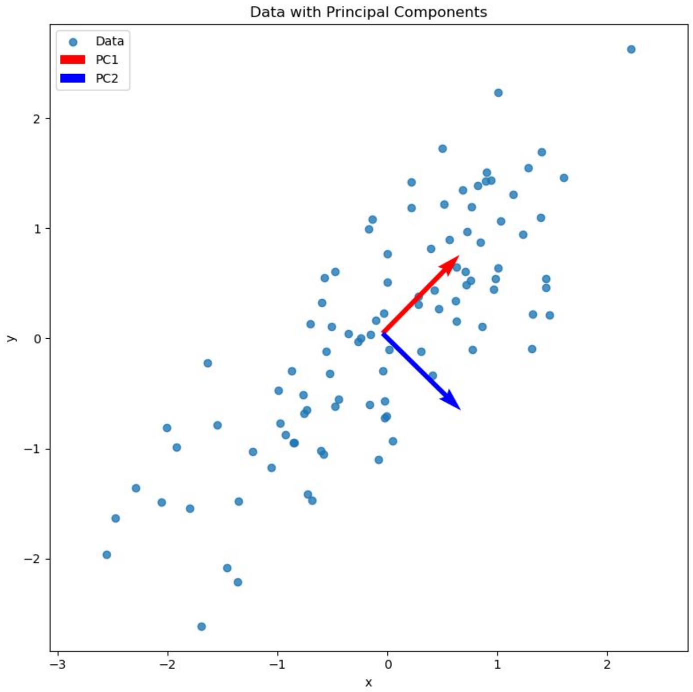
- The variance (in the PCs) and the principal components themselves, corresponds to the eigenvectors and eigenvalues of the covariance matrix (**S**), respectively.

$$\mathbf{S} = \begin{bmatrix} \mathrm{Var}(X_1) & \mathrm{Cov}(X_1, X_2) & \dots & \mathrm{Cov}(X_1, X_n) \\ \mathrm{Cov}(X_2, X_1) & \mathrm{Var}(X_2) & \dots & \mathrm{Cov}(X_2, X_n) \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{Cov}(X_n, X_1) & \mathrm{Cov}(X_n, X_2) & \dots & \mathrm{Var}(X_n) \end{bmatrix}$$

A simple example...

```
X1 = [1.0,2.0,3.0,4.0]
X2 = [2.0,4.0,6.0,8.0]

mean_X1 = (1.0 + 2.0 + 3.0 + 4.0) / 4.0 = 2.5
mean_X2 = (2.0 + 4.0 + 6.0 + 8.0) / 4.0 = 5.0

X1_centered = X1-mean_X1 = [-1.5,-0.5,0.5,1.5]
X2_ceneterd = X2-mean_X2 = [-3.0,-1.0,1.0,3.0]

S_11 = ( (-1.5)*(-1.5)+(-0.5)*(-0.5)+0.5*0.5+1.5*1.5 ) / 3.0 = 1.6667
S_12 = ( (-1.5)*(-3.0)+(-0.5)*(-1.0)+0.5*1.0+1.5*3.0 ) / 3.0 = 3.3333
S_21 = S_12 = 3.3333
S_22 = ( (-3.0)*(-3.0)+(-1.0)*(-1.0)+1.0*1.0+3.0*3.0 ) / 3.0 = 6.6667

S = [[1.6667, 3.3333],
     [3.3333, 6.6667]]

eigenvalues_of_S =[0.,8.33333333]
eigenvectors_of_S = [[-0.89442719,-0.4472136 ],
                     [  0.4472136,-0.89442719]]
```

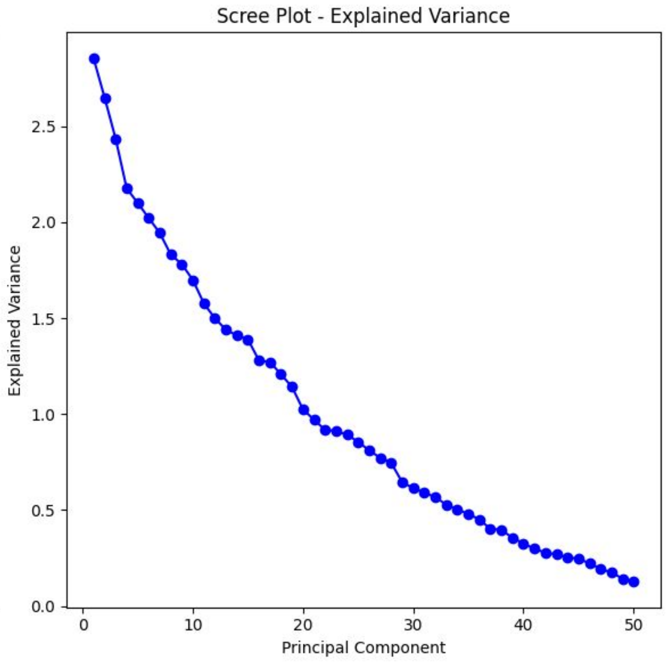
Principal Component Analysis (PCA) - standardizing



- If the ranges for the different independent variables are very different, our PCA analysis becomes skewed.
- One way to remedy this problem is to standardize the x-values.
- The procedure: Center to the mean and component wise scale to unit variance.

UPPSALA UNIVERSITET

Principal Component Analysis (PCA) - Scree-plot



- In a scree plot, the eigenvalues or variances in the PCs are plotted in descending order.
- By analyzing the scree plot, you can make an informed decision about the number of principal components or factors to retain.
- Generally, you would select the number of components or factors before the drop-off point in order to capture most of the relevant information while reducing the dimensionality of the data.
- Kaiser-Guttman criterion: keep only components with "Explained variance" larger than 1. (Data needs to be standardized!)

UPPSALA UNIVERSITET

A simple code:

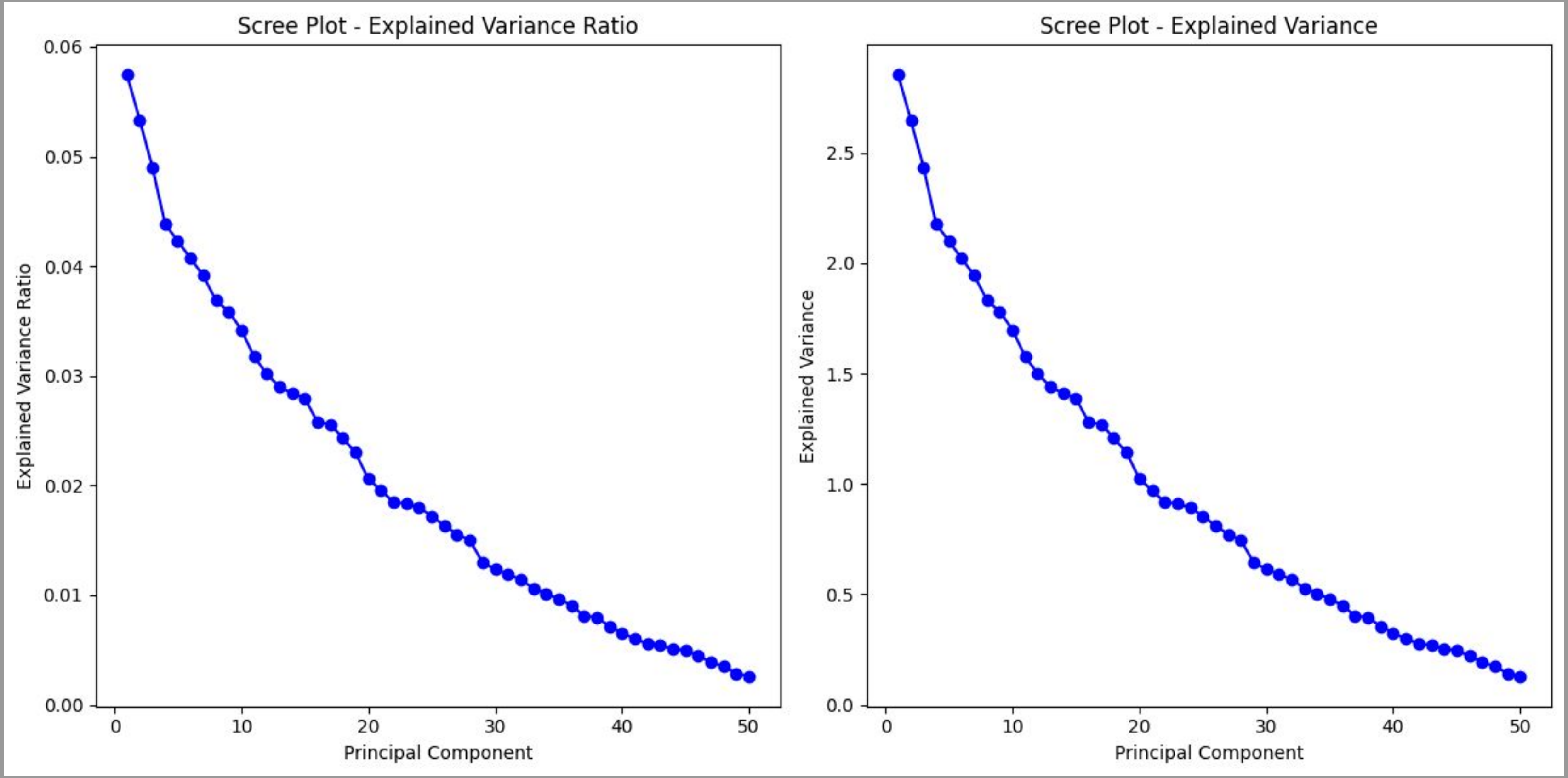
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

# Generate a synthetic high-dimensional dataset
np.random.seed(42)
num_samples = 100
num_features = 50
X = np.random.randn(num_samples, num_features)

# Perform PCA
pca = PCA()
X_pca = pca.fit_transform(X)

# Calculate the explained variance ratio and variance
explained_variance_ratio = pca.explained_variance_ratio_
explained_variance = pca.explained_variance_
```

Conveniently enough, there is a built-in function to obtained the explained variance from sklearn!

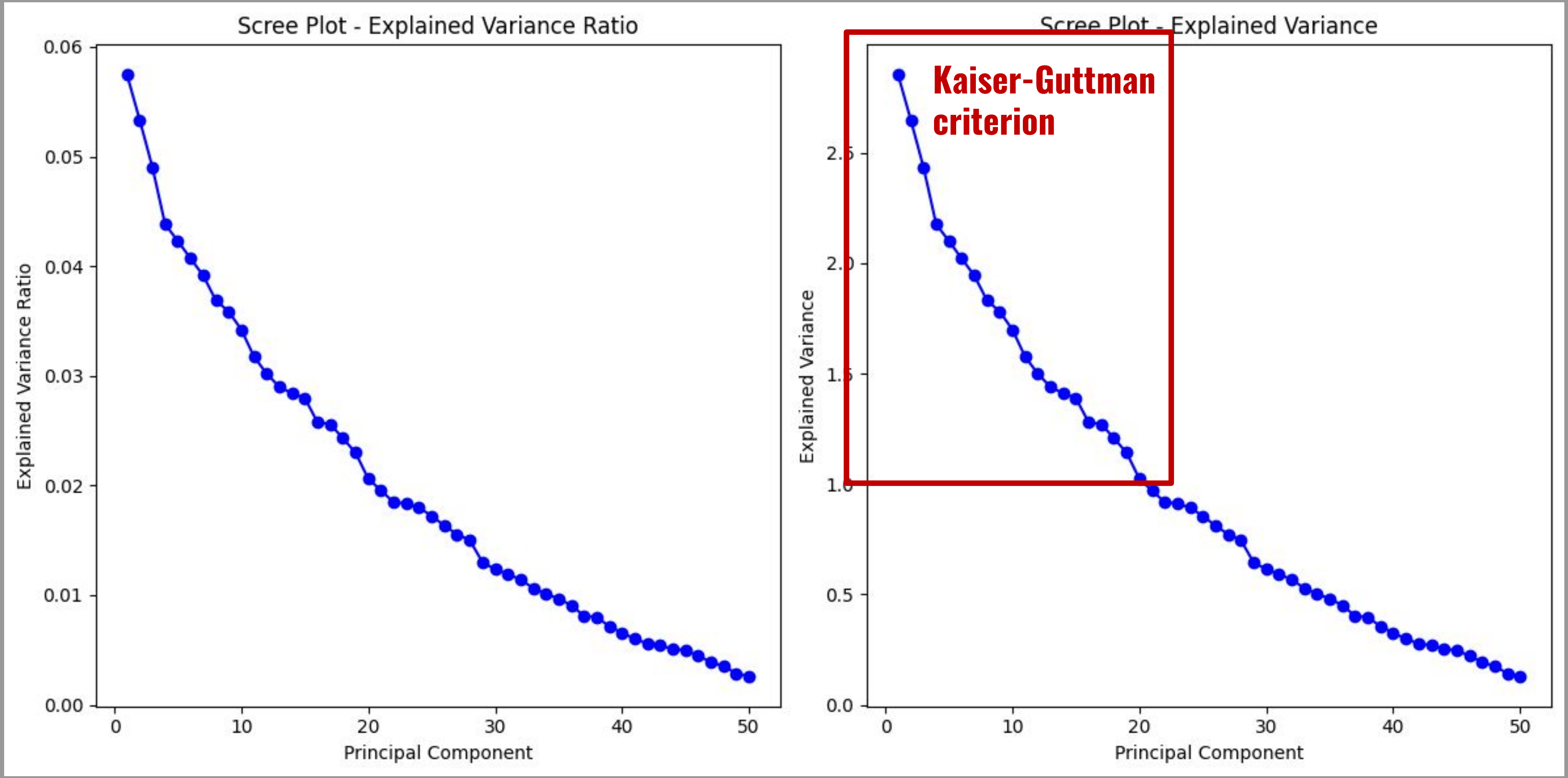
A simple code (cont.):

```
# Create the scree plot for explained variance ratio
num_components = len(explained_variance_ratio)
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(np.arange(1, num_components + 1), explained_variance_ratio, 'bo-')
plt.xlabel('Principal Component')
plt.ylabel('Explained Variance Ratio')
plt.title('Scree Plot - Explained Variance Ratio')

# Create the scree plot for actual variance
plt.subplot(1, 2, 2)
plt.plot(np.arange(1, num_components + 1), explained_variance, 'bo-')
plt.xlabel('Principal Component')
plt.ylabel('Explained Variance')
plt.title('Scree Plot - Explained Variance')

plt.tight_layout()
plt.show()
```

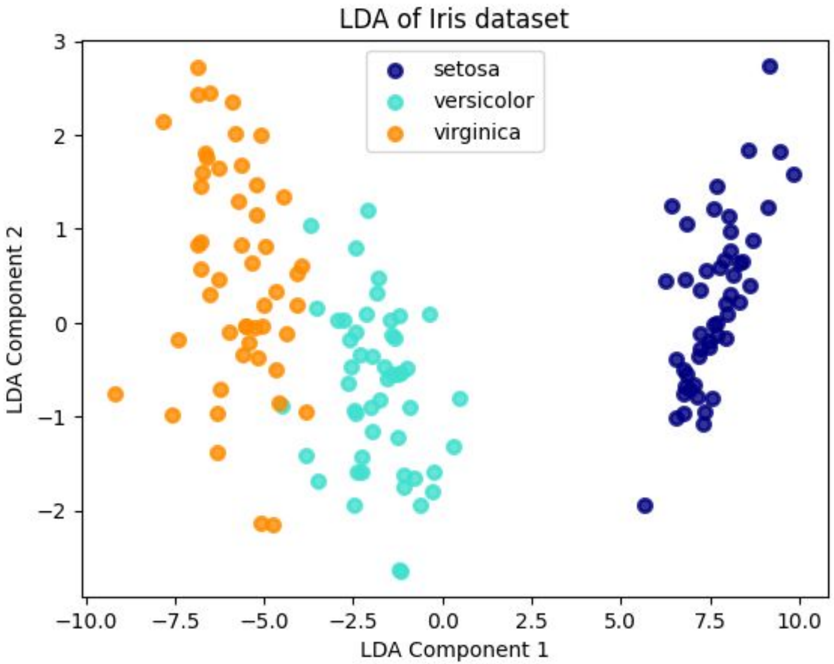
A simple code (Output):



A simple code (Output):



Linear Discriminant Analysis (LDA)



- In LDA we look for a linear combination that characterizes or separates two or more classes of objects.
- Discriminant analysis has continuous independent variables and a categorical dependent variable (an integer or class label).

UPPSALA UNIVERSITET

A simple code (input)

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.datasets import load_iris

# Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Perform Linear Discriminant Analysis
lda = LinearDiscriminantAnalysis(n_components=2)
X_lda = lda.fit_transform(X, y)

# Plot the results
target_names = iris.target_names
colors = ['navy', 'turquoise', 'darkorange']
plt.figure()
for color, i, target_name in zip(colors, [0, 1, 2], target_names):
     plt.scatter(X_lda[y == i, 0], X_lda[y == i, 1], color=color, alpha=0.8, lw=2,
label=target_name)
plt.legend(loc='best', shadow=False, scatterpoints=1)
plt.title('LDA of Iris dataset')
plt.xlabel('LDA Component 1')
plt.ylabel('LDA Component 2')
plt.show()
```

A simple code (input)

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.datasets import load_iris

# Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Perform Linear Discriminant Analysis
lda = LinearDiscriminantAnalysis(n_components=2)
X_lda = lda.fit_transform(X, y)

# Plot the results
target_names = iris.target_names
colors = ['navy', 'turquoise', 'darkorange']
plt.figure()
for color, i, target_name in zip(colors, [0, 1, 2], target_names):
     plt.scatter(X_lda[y == i, 0], X_lda[y == i, 1], color=color, alpha=0.8, lw=2,
label=target_name)
plt.legend(loc='best', shadow=False, scatterpoints=1)
plt.title('LDA of Iris dataset')
plt.xlabel('LDA Component 1')
plt.ylabel('LDA Component 2')
plt.show()
```

We start by loading a data-set.

A simple code (input)

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.datasets import load_iris

# Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Perform Linear Discriminant Analysis
lda = LinearDiscriminantAnalysis(n_components=2)
X_lda = lda.fit_transform(X, y)

# Plot the results
target_names = iris.target_names
colors = ['navy', 'turquoise', 'darkorange']
plt.figure()
for color, i, target_name in zip(colors, [0, 1, 2], target_names):
     plt.scatter(X_lda[y == i, 0], X_lda[y == i, 1], color=color, alpha=0.8, lw=2,
label=target_name)
plt.legend(loc='best', shadow=False, scatterpoints=1)
plt.title('LDA of Iris dataset')
plt.xlabel('LDA Component 1')
plt.ylabel('LDA Component 2')
plt.show()
```

Performing the LDA is as simple as this!

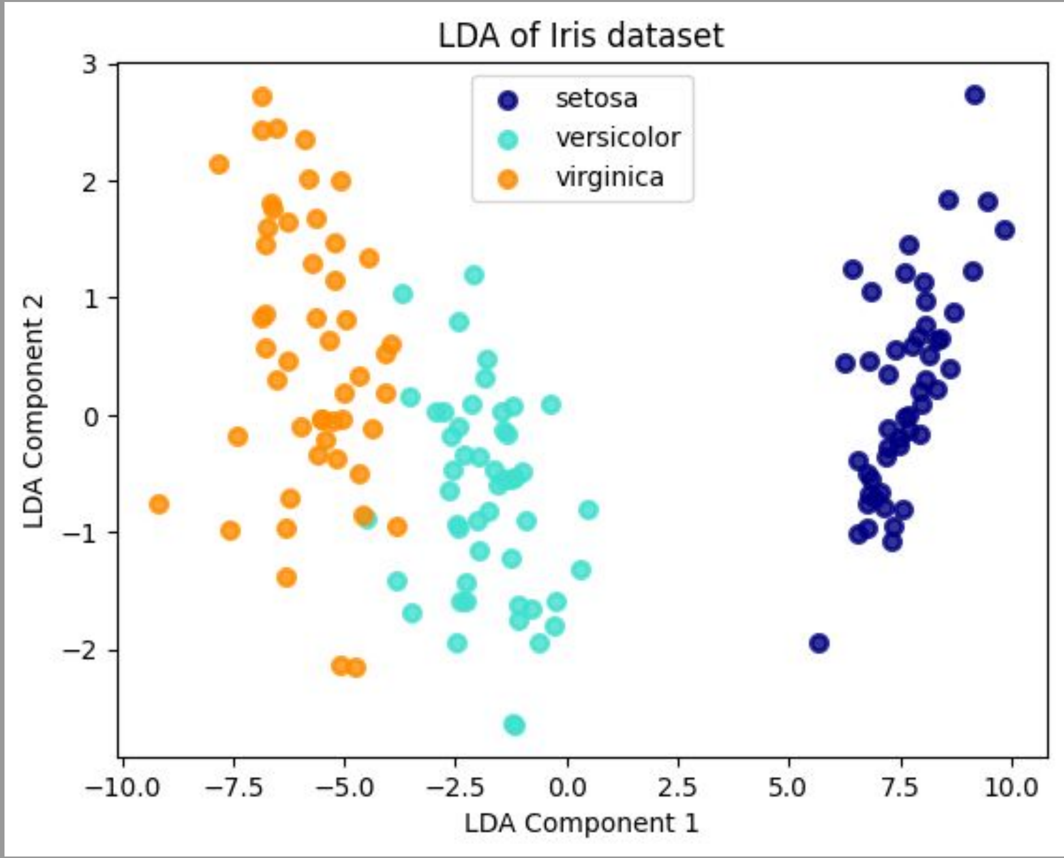
A simple code (input)

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.datasets import load_iris

# Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Perform Linear Discriminant Analysis
lda = LinearDiscriminantAnalysis(n_components=2)
X_lda = lda.fit_transform(X, y)

# Plot the results
target_names = iris.target_names
colors = ['navy', 'turquoise', 'darkorange']
plt.figure()
for color, i, target_name in zip(colors, [0, 1, 2], target_names):
     plt.scatter(X_lda[y == i, 0], X_lda[y == i, 1], color=color, alpha=0.8, lw=2,
label=target_name)
plt.legend(loc='best', shadow=False, scatterpoints=1)
plt.title('LDA of Iris dataset')
plt.xlabel('LDA Component 1')
plt.ylabel('LDA Component 2')
plt.show()
```

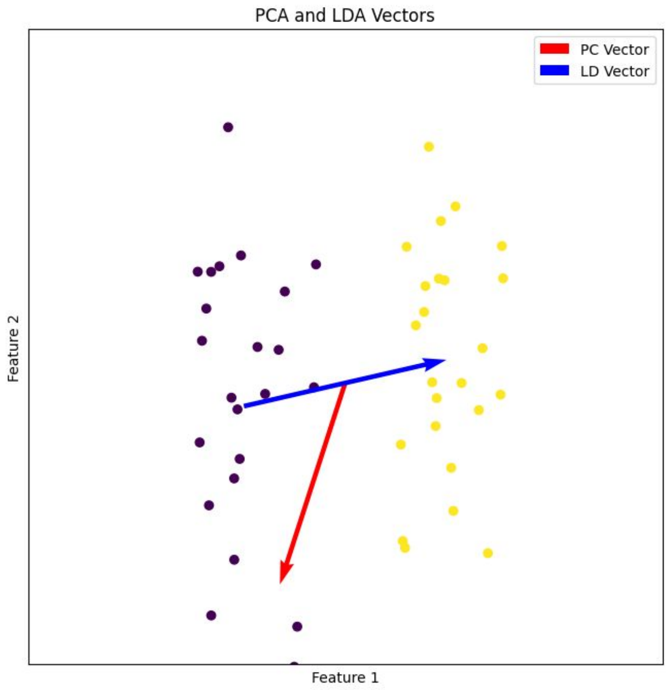
Here we plot each group in the space spanned by LD1 and LD2. We color each group.

Note the clever use of masking: X_lda[y ==i,0]. This selects all elements from X_lda where the corresponding y-value is == i.

A simple code (output)



PCA vs LDA



- PCA identifies the principal components, which are *linear combinations* of the original variables that capture the maximum variance in the data.

- In LDA we look for a linear combination that characterizes or separates two or more classes of objects.