Laboratory 1

Data analysis and visualization with machine learning

A. An introduction to machine learning with scikit-learn Scikit-learn is a Python module integrating classic machine learning algorithms in the tightly-knit world of scientific Python packages (NumPy, SciPy, matplotlib). Analyze the following tutorial by executing the given examples: http://scikit-learn.org/stable/tutorial/basic/tutorial.html

1 - In Python, usually, the functions are included in libraries which sould be imported. For example to import the scikit-learn library :

```
In [106…
from sklearn import *
```

2 - Import the libraries numpy (scientific compûtation) and matplotlib.pyplot (visualization).

```
In [106…
import numpy as np
import matplotlib.pyplot as mp
```

3 -Load the Iris dataset using:

```
In [106…
iris = datasets.load_iris()
```

The variable iris is an object which contains the dataset matrix iris.data), a vector containing the lable/classes (target), the name of variables (feature_names) and the name of classes (target_names).

4- Print the number of data, names of variables and the name of classes (use print).

```
In [106…
print(iris.data)
```

```
[[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]
 [4.8 3.4 1.6 0.2]
 [4.8 3.  1.4 0.1]
 [4.3 3.  1.1 0.1]
 [5.8 4.  1.2 0.2]
 [5.7 4.4 1.5 0.4]
 [5.4 3.9 1.3 0.4]
 [5.1 3.5 1.4 0.3]
 [5.7 3.8 1.7 0.3]
 [5.1 3.8 1.5 0.3]
 [5.4 3.4 1.7 0.2]
 [5.1 3.7 1.5 0.4]
 [4.6 3.6 1.  0.2]
 [5.1 3.3 1.7 0.5]
 [4.8 3.4 1.9 0.2]
 [5.  3.  1.6 0.2]
 [5.  3.4 1.6 0.4]
 [5.2 3.5 1.5 0.2]
 [5.2 3.4 1.4 0.2]
 [4.7 3.2 1.6 0.2]
 [4.8 3.1 1.6 0.2]
 [5.4 3.4 1.5 0.4]
 [5.2 4.1 1.5 0.1]
 [5.5 4.2 1.4 0.2]
 [4.9 3.1 1.5 0.2]
 [5.  3.2 1.2 0.2]
 [5.5 3.5 1.3 0.2]
 [4.9 3.6 1.4 0.1]
 [4.4 3.  1.3 0.2]
 [5.1 3.4 1.5 0.2]
 [5.  3.5 1.3 0.3]
 [4.5 2.3 1.3 0.3]
 [4.4 3.2 1.3 0.2]
 [5.  3.5 1.6 0.6]
 [5.1 3.8 1.9 0.4]
 [4.8 3.  1.4 0.3]
 [5.1 3.8 1.6 0.2]
 [4.6 3.2 1.4 0.2]
 [5.3 3.7 1.5 0.2]
 [5.  3.3 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]
 [5.7 2.8 4.5 1.3]
 [6.3 3.3 4.7 1.6]
 [4.9 2.4 3.3 1. ]
 [6.6 2.9 4.6 1.3]
 [5.2 2.7 3.9 1.4]
 [5.  2.  3.5 1. ]
 [5.9 3.  4.2 1.5]
 [6.  2.2 4.  1. ]
 [6.1 2.9 4.7 1.4]
 [5.6 2.9 3.6 1.3]
 [6.7 3.1 4.4 1.4]
 [5.6 3.  4.5 1.5]
 [5.8 2.7 4.1 1. ]
 [6.2 2.2 4.5 1.5]
 [5.6 2.5 3.9 1.1]
 [5.9 3.2 4.8 1.8]
 [6.1 2.8 4.  1.3]
 [6.3 2.5 4.9 1.5]
 [6.1 2.8 4.7 1.2]
 [6.4 2.9 4.3 1.3]
 [6.6 3.  4.4 1.4]
 [6.8 2.8 4.8 1.4]
 [6.7 3.  5.  1.7]
 [6.  2.9 4.5 1.5]
 [5.7 2.6 3.5 1. ]
 [5.5 2.4 3.8 1.1]
 [5.5 2.4 3.7 1. ]
 [5.8 2.7 3.9 1.2]
 [6.  2.7 5.1 1.6]
 [5.4 3.  4.5 1.5]
 [6.  3.4 4.5 1.6]
 [6.7 3.1 4.7 1.5]
 [6.3 2.3 4.4 1.3]
 [5.6 3.  4.1 1.3]
 [5.5 2.5 4.  1.3]
 [5.5 2.6 4.4 1.2]
 [6.1 3.  4.6 1.4]
 [5.8 2.6 4.  1.2]
 [5.  2.3 3.3 1. ]
 [5.6 2.7 4.2 1.3]
 [5.7 3.  4.2 1.2]
 [5.7 2.9 4.2 1.3]
 [6.2 2.9 4.3 1.3]
 [5.1 2.5 3.  1.1]
 [5.7 2.8 4.1 1.3]
 [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]
 [6.5 3.  5.8 2.2]
 [7.6 3.  6.6 2.1]
 [4.9 2.5 4.5 1.7]
 [7.3 2.9 6.3 1.8]
 [6.7 2.5 5.8 1.8]
 [7.2 3.6 6.1 2.5]
 [6.5 3.2 5.1 2. ]
 [6.4 2.7 5.3 1.9]
 [6.8 3.  5.5 2.1]
 [5.7 2.5 5.  2. ]
 [5.8 2.8 5.1 2.4]
 [6.4 3.2 5.3 2.3]
 [6.5 3.  5.5 1.8]
 [7.7 3.8 6.7 2.2]
 [7.7 2.6 6.9 2.3]
 [6.  2.2 5.  1.5]
 [6.9 3.2 5.7 2.3]
 [5.6 2.8 4.9 2. ]
 [7.7 2.8 6.7 2. ]
 [6.3 2.7 4.9 1.8]
 [6.7 3.3 5.7 2.1]
 [7.2 3.2 6.  1.8]
 [6.2 2.8 4.8 1.8]
 [6.1 3.  4.9 1.8]
 [6.4 2.8 5.6 2.1]
 [7.2 3.  5.8 1.6]
 [7.4 2.8 6.1 1.9]
 [7.9 3.8 6.4 2. ]
 [6.4 2.8 5.6 2.2]
 [6.3 2.8 5.1 1.5]
 [6.1 2.6 5.6 1.4]
 [7.7 3.  6.1 2.3]
 [6.3 3.4 5.6 2.4]
 [6.4 3.1 5.5 1.8]
 [6.  3.  4.8 1.8]
 [6.9 3.1 5.4 2.1]
 [6.7 3.1 5.6 2.4]
 [6.9 3.1 5.1 2.3]
 [5.8 2.7 5.1 1.9]
 [6.8 3.2 5.9 2.3]
 [6.7 3.3 5.7 2.5]
 [6.7 3.  5.2 2.3]
 [6.3 2.5 5.  1.9]
 [6.5 3.  5.2 2. ]
 [6.2 3.4 5.4 2.3]
 [5.9 3.  5.1 1.8]]
```

```
In [106…
print('The names of the dataset variables:\n',iris['feature_names'])
```

```
The names of the dataset variables:
 ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
```

```
In [106…
print("Name of classes: \n", list(iris.target_names))
```

```
Name of classes:
 ['setosa', 'versicolor', 'virginica']
```

B. Data normalization

The sklearn.preprocessing package provides several common utility functions
and transformer classes to change raw feature vectors into a representation that is more suitable for the downstream estimators. Standardization of datasets is a common requirement for many machine learning estimators implemented in the scikit: they might behave badly if the individual feature do not more or less look like standard normally distributed data: Gaussian with zero mean and unit variance. In practice we often ignore the shape of the distribution and just transform the data to center it by removing the mean value of each feature, then scale it by dividing non-constant features by their standard deviation. For instance, many elements used in the objective function of a learning algorithm (such as the RBF kernel of Support Vector Machines or the l1 and l2 regularizes of linear models) assume that all features are centered around zero and have variance in the same order. If a feature has a variance that is orders of magnitude larger those others, it might dominate the objective function and make the estimator unable to learn from other features correctly as expected.

```
In [106…
from sklearn.preprocessing import scale
```

1- Create the following matrix X: 1, -1, 2, 2, 0, 0, 0, 1, -1

```
In [106…
X=[[1,-1,2],[2,0,0], [0,1,-1]]
```

2- Print the matrix and compute the mean of the variables.

```
In [106…
print(X)
M = np.mean(X)
print("The mean of the variables is:  {:.2f}".format(M))
```

```
[[1, -1, 2], [2, 0, 0], [0, 1, -1]]
The mean of the variables is:  0.44
```

3- Use the scale function to normalize X. Analyze the result.

```
In [106…
scaled = scale( X )
scaled
```

```
Out[106…
array([[ 0.        , -1.22474487,  1.33630621],
       [ 1.22474487,  0.        , -0.26726124],
       [-1.22474487,  1.22474487, -1.06904497]])
```

4- Compute the mean and the variance of the scaled X. What can you conclude?

```
In [107…
mean = np.mean(scaled)
variance = np.var(scaled)
print("The mean of the scaled matrix is: {:.2f}".format(mean))
print("The variance of the scaled matrix is: {:.2f}".format(variance))
print("Mean = {}".format(np.mean(scaled, axis=0)))
print("Variance = {}".format(np.var(scaled, axis=0)))
```

```
The mean of the scaled matrix is: 0.00
The variance of the scaled matrix is: 1.00
Mean = [0. 0. 0.]
Variance = [1. 1. 1.]
```

We have scaled our matrix so that all 3 features could be in the same scaling. We verified this by cheking the mean of the matrix and variance and we can observe that the mean is 0 and variance is 1, so they make our data unitless. This is because we have used the Standardization type for scaling, but there also is Normalization that is used when we want to bound our values between two numbers, typically, between [0,1] or [-1,1]. Machine learning algorithm just sees number — if there is a vast difference in the range say few ranging in thousands and few ranging in the tens, and it makes the underlying assumption that higher ranging numbers have superiority of some sort. So these more significant number starts playing a more decisive role while training the model.

C. MinMax Normalization

An alternative standardization is scaling features to lie between a given minimum and maximum value, often between zero and one. This can be achieved using MinMaxScaler.

1- Create the following matrix X2: 1, -1, 2, 2, 0, 0, 0, 1, -1

```
In [107…
X2=[[1,-1,2],[2,0,0], [0,1,-1]]
```

2- Print the matrix and compute the mean of the variables.

```
In [107…
print(X2)
M = np.mean(X2)
print("The mean of the variables is: {:.2f}".format(M))
```

```
[[1, -1, 2], [2, 0, 0], [0, 1, -1]]
The mean of the variables is: 0.44
```

3- Normalize the data using MinMaxScaler. Print the scaled matrix and compute the mean and the variance. What can you conclude?

```
In [107…
from sklearn.preprocessing import MinMaxScaler
print(X2)
# define min max scaler
scaler = MinMaxScaler()
# transform data
scaled = scaler.fit_transform(X2)
print(scaled)
```

```
[[1, -1, 2], [2, 0, 0], [0, 1, -1]]
[[0.5        0.         1.        ]
 [1.         0.5        0.33333333]
 [0.         1.         0.        ]]
```

```
In [107…
mean = np.mean(scaled)
variance = np.var(scaled)
print("The mean of the scaled matrix is: {:.2f}".format(mean))
print("The variance of the scaled matrix is: {:.2f}".format(variance))
```

```
The mean of the scaled matrix is: 0.48
The variance of the scaled matrix is: 0.17
```

MinMaxScaler transforms features by scaling each feature to a given range. This estimator scales and translates each feature individually such that it is in the given range on the training set, e.g., between zero and one. This Scaler shrinks the data within the range of -1 to 1 if there are negative values. This is why the variance of the scaled matrix is so small and mean is not 0, but 0.48.

D. Data visualization

1- Import the Iris dataset using : iris = datasets.load_iris()

This data sets consists of 3 different types of irises' (Setosa, Versicolour, and Virginica) petal and sepal length, stored in a 150x4 numpy.ndarray

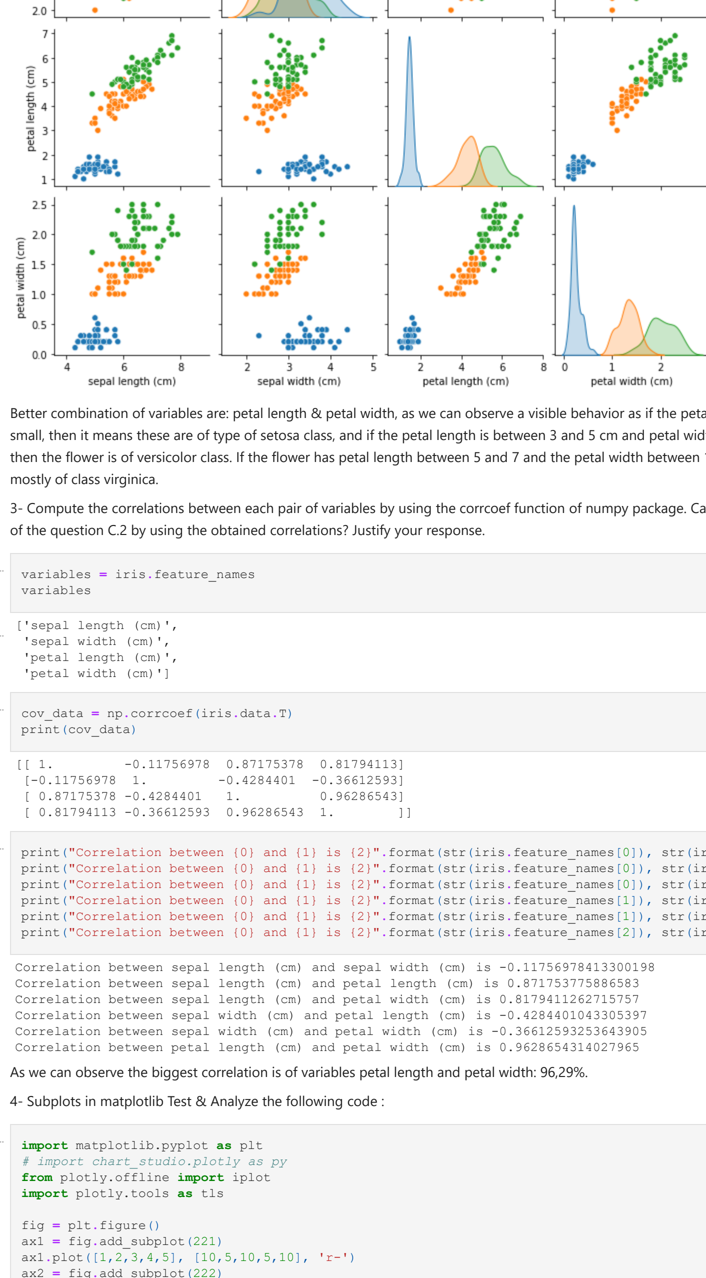
The rows being the samples and the columns being: Sepal Length, Sepal Width, Petal Length and Petal Width. More information about this dataset can be found here: https://archive.ics.uci.edu/ml/datasets/iris The variable iris is an object in Phyton which contains the matrix of data (iris.data), the corresponding label (target), the names of the variables (feature_names) and the name of classes (target_names).

```
In [107…
iris    =       datasets.load_iris()
```

2- Plot the data points into 2D dimension with all the possible combination between variables and use the label for the color points. Vizually, which is the better combination of variables? Justify the answer.

```
In [107…
import pandas as pd
import plotly.express as px
df = pd.DataFrame(data=iris.data, columns=iris.feature_names)

target_map = {i: iris.target_names[i] for i in range(0, len(iris.target_names))}

df['species'] = pd.Series(iris.target).map(target_map)
```

```
In [107…
import seaborn as sns
df['species'] = pd.Series(iris.target).map(target_map)

g = sns.pairplot(df,hue="species")
```



Better combination of variables are: petal length & petal width, as we can observe a visible behavior as if the petal width and length are small, then it means these are type of setosa class, and if the petal length is between 3 and 5 cm and petal width is between 1 and 2 cm then the flower is of versicolor class. If the flower has petal length between 5 and 7 and the petal width between 1.2 and 2.8 cm then is is mostly of class virginica.

3- Compute the correlations between each pair of variables by using the corrcoef function of numpy package. Can you validate the answer of the question C.2 by using the obtained correlations? Justify your response.

```
In [107…
variables = iris.feature_names
variables
```

```
Out[107…
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

```
In [107…
cov_data = np.corrcoef(iris.data.T)
print(cov_data)
```

```
[[ 1.         -0.11756978  0.87175378  0.81794113]
 [-0.11756978  1.         -0.4284401  -0.36612593]
 [ 0.87175378 -0.4284401   1.          0.96286543]
 [ 0.81794113 -0.36612593  0.96286543  1.        ]]
```

```
In [108…
print("Correlation between {0} and {1} is {2}".format(str(iris.feature_names[0]), str(iris.feature_names[1]), r
print("Correlation between {0} and {1} is {2}".format(str(iris.feature_names[0]), str(iris.feature_names[2]),np
print("Correlation between {0} and {1} is {2}".format(str(iris.feature_names[0]), str(iris.feature_names[3]),np
print("Correlation between {0} and {1} is {2}".format(str(iris.feature_names[1]), str(iris.feature_names[2]),np
print("Correlation between {0} and {1} is {2}".format(str(iris.feature_names[1]), str(iris.feature_names[3]),np
print("Correlation between {0} and {1} is {2}".format(str(iris.feature_names[2]), str(iris.feature_names[3]),np
```

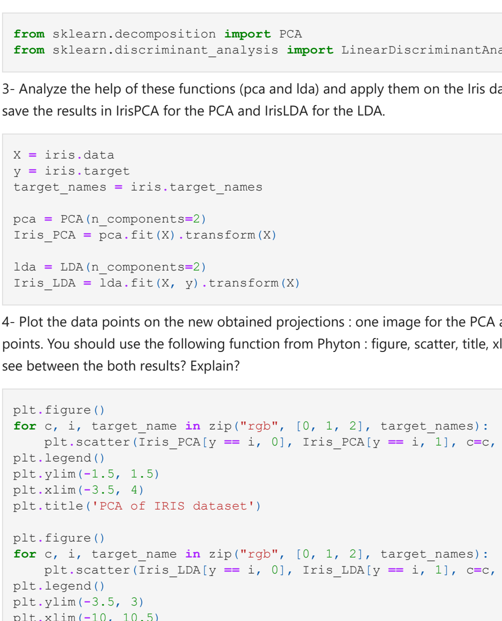
```
Correlation between sepal length (cm) and sepal width (cm) is -0.11756978413300198
Correlation between sepal length (cm) and petal length (cm) is 0.871753775886583
Correlation between sepal length (cm) and petal width (cm) is 0.8179411262715757
Correlation between sepal width (cm) and petal length (cm) is -0.4284401043305397
Correlation between sepal width (cm) and petal width (cm) is -0.36612593253643905
Correlation between petal length (cm) and petal width (cm) is 0.9628654314027965
```

As we can observe the biggest correlation is of variables petal length and petal width: 96,29%.

4- Subplots in matplotlib Test & Analyze the following code :

```
In [108…
import matplotlib.pyplot as plt
# import chart_studio.plotly as py
from plotly.offline import iplot
import plotly.tools as tls

fig = plt.figure()
ax1 = fig.add_subplot(221)
ax1.plot([1,2,3,4,5], [10,5,10,5,10], 'r-')
ax2 = fig.add_subplot(222)
ax2.plot([1,2,3,4], [1,4,9,16], 'k-')
ax3 = fig.add_subplot(223)
ax3.plot([1,2,3,4], [1,10,100,1000], 'b-')
ax4 = fig.add_subplot(224)
ax4.plot([1,2,3,4], [0,0,1,1], 'g-')
plt.tight_layout()
fig = plt.gcf()
plotly_fig = tls.mpl_to_plotly( fig )
plotly_fig['layout']['title'] = 'Simple Subplot Example Title'
plotly_fig['layout']['margin'].update({'t':40})
iplot(plotly_fig)
```

```
In [108…
import matplotlib.pyplot as plt

fig = plt.figure()

ax1 = fig.add_subplot(231)
ax1.scatter(iris.data[:, 0], iris.data[:, 1], c=iris.target, cmap=plt.cm.Set1)
ax1.set_xlabel(iris.feature_names[0])
ax1.set_ylabel(iris.feature_names[1])

ax2 = fig.add_subplot(232)
ax2.scatter(iris.data[:, 0], iris.data[:, 2], c=iris.target, cmap=plt.cm.Set1)
ax2.set_xlabel(iris.feature_names[0])
ax2.set_ylabel(iris.feature_names[2])

ax3 = fig.add_subplot(233)
ax3.scatter(iris.data[:, 0], iris.data[:, 3], c=iris.target, cmap=plt.cm.Set1)
ax3.set_xlabel(iris.feature_names[0])
ax3.set_ylabel(iris.feature_names[3])

ax4 = fig.add_subplot(234)
ax4.scatter(iris.data[:, 1], iris.data[:, 2], c=iris.target, cmap=plt.cm.Set1)
ax4.set_xlabel(iris.feature_names[1])
ax4.set_ylabel(iris.feature_names[2])
8
ax4 = fig.add_subplot(235)
ax4.scatter(iris.data[:, 1], iris.data[:, 3], c=iris.target, cmap=plt.cm.Set1)
ax4.set_xlabel(iris.feature_names[1])
ax4.set_ylabel(iris.feature_names[3])

ax4 = fig.add_subplot(236)
ax4.scatter(iris.data[:, 2], iris.data[:, 3], c=iris.target, cmap=plt.cm.Set1)
ax4.set_xlabel(iris.feature_names[2])
ax4.set_ylabel(iris.feature_names[3])

plt.tight_layout()
fig = plt.gcf()
```



E. Data reduction and visualization

1. Use the correlations information's found in D.3 and reduce the dataset to 3 variables then to 2 variables. In the following, we will apply PCA and LDA to visualize the datasets. We are not interested here in the details of these methods, as these approaches will be presented in Data Mining and Predictive Analytics lecture.

- Principal Component Analysis (PCA) applied to this data identifies the combination of attributes (principal components, or directions in the feature space) that account for the most variance in the data. Here we plot the different samples on the 2 first principal components.
- Linear Discriminant Analysis (LDA) tries to identify attributes that account for the most variance between classes. In particular, LDA, in contrast to PCA, is a supervised method, using known class labels.

```
In [108…
iris.feature_names = ['sepal length (cm)', 'petal length (cm)', 'petal width (cm)']
iris.data = iris.data[:, [0, 2, 3]]
```

```
In [108…
print("Correlation between {0} and {1} is {2}".format(str(iris.feature_names[0]), str(iris.feature_names[1]),np
print("Correlation between {0} and {1} is {2}".format(str(iris.feature_names[0]), str(iris.feature_names[2]),np
print("Correlation between {0} and {1} is {2}".format(str(iris.feature_names[1]), str(iris.feature_names[2]),np
```

```
Correlation between sepal length (cm) and petal length (cm) is 0.871753775886583
Correlation between sepal length (cm) and petal width (cm) is 0.8179411262715757
Correlation between petal length (cm) and petal width (cm) is 0.9628654314027965
```

```
In [108…
iris.feature_names = ['petal length (cm)', 'petal width (cm)']
iris.data = iris.data[:, [1, 2]]
```

2- The PCA and LDA methods can be imported from the following packages : from sklearn.decomposition import PCA from sklearn.lda import LDA

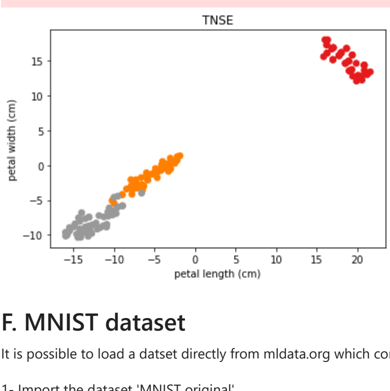
```
In [108…
from sklearn.decomposition import PCA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
```

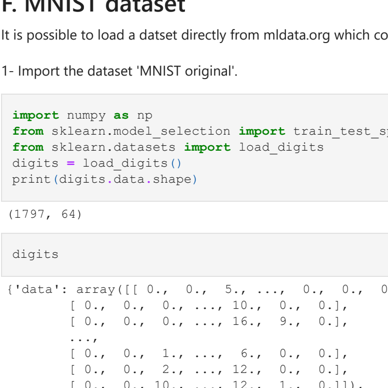
3- Analyze the help of these functions (pca and lda) and apply them on the Iris dataset. You have to use here pca.fit(Iris).transform(Iris) and save the results in IrisPCA for the PCA and IrisLDA for the LDA.

```
In [108…
X = iris.data
y = iris.target
target_names = iris.target_names

pca = PCA(n_components=2)
Iris_PCA = pca.fit(X).transform(X)

lda = LDA(n_components=2)
Iris_LDA = lda.fit(X, y).transform(X)
```

4- Plot the data points on the new obtained projections : one image for the PCA and another for the LDA and use the label as color for the points. You should use the following function from Phyton : figure, scatter, title, xlim, ylim, xlabel, ylabel et show. Which difference you can see between the both results? Explain?

```
In [108…
plt.figure()
for c, i, target_name in zip("rgb", [0, 1, 2], target_names):
    plt.scatter(Iris_PCA[y == i, 0], Iris_PCA[y == i, 1], c=c, label=target_name)
plt.legend()
plt.ylim(-1.5, 1.5)
plt.xlim(-3.5, 4)
plt.title('PCA of IRIS dataset')

plt.figure()
for c, i, target_name in zip("rgb", [0, 1, 2], target_names):
    plt.scatter(Iris_LDA[y == i, 0], Iris_LDA[y == i, 1], c=c, label=target_name)
plt.legend()
plt.ylim(-3.5, 3)
plt.xlim(-10, 10.5)
plt.title('LDA of IRIS dataset')

plt.show()
```

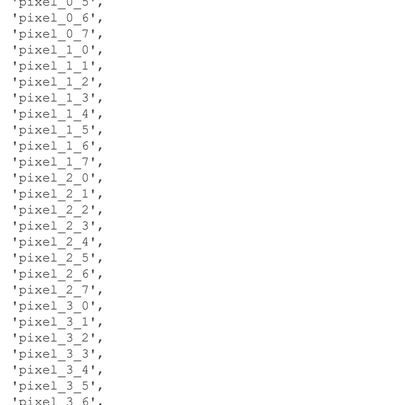




As we know, to use the PCA we must firstly scale the data, but the LDA can handle the data that is not scaled. We can observe that the points have almost the same distribution, but th ranges are different: for PCA the X axis is between -3.2 and 4, and Y axis is between -1.5 and 1.5. And for LDA the X axis is between range -10 and 10 and for Y axis is between -3.2 and 3. We can picture PCA as a technique that finds the directions of maximal variance. In contrast to PCA, LDA attempts to find a feature subspace that maximizes class separability.

```
In [108…
from sklearn.manifold import TSNE

tsne = TSNE(n_components=2)
tsne.fit(iris.data)
iris_tsne = tsne.fit_transform(iris.data)

plt.scatter(iris_tsne[:, 0], iris_tsne[:, 1], c=iris.target, cmap=plt.cm.Set1)
plt.xlabel(iris.feature_names[0])
plt.ylabel(iris.feature_names[1])
plt.title('TNSE')
plt.show()
```

```
C:\Users\User\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\manifold\_t_sne.py:780: FutureWa
rning:

The default initialization in TSNE will change from 'random' to 'pca' in 1.2.

C:\Users\User\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\manifold\_t_sne.py:790: FutureWa
rning:

The default learning rate in TSNE will change from 200.0 to 'auto' in 1.2.

C:\Users\User\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\manifold\_t_sne.py:780: FutureWa
rning:

The default initialization in TSNE will change from 'random' to 'pca' in 1.2.

C:\Users\User\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\manifold\_t_sne.py:790: FutureWa
rning:

The default learning rate in TSNE will change from 200.0 to 'auto' in 1.2.
```



F. MNIST dataset

It is possible to load a datset directly from mldata.org which contains a lot of available datasets using the function datasets.fetch_mldata.

1- Import the dataset 'MNIST original'.

```
In [109…
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_digits
digits = load_digits()
print(digits.data.shape)
```

```
(1797, 64)
```

```
In [109…
digits
```

```
Out[109…
{'data': array([[ 0.,  0.,  5., ...,  0.,  0.,  0.],
        [ 0.,  0.,  0., ..., 10.,  0.,  0.],
        [ 0.,  0.,  0., ..., 16.,  9.,  0.],
        ...,
        [ 0.,  0.,  1., ...,  6.,  0.,  0.],
        [ 0.,  0.,  2., ..., 12.,  0.,  0.],
        [ 0.,  0., 10., ..., 12.,  1.,  0.]]),
 'target': array([0, 1, 2, ..., 8, 9, 8]),
 'frame': None,
 'feature_names': ['pixel_0_0',
  'pixel_0_1',
  'pixel_0_2',
  'pixel_0_3',
  'pixel_0_4',
  'pixel_0_5',
  'pixel_0_6',
  'pixel_0_7',
  'pixel_1_0',
  'pixel_1_1',
  'pixel_1_2',
  'pixel_1_3',
  'pixel_1_4',
  'pixel_1_5',
  'pixel_1_6',
  'pixel_1_7',
  'pixel_2_0',
  'pixel_2_1',
  'pixel_2_2',
  'pixel_2_3',
  'pixel_2_4',
  'pixel_2_5',
  'pixel_2_6',
  'pixel_2_7',
  'pixel_3_0',
  'pixel_3_1',
  'pixel_3_2',
  'pixel_3_3',
  'pixel_3_4',
  'pixel_3_5',
  'pixel_3_6',
  'pixel_3_7',
  'pixel_4_0',
  'pixel_4_1',
  'pixel_4_2',
  'pixel_4_3',
  'pixel_4_4',
  'pixel_4_5',
  'pixel_4_6',
  'pixel_4_7',
  'pixel_5_0',
  'pixel_5_1',
  'pixel_5_2',
  'pixel_5_3',
  'pixel_5_4',
  'pixel_5_5',
  'pixel_5_6',
  'pixel_5_7',
  'pixel_6_0',
  'pixel_6_1',
  'pixel_6_2',
  'pixel_6_3',
  'pixel_6_4',
  'pixel_6_5',
  'pixel_6_6',
  'pixel_6_7',
  'pixel_7_0',
  'pixel_7_1',
  'pixel_7_2',
  'pixel_7_3',
  'pixel_7_4',
  'pixel_7_5',
  'pixel_7_6',
  'pixel_7_7'],
 'target_names': array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]),
 'images': array([[[ 0.,  0.,  5., ...,  1.,  0.,  0.],
         [ 0.,  0., 13., ..., 15.,  5.,  0.],
         [ 0.,  3., 15., ..., 11.,  8.,  0.],
         ...,
         [ 0.,  4., 11., ..., 12.,  7.,  0.],
         [ 0.,  2., 14., ..., 12.,  0.,  0.],
         [ 0.,  0.,  6., ...,  0.,  0.,  0.]],

        [[ 0.,  0.,  0., ...,  5.,  0.,  0.],
         [ 0.,  0.,  0., ...,  9.,  0.,  0.],
         [ 0.,  0.,  3., ...,  6.,  0.,  0.],
         ...,
         [ 0.,  0.,  1., ...,  6.,  0.,  0.],
         [ 0.,  0.,  1., ...,  6.,  0.,  0.],
         [ 0.,  0.,  0., ..., 10.,  0.,  0.]],

        [[ 0.,  0.,  0., ..., 12.,  0.,  0.],
         [ 0.,  0.,  3., ..., 14.,  0.,  0.],
         [ 0.,  0.,  8., ..., 16.,  0.,  0.],
         ...,
         [ 0.,  9., 16., ...,  0.,  0.,  0.],
         [ 0.,  3., 13., ..., 11.,  5.,  0.],
         [ 0.,  0.,  0., ..., 16.,  9.,  0.]],

        ...,

        [[ 0.,  0.,  1., ...,  1.,  0.,  0.],
         [ 0.,  0., 13., ...,  2.,  1.,  0.],
         [ 0.,  0., 16., ..., 16.,  5.,  0.],
         ...,
         [ 0.,  0., 16., ..., 15.,  0.,  0.],
         [ 0.,  0., 15., ..., 16.,  0.,  0.],
         [ 0.,  0.,  2., ...,  6.,  0.,  0.]],

        [[ 0.,  0.,  2., ...,  0.,  0.,  0.],
         [ 0.,  0., 14., ..., 15.,  1.,  0.],
         [ 0.,  4., 16., ..., 16.,  7.,  0.],
         ...,
         [ 0.,  0.,  0., ..., 16.,  2.,  0.],
         [ 0.,  0.,  4., ..., 16.,  2.,  0.],
         [ 0.,  0.,  5., ..., 12.,  0.,  0.]],

        [[ 0.,  0., 10., ...,  1.,  0.,  0.],
         [ 0.,  2., 16., ...,  1.,  0.,  0.],
         [ 0.,  0., 15., ..., 15.,  0.,  0.],
         ...,
         [ 0.,  4., 16., ..., 16.,  6.,  0.],
         [ 0.,  8., 16., ..., 16.,  8.,  0.],
         [ 0.,  1.,  8., ..., 12.,  1.,  0.]]]),
 'DESCR': ".. _digits_dataset:\n\nOptical recognition of handwritten digits dataset\n--------------------------
------------------------\n\n**Data Set Characteristics:**\n\n    :Number of Instances: 1797\n    :Number of Att
ributes: 64\n    :Attribute Information: 8x8 image of integer pixels in the range 0..16.\n    :Missing Attribut
e Values: None\n    :Creator: E. Alpaydin (alpaydin '@' boun.edu.tr)\n    :Date: July; 1998\n\nThis is a copy o
f the test set of the UCI ML hand-written digits datasets\nhttps://archive.ics.uci.edu/ml/datasets/Optical+Reco
gnition+of+Handwritten+Digits\n\nThe data set contains images of hand-written digits: 10 classes where\neach cl
ass refers to a digit.\n\nPreprocessing programs made available by NIST were used to extract\nnormalized bitmap
s of handwritten digits from a preprinted form. From a\ntotal of 43 people, 30 contributed to the training set
and different 13\nto the test set. 32x32 bitmaps are divided into nonoverlapping blocks of\n4x4 and the number
of on pixels are counted in each block. This generates\nan input matrix of 8x8 where each element is an integer
in the range\n0..16. This reduces dimensionality and gives invariance to small\ndistortions.\n\nFor info on NIS
T preprocessing routines, see M. D. Garris, J. L. Blue, G.\nT. Candela, D. L. Dimmick, J. Geist, P. J. Grother,
S. A. Janet, and C.\nL. Wilson, NIST Form-Based Handprint Recognition System, NISTIR 5469,\n1994.\n\n.. topic::
References\n\n  - C. Kaynak (1995) Methods of Combining Multiple Classifiers and Their\n    Applications to Han
dwritten Digit Recognition, MSc Thesis, Institute of\n    Graduate Studies in Science and Engineering, Bogazici
University.\n  - E. Alpaydin, C. Kaynak (1998) Cascading Classifiers, Kybernetika.\n  - Ken Tang and Ponnuthura
i N. Suganthan and Xi Yao and A. Kai Qin.\n    Linear dimensionalityreduction using relevance weighted LDA. Sch
ool of\n    Electrical and Electronic Engineering Nanyang Technological University.\n    2005.\n  - Claudio Gen
tile. A New Approximate Maximal Margin Classification\n    Algorithm. NIPS. 2000.\n"}
```

2- Plot the dataset matrix, the number of data, number of variables and classes.

```
In [109…
print('Dataset matrix')
print(digits.data)
```

```
Dataset matrix
[[ 0.  0.  5. ...  0.  0.  0.]
 [ 0.  0.  0. ... 10.  0.  0.]
 [ 0.  0.  0. ... 16.  9.  0.]
 ...
 [ 0.  0.  1. ...  6.  0.  0.]
 [ 0.  0.  2. ... 12.  0.  0.]
 [ 0.  0. 10. ... 12.  1.  0.]]
```

```
In [109…
print('Number of data')
print(digits.data.shape)
```

```
Number of data
(1797, 64)
```

It means we have 1797 observations and 64 variables.

```
In [109…
print('Variables')
print(digits['feature_names'])
print('Number of variables: {:}'.format(len(digits['feature_names'])))
```

```
Variables
['pixel_0_0', 'pixel_0_1', 'pixel_0_2', 'pixel_0_3', 'pixel_0_4', 'pixel_0_5', 'pixel_0_6', 'pixel_0_7', 'pixel
_1_0', 'pixel_1_1', 'pixel_1_2', 'pixel_1_3', 'pixel_1_4', 'pixel_1_5', 'pixel_1_6', 'pixel_1_7', 'pixel_2_0',
'pixel_2_1', 'pixel_2_2', 'pixel_2_3', 'pixel_2_4', 'pixel_2_5', 'pixel_2_6', 'pixel_2_7', 'pixel_3_0', 'pixel_
3_1', 'pixel_3_2', 'pixel_3_3', 'pixel_3_4', 'pixel_3_5', 'pixel_3_6', 'pixel_3_7', 'pixel_4_0', 'pixel_4_1',
'pixel_4_2', 'pixel_4_3', 'pixel_4_4', 'pixel_4_5', 'pixel_4_6', 'pixel_4_7', 'pixel_5_0', 'pixel_5_1', 'pixel_
5_2', 'pixel_5_3', 'pixel_5_4', 'pixel_5_5', 'pixel_5_6', 'pixel_5_7', 'pixel_6_0', 'pixel_6_1', 'pixel_6_2',
'pixel_6_3', 'pixel_6_4', 'pixel_6_5', 'pixel_6_6', 'pixel_6_7', 'pixel_7_0', 'pixel_7_1', 'pixel_7_2', 'pixel_
7_3', 'pixel_7_4', 'pixel_7_5', 'pixel_7_6', 'pixel_7_7']
Number of variables: 64
```

```
In [109…
print('Classes')
print(digits['target_names'])
print('Number of classes: {:}'.format(len(digits['target_names'])))
```

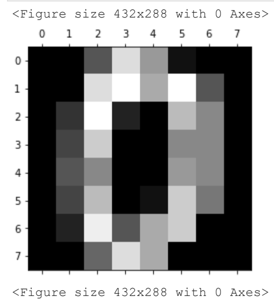
```
Classes
[0 1 2 3 4 5 6 7 8 9]
Number of classes: 10
```

3- Use the MNIST visulaization tutorial to visualize the digits.

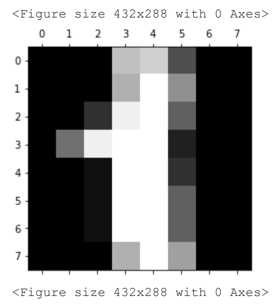
In [109…

```python
import matplotlib.pyplot as plt
for i in range(0,10):
    plt.gray()
    plt.matshow(digits.images[i])
    plt.show()
```

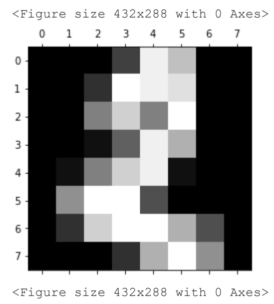
<Figure size 432x288 with 0 Axes>



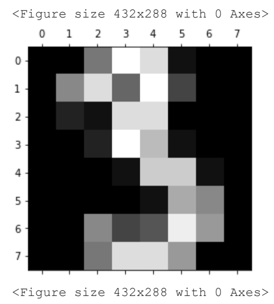
<Figure size 432x288 with 0 Axes>



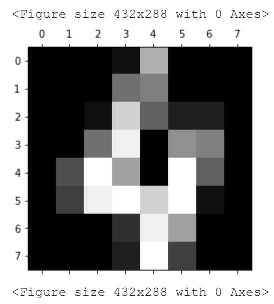
<Figure size 432x288 with 0 Axes>



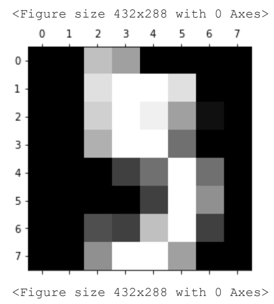
<Figure size 432x288 with 0 Axes>



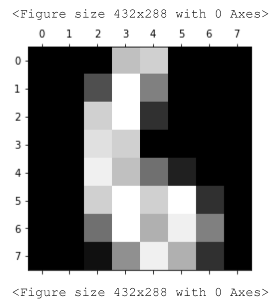
<Figure size 432x288 with 0 Axes>



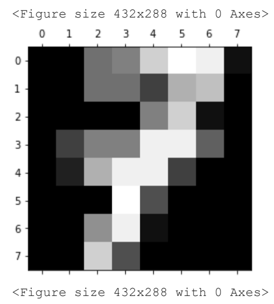
<Figure size 432x288 with 0 Axes>



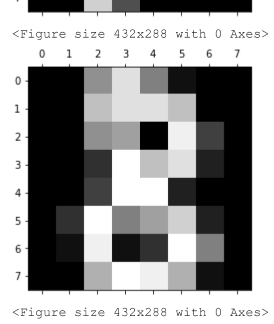
<Figure size 432x288 with 0 Axes>



<Figure size 432x288 with 0 Axes>



<Figure size 432x288 with 0 Axes>



<Figure size 432x288 with 0 Axes>