1_Basics_students

July 13, 2019

1 HaMLeT

2 Session 1: Basics

2.1 Goals of this session

After this session, you will have an understanding of: - basic python programming - python datastructures - some important python packages: `NumPy`, `matplotlib.pyplot`, `scikit-learn` - working with images - some common terms used in ML - the Iris dataset - PCA

2.1.1 Structure of the Notebook:

You need to perform the tasks and answer the questions in the notebook. For most of the tasks you will find either examples or hints to help you. For this notebook the hints are mostly in the form of commands that you will be required to use. Feel free to search the internet and find out how to use the required command, eg. which inputs need to provided and how many outputs are to be expected, etc.

Let's start with some basic python programming. Have fun!

Python requires you to `import` the package you want to work on. Let's `import` an important package that motivates us to learn python!

```
In [3]: # importing necessary package
        import os
        import sys

        #python3

        os.environ["CUDA_VISIBLE_DEVICES"]="2"

        import antigravity
```

2.2 1. Basic `NumPy` operations

```
In [4]: # importing necessary packages
        import numpy as np
        #import gym
```

Task: Create a 1D array with 10 elements Hint: you may use any of `NumPy` built-in array generators, eg. `np.zeros`, `np.ones`, `np.arange`, `np.random`, etc.

```
In [5]: # Your code here:
        int_array1 = np.random.randint(1,10,10)

        print(int_array1)
```

```
[6 2 3 3 8 3 9 3 8 8]
```

Task: Reshape the above array (`np.reshape`)

```
In [6]: # Your code here:
        int_array1.reshape(2,5)
```

```
Out[6]: array([[6, 2, 3, 3, 8],
               [3, 9, 3, 8, 8]])
```

Task: Create another 1D array and combine it with the previous array (a) vertically (b) horizontally (eg. using `np.concatenate`, `np.vstack`, etc.)

```
In [7]: # Your code here:
        int_array2 = np.random.randint(1,20,10)

        print(int_array2)

        #Concatenate
        int_array3 = np.vstack((int_array1,int_array1))
        print("vertical concat",int_array3)

        int_array4 = np.hstack((int_array1,int_array2))
        print("horizontal concat",int_array4)
```

```
[19 11  7 14 11 10  3 13 19 14]
vertical concat [[6 2 3 3 8 3 9 3 8 8]
 [6 2 3 3 8 3 9 3 8 8]]
horizontal concat [ 6  2  3  3  8  3  9  3  8  8 19 11  7 14 11 10  3 13 19 14]
```

Task: Get positions where first and second array have the same elements (eg. `np.where`)

```
In [8]: # Your code here:
        A_inds = np.where(int_array1==int_array2)
        print(A_inds)
```

```
(array([], dtype=int64),)
```

Task: Extract the following from the matrix you created by horizontal concatenation: (a) the first element of the matrix, (b) the first row, (c) the first column, (d) any subset of your choice, eg.the four center elements.

This is called 'Slicing'

```
In [9]: # Your code here:
        m = np.asmatrix(int_array4)
        m.shape
        print("a=",m[0,0])
        print("\n")
        print("b=",m[0,:])
        print("\n")
        print("c=",m[:,0])
        print("d=",m[0,8:12])
```

```
a= 6


b= [[ 6  2  3  3  8  3  9  3  8  8 19 11  7 14 11 10  3 13 19 14]]


c= [[6]]
d= [[ 8  8 19 11]]
```

2.3 2. Python data types

- numbers (eg. int, float32)
- booleans (True, False)
- 'null' type (None)
- strings (eg. 'hello', 'world')
- lists (eg. [8, 'hello', 5>7])
- tuples (eg. (8,'hello',5>7))

Hopefully you are already familiar with the common datatypes. We are going to discuss the last two:

2.3.1 List

Python lists allow you to store a sequence of different objects, as in the above example, [int, string, bool].

Note: Lists are created using sqaure brackets [ ] and comma separated values

Tasks on List 1. Create a list containing a string, a bool, and an int 2. Change any one object of your choice from the list 3. Add new objects to the end of the existing list using (a) `append` (b) `extend`.

```
In [10]: # Your code here:
         list1 = ["abc",8,25.4,True]
         print(list1)
         list1[1]=4
         print("after change",list1)
         list1.append(5678)
```

```
        print("after append",list1)
        list1.extend([3,4,5])
        print("after extend",list1)
```

```
['abc', 8, 25.4, True]
after change ['abc', 4, 25.4, True]
after append ['abc', 4, 25.4, True, 5678]
after extend ['abc', 4, 25.4, True, 5678, 3, 4, 5]
```

Question: What is the difference between `append` and `extend`?
Answer:
APPEND : append adds an item to list EXTEND : extends cureent list by another list
Tasks on List (cont.) 5. Try out the examples using the functions: `pop`, `remove`, and `del`

```
In [11]: myList = [0,5,3,4,3]
         ans=myList.pop(3)
         print("pop=",ans)
         print(myList)

         myList = [0,5,3,4,3]
         myList.remove(3)
         print("After remove",myList)


         myList = [0,5,3,4,3]
         del myList[3]
         print("After delete",myList)
```

```
pop= 4
[0, 5, 3, 3]
After remove [0, 5, 4, 3]
After delete [0, 5, 3, 3]
```

Question: What is the difference among functions from task 5?
Answer: pop : returns the value at that index and removes from list remove : Removes the first found item in the list(left to right) del : delete value at specific index

2.3.2 Tuple

Similar to list, tuples can also store a sequence of arbitrary objects. They are constructed using parentheses.

Important: List can be changed after construction (mutable). Tuple cannot be changed (immutable).

Tasks on Tuple 1. Create a tuple object and replace the first object with the int object 0 2. Convert the above tuple to list, change the first object, convert back to tuple

Hint: Use functions `list()` and `tuple()` to convert datatype Tip: You can check datatype using either `type()` or `isinstance(var,dtype)`

```
In [12]: # Your code here:

         tup = (3,4,5,6)
         print(tup)
         #tup(0) = 5
         listNumbers = list(tup)
         print(listNumbers)
         listNumbers[0] = 5
         print(listNumbers)

         tup = tuple(listNumbers)
         isinstance(tup,tuple)

         print(tup)
```

```
(3, 4, 5, 6)
[3, 4, 5, 6]
[5, 4, 5, 6]
(5, 4, 5, 6)
```

2.3.3 Sequence types

- list
- string
- tuple
- NumPy array

Task on sequence types (handy when programming in python!)

- Checking if object is contained within sequence or not.

Understand the following example code and then perform the given tasks.

```
In [13]: # Example code:
         x = (1,2,3)
         y1 = 3 in x
         y2 = 5 not in x
         print('y1 =',y1)
         print('y2 =',y2)
```

```
y1 = True
y2 = True
```

Task: Perform the tasks according to the instructions in the following three cells and answer the question at the end

```
In [14]: # Given:
         task1 = 'let us learn python'
         # Task: Check if 'us' is present in task1 or not
         # Your code here:

         y1 = 'us' in task1
         print('y1 =',y1)
```

```
y1 = True
```

```
In [15]: # Given:
         task2 = [10,20,30,40]
         # Task: Check if [10,20] is present in task2 or not
         # Your code here:
         y1 = [10,20] in task2
         print('y1 =',y1)
```

```
y1 = False
```

```
In [16]: # Given:
         task3 = [True,[2,3],'hello']
         # Task: Check if [2,3] is present in task3 or not
         # Your code here:
         y1 = [2,3] in task3
         print('y1 =',y1)
```

```
y1 = True
```

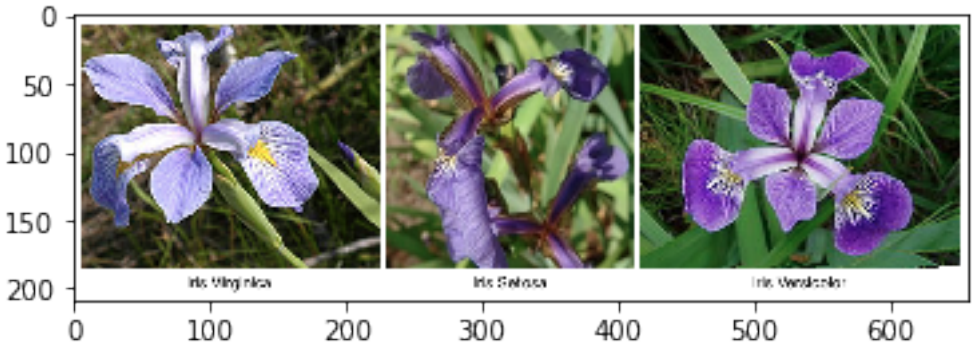
Question: Explain the different outputs in task 2 and task 3 above.
Answer:
Task 3 : [2,3] was treated as a single element

2.4 3. Working with images

We will learn to load, display and manipulate images using the `matplotlib` package.

```
In [17]: # importing necessary packages
         %matplotlib inline
         import matplotlib.pyplot as plt
         import cv2
```

```
In [18]: # Load image
         irisImg = cv2.imread('irisImage.png')
         irisImg = cv2.cvtColor(irisImg, cv2.COLOR_BGR2RGB) # because cv2 reads images as bgr!
         # Display  image
         plt.imshow(irisImg) # display in RGB
         plt.show()
         irisImg.shape
```



Out[18]: (209, 658, 3)

Task: Find out the datatype of the image. Hint: You've already used this command above!

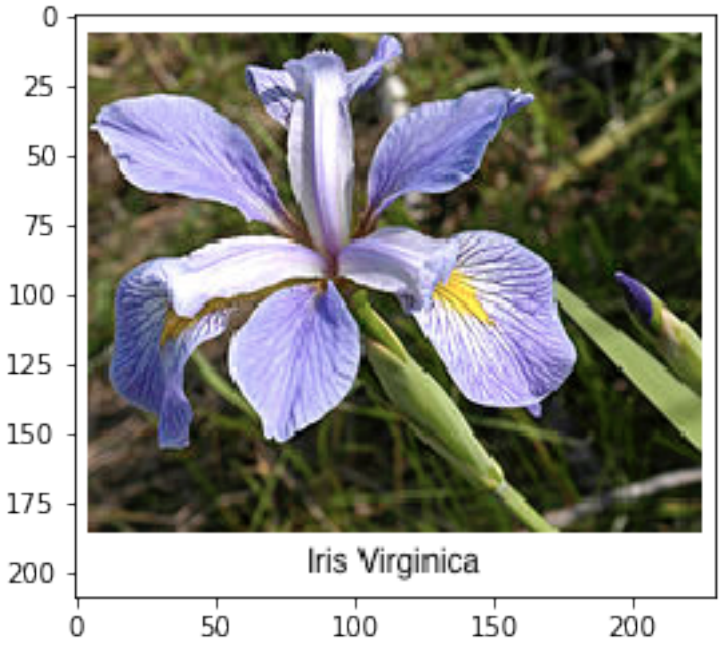
```
In [19]: # Your code here:
         irisImg.dtype
```

Out[19]: dtype('uint8')

2.4.1 Let's use the NumPy manipulations we learned to manipulate the image

Task: Extract and display only one of the species Hint: You can do so by the array slicing method you learned above

```
In [20]: # Your code here:
         # Please name the image 'iris1'
         iris1 = irisImg[:,:230]
         plt.imshow(iris1) # display in RGB
         plt.show()
```



Task: Delete the white border from all the sides of the extracted image

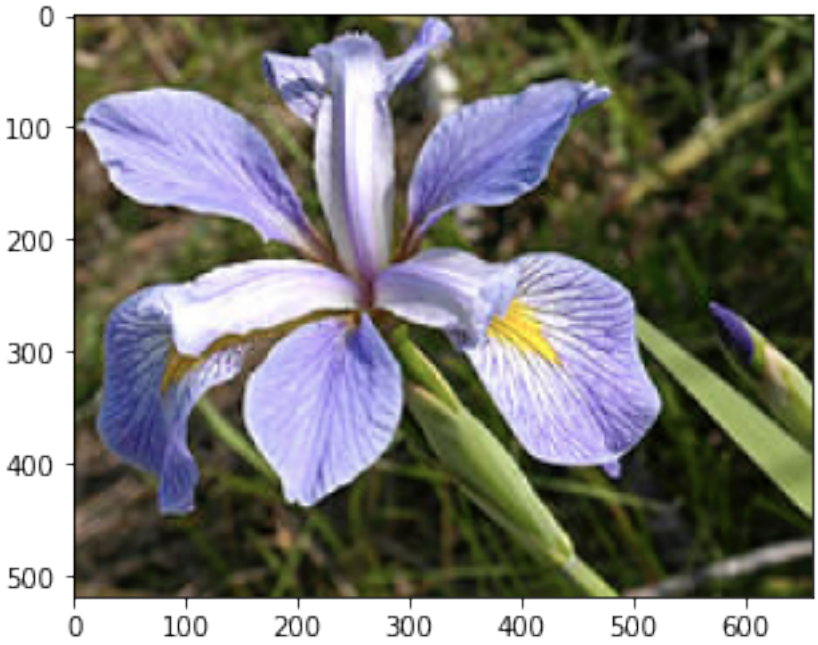
```
In [21]: # Your code here:
         # You may use plt.axis('off') to display images without the axis
         img_cropped = iris1[7:180, 5:225]
         plt.imshow(img_cropped)
         plt.axis('off')

Out[21]: (-0.5, 219.5, 172.5, -0.5)
```

2.4.2 Some useful image manipulations

Task: Resize the image by three times (use `cv2.resize`)

```
In [22]: # Your code here:
         # Please name the image 'resizedImg'
         val=img_cropped.shape
         print(val)
         imgResized = cv2.resize(img_cropped, (val[1]*3, val[0]*3))                    # Resiz
         plt.imshow(imgResized)
         plt.show()
```

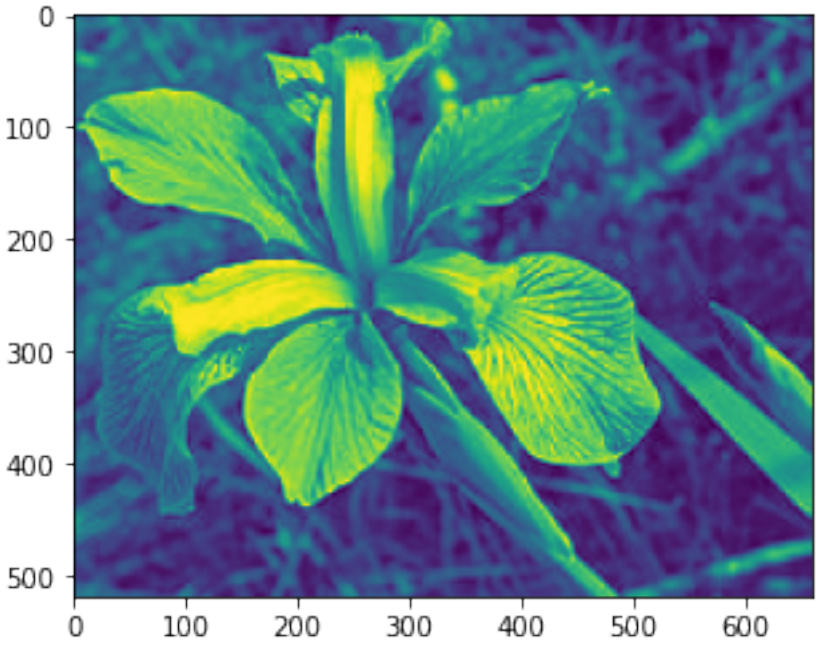
```
(173, 220, 3)
```



Task: Convert the image to grayscale. This can be done by taking the sum of weighted RGB values as follows: 0.2989*R* + *0.5870*G + 0.1140*B

Hint: Use slicing to extract individual channels

```
In [23]: # Your code here:
         # Please name your image 'irisGray'

         irisImgR =imgResized[:,:,0]
         irisImgG =imgResized[:,:,1]
         irisImgB =imgResized[:,:,2]

         irisGrey =  0.2989 * irisImgR + 0.5870 * irisImgG + 0.1140 * irisImgB
         plt.imshow(irisGrey)
         plt.show()
```



Alternatively, we could use the built-in function `cv2.COLOR_RGB2GRAY`

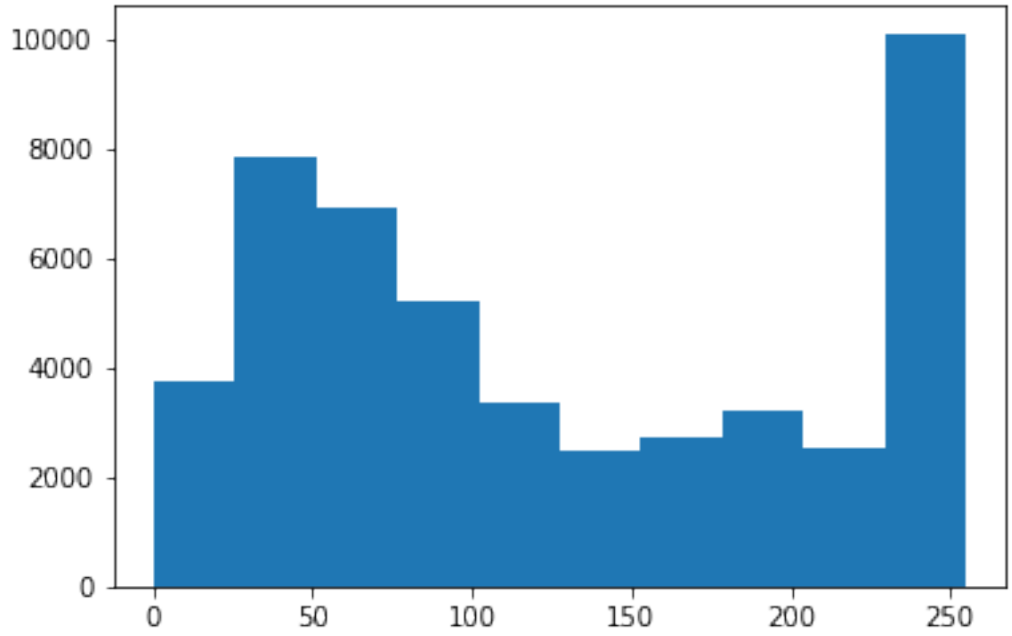
```
In [24]: irisGray = cv2.cvtColor(iris1, cv2.COLOR_RGB2GRAY)
         plt.imshow(irisGray, cmap='gray')
         plt.axis('off')
         plt.show()
```

Iris Virginica

Task: Plot the histogram of the gray scale image (use `plt.hist()` with `ravel()`)

```
In [25]: # Your code here:
         iris_flatten = irisGray.ravel()
         plt.hist(iris_flatten)
```

```
Out[25]: (array([  3733.,   7826.,   6900.,   5201.,   3342.,   2485.,   2741.,
                   3217.,   2529.,  10096.]),
          array([   0. ,   25.5,   51. ,   76.5,  102. ,  127.5,  153. ,  178.5,
                  204. ,  229.5,  255. ]),
          <a list of 10 Patch objects>)
```



2.4.3 Normalization

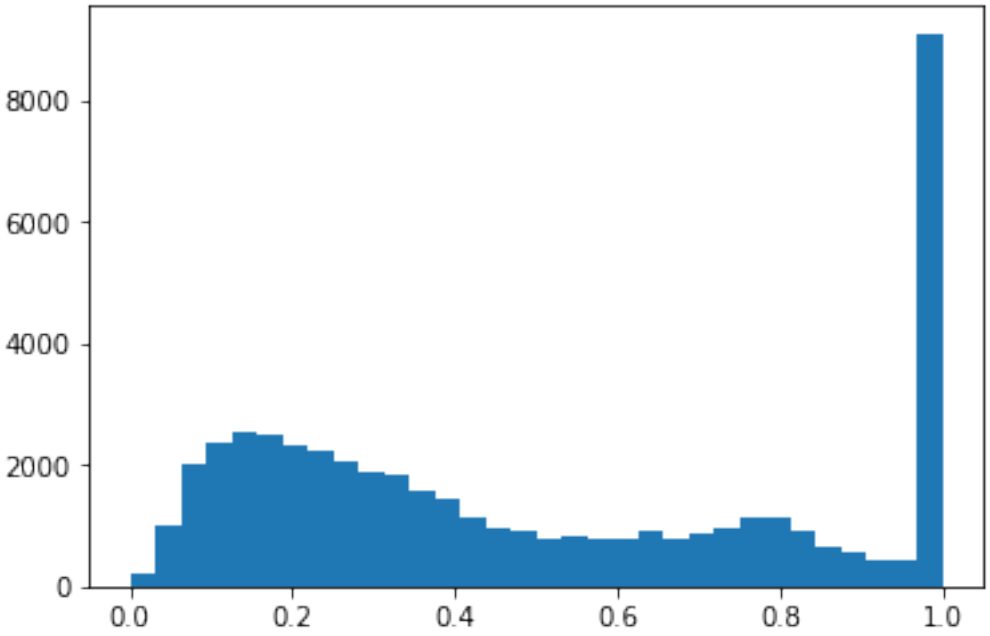
In Machine Learning tasks, normalization is often the first step. It essentially means scaling and centering the data values for faster convergence and improved accuracy. Here we show one method of normalizing by subtracting the minimum value from all data points and then dividing by the range of values.

2.4.4 Normalization method 1:

```
In [26]: # Normalization method 1:
         irisNorm1 = (irisGray-np.min(irisGray))/(np.max(irisGray)-np.min(irisGray))
         plt.imshow(irisNorm1, cmap='gray')
```

```
plt.axis('off')
plt.show()
# To see the histogram of the normalized image
plt.hist(irisNorm1.ravel(),32)
plt.show()
```

Iris Virginica



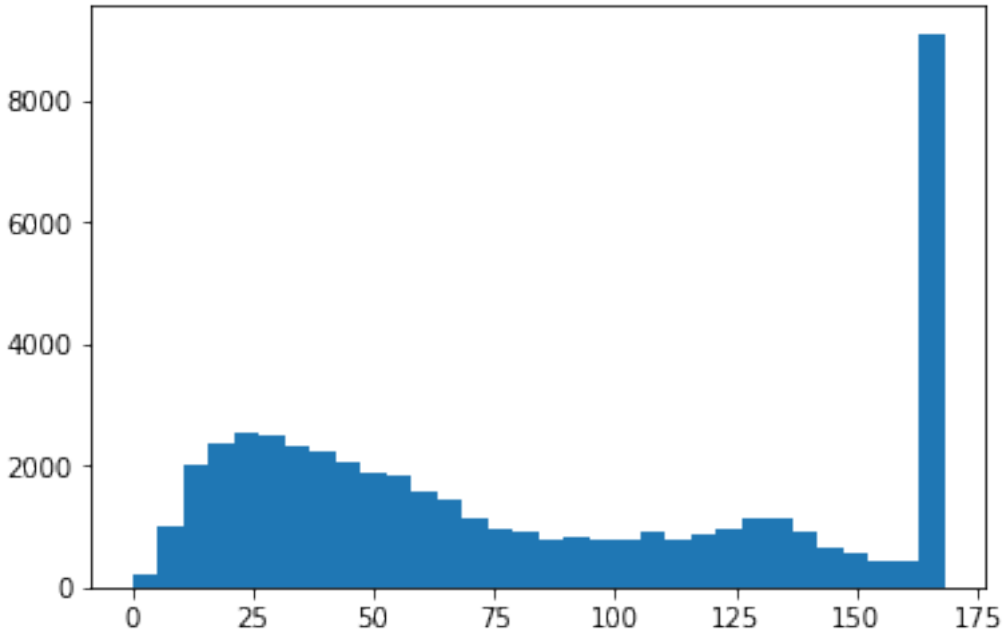
Note that after normalization the image is rescaled between 0 and 1.

2.4.5 Normalization method 2:

Task: Perform normalization on the grayscale image by subtracting the mean and scaling by the standard deviation

```
In [27]: # Normalization method 2:
         # Your code here:
         # Please name your image 'irisNorm2'
         mean = np.mean(irisGray)
         sd   = np.std(irisGray)

         irisNorm2 = (irisGray * sd)/ mean
         plt.imshow(irisNorm2, cmap='gray')
         plt.axis('off')
         plt.show()
         # To see the histogram of the normalized image
         plt.hist(irisNorm2.ravel(),32)
         plt.show()
```

Iris Virginica



Now we're a little comfortable with python and handling different kinds of data. We now move on to the next step!

2.5 4. Machine Learning

In ML, we try to create a model based on the data we have at hand. So the first most important thing is to learn to represent that data such that the computer understands it. In our course, we will use python's Scikit-Learn package.

2.5.1 The dataset

The Iris flower dataset (or Fisher's or Anderson's Iris data set) is a multivariate data set introduced by the British statistician and biologist Ronald Fisher in his 1936 paper. The aim was to quantify the morphologic variation of Iris flowers of three related species (the three flowers we saw above!).

- The data set consists of 50 samples from each of three species of Iris (Iris setosa, Iris virginica and Iris versicolor).
- Four features were measured from each sample: the length and the width of the sepals and petals, in centimetres.

Based on the combination of these four features, Fisher developed a linear discriminant model to distinguish the species from each other.

We will analyze and visualize the data using `seaborn`. It is a handy python package for visualising statistic data and it offers a plethora of nice plotting tools. It also has the aforementioned Iris dataset included. The Iris dataset in seaborn does not use `numpy` or `cv2`, as we don't work on the images directly. Instead, we work with 'features,' as mentioned above. This information is stored in a so called `pandas dataframe`, another useful tool for statistical analyses.

```
In [28]: # We will now load this dataset and display the first few elements.
         # importing necessary packages
         import seaborn as sns
         iris = sns.load_dataset('iris')
         iris.head() #displaying the data (partially)
```

Out[28]:

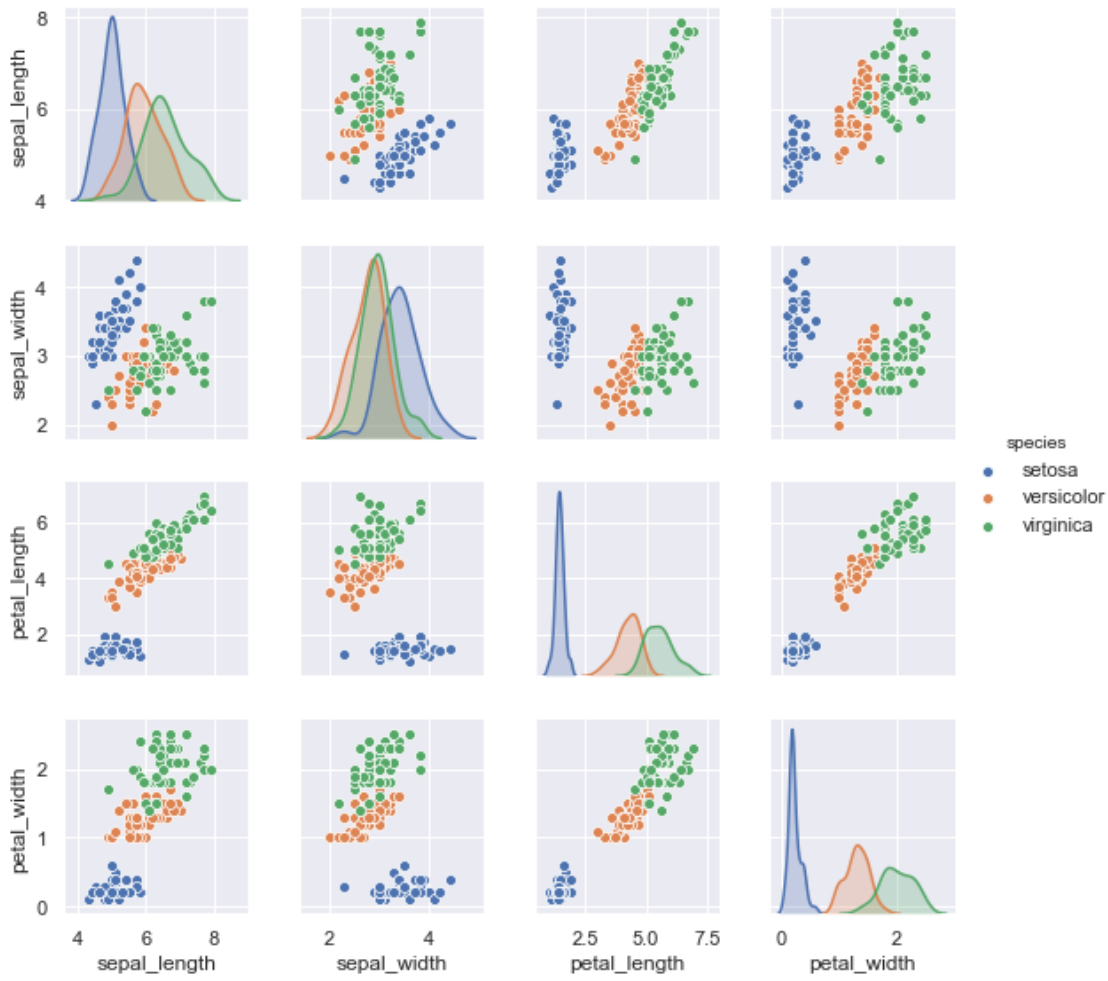
	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

Note: pandas dataframe displays the data in a very intuitive tabular format. Here we see the four features our data has as individual columns. The species on the last column are formed by all the features from the corresponding rows.

2.5.2 Data visualization

```
In [29]: %matplotlib inline
         sns.set()
         sns.pairplot(iris, hue='species',height=2)
```

```
Out[29]: <seaborn.axisgrid.PairGrid at 0x7fe5e0050fd0>
```



Note: Here we see a visual pair-wise comparison of all the features

2.5.3 Working with our dataset using `sklearn`

We will now load the data using the Python ML library `sklearn`. It includes many ML algorithms, some of which we will be learning in this course.

```
In [30]: # importing necessary packages
         from sklearn import datasets
         iris = datasets.load_iris()

In [31]: # Displaying the dataset
         print(iris.data)
         # Displaying the datatype
         print(type(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  0.1]
[ 5.   3.2  1.2  0.2]
[ 5.5  3.5  1.3  0.2]
[ 4.9  3.1  1.5  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]]
<class 'numpy.ndarray'>
```

Note: 1. The dataset was loaded as a numpy array 2. We do not see the species information here

In `sklearn`, the dataset can be separated into 'features' and 'targets' as follows (`sklearn` stores species information as targets)

```
In [32]: features = iris.data[:,[0,1,2,3]]
         print(features.shape)
         targets = iris.target
         print(targets)
```

```
(150, 4)
[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 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 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 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

Task: Extract all the features of the first flower species Hint: You will find some more information when you inspect the shape of 'features' (and 'targets')

```
In [33]: # Your code here:
         index = []
         for i in range(len(targets)):
             if(targets[i]==0):
                 index.append(i)

         print(" index of the first flower species ",index)
         print(" features of the first flower species ",iris.data[index,0])
```

```
 index of the first flower species  [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,
 features of the first flower species  [ 5.1  4.9  4.7  4.6  5.   5.4  4.6  5.   4.4  4.9  5.4
  5.7  5.4  5.1  5.7  5.1  5.4  5.1  4.6  5.1  4.8  5.   5.   5.2  5.2  4.7
  4.8  5.4  5.2  5.5  4.9  5.   5.5  4.9  4.4  5.1  5.   4.5  4.4  5.   5.1
  4.8  5.1  4.6  5.3  5. ]
```

Now let's try to visualize the relationship between some of the features

```
In [34]: # Plotting the relationship between the Sepal Length and Sepal Width
         feature1 = features[:, 0] # Sepal Length
         feature2 = features[:, 1] # Sepal Width
```

```
species = ('Iris-setosa', 'Iris-versicolor', 'Iris-virginica')
colors = ('blue','green','red')

data = [[features[np.where(targets == target)][:, feature] for feature in [0, 1]] for

for item, color, group in zip(data, colors, species):
    plt.scatter(item[0], item[1], color=color)
    plt.title('Iris dataset scatter plot')
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.show()
```

Task: In a similar manner, plot petal length versus petal width

In [35]:
```
# Your code here:

feature3 = features[:, 2] # Sepal Length
feature4 = features[:, 3] # Sepal Width

species = ('Iris-setosa', 'Iris-versicolor', 'Iris-virginica')
colors = ('blue','green','red')

data = [[features[np.where(targets == target)][:, feature] for feature in [2, 3]] for
```

```
for item, color, group in zip(data, colors, species):
    plt.scatter(item[0], item[1], color=color)
    plt.title('Iris dataset scatter plot')
plt.xlabel('Petal length')
plt.ylabel('Petal width')
plt.show()
```

Question: State your findings, eg. do these two features provide a better species separation?
Answer:

We have been comparing only two features at time only because it is easier to visualize. In ML, however, it is quite common to have hundreds of features. In such a case, we might have some features that are redundant by, for eg., being just a linear combination of some other features! Hence, it is common to perform dimensionality reduction to retain only the most representative features from the entire set. One way of doing is by Principal Component Analysis (PCA). Here we try to understand and implement PCA:

2.6 5. Principal Component Analysis (PCA)

2.6.1 PCA Summary:

- Standardize the data.
- Calculate Eigenvectors and Eigenvalues from the covariance matrix.
- Sort eigenvalues in descending order
- Choose the k eigenvectors that correspond to the k largest eigenvalues (k=number of dimensions of the new feature subspace (kd)).

- Construct the projection matrix W from the selected k eigenvectors.
- Transform the original dataset X via W to obtain a k-dimensional feature subspace Y.

```
In [36]: X = features
         Y = targets
         # importing necessary packages for standardizing
         from sklearn.preprocessing import StandardScaler

         from numpy import linalg as LA
         import pandas as pd
         X1 = StandardScaler().fit_transform(X)
```

Task: Calculate the covariance matrix of the **transposed** feature matrix (use `NumPy`'s `cov` function)

```
In [37]: # Your code here:
         # Please name the matrix as 'cov_mat'
         # Remember to transpose the matrix before computing the covariance
         print(X1.shape)

         cov_mat=np.cov(X1.T)

         print(cov_mat)
```

```
(150, 4)
[[ 1.00671141 -0.11010327  0.87760486  0.82344326]
 [-0.11010327  1.00671141 -0.42333835 -0.358937  ]
 [ 0.87760486 -0.42333835  1.00671141  0.96921855]
 [ 0.82344326 -0.358937    0.96921855  1.00671141]]
```

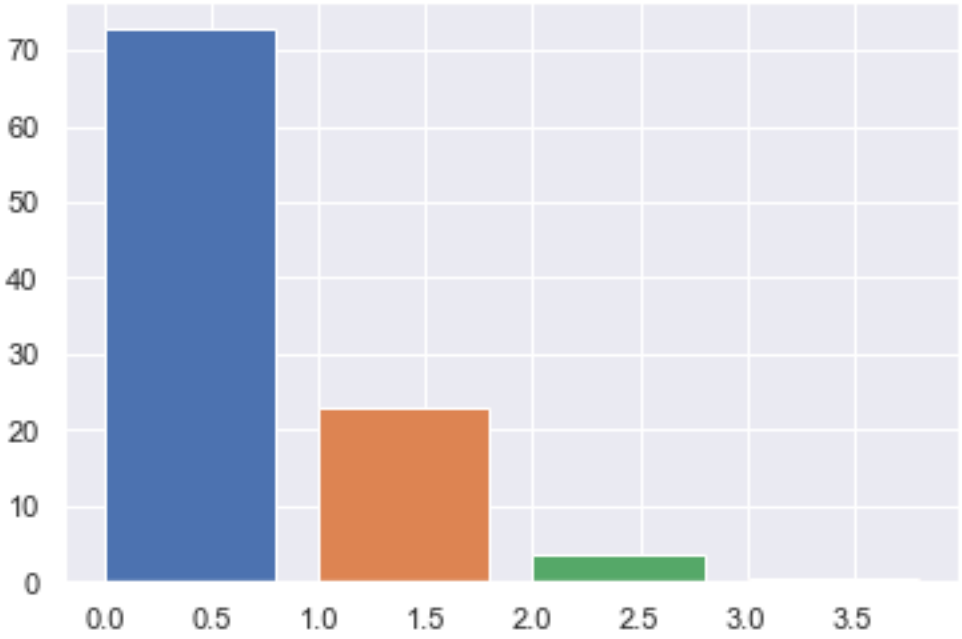
Task: Perform eigenvalue decomposition on cov_mat (use `NumPy`'s linear algebra function `eig`)

```
In [38]: # Your code here:
         # Please name the matrix as 'cov_mat'
         # Please call your variables eig_vecs and eig_vals

         eig_vals, eig_vecs = LA.eig(cov_mat)
```

Question: How many PC values should we retain?
Answer: I don't know either ;)
But we can both find it out by calulating the 'Explained Variance.'

```
In [39]: total = sum(eig_vals)
         var_exp = [(i/total)*100 for i in sorted(eig_vals, reverse=True)]

         for pos, data in enumerate(var_exp):
             plt.bar(pos, data, align='edge')
         plt.show
```

```
Out[39]: <function matplotlib.pyplot.show(*args, **kw)>
```



This plot shows that around 73% of the variance is captured by the first PC and almost 23% by the second. We can safely ignore the third and the fourth component without losing much information. In principle, we are reducing the 4D feature space to a 2D feature subspace, by choosing the "top 2" eigenvectors with the highest eigenvalues to construct a dŒEk-dimensional eigenvector matrix W. So let's construct W.

```
In [40]: # First, make a list of (eigenvalue, eigenvector) tuples
         eig_pairs = [(np.abs(eig_vals[i]), eig_vecs[:,i]) for i in range(len(eig_vals))]

         # Sort the (eigenvalue, eigenvector) tuples from high to low
         eig_pairs.sort()
         eig_pairs.reverse()

         matrix_w = np.hstack((eig_pairs[0][1].reshape(4,1),
                               eig_pairs[1][1].reshape(4,1)))

         print('Matrix W:\n', matrix_w)
```

```
Matrix W:
 [[ 0.52237162 -0.37231836]
 [-0.26335492 -0.92555649]
 [ 0.58125401 -0.02109478]
 [ 0.56561105 -0.06541577]]
```

Lastly, we will use the 4Œ2-dimensional projection matrix W to transform our samples onto the new subspace via the equation Y=XŒW, where Y is a 150Œ2 matrix of our transformed samples.

```
In [41]: Y1 = X1.dot(matrix_w)
         print(Y1)
```

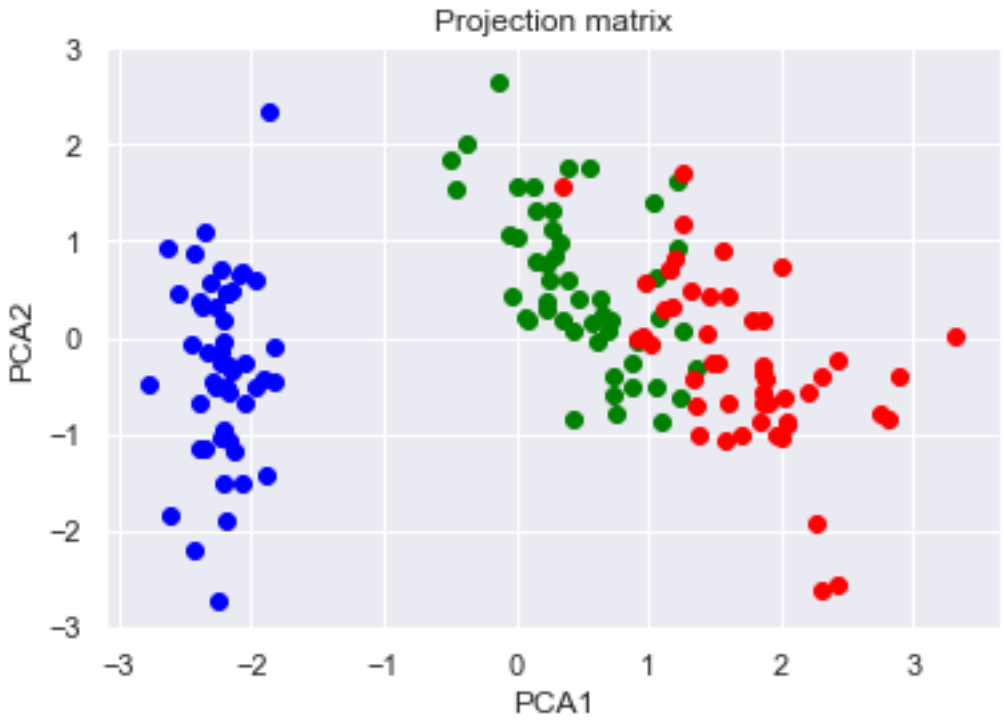
```
[[ -2.26454173e+00  -5.05703903e-01]
 [ -2.08642550e+00   6.55404729e-01]
 [ -2.36795045e+00   3.18477311e-01]
 [ -2.30419716e+00   5.75367713e-01]
 [ -2.38877749e+00  -6.74767397e-01]
 [ -2.07053681e+00  -1.51854856e+00]
 [ -2.44571134e+00  -7.45626750e-02]
 [ -2.23384186e+00  -2.47613932e-01]
 [ -2.34195768e+00   1.09514636e+00]
 [ -2.18867576e+00   4.48629048e-01]
 [ -2.16348656e+00  -1.07059558e+00]
 [ -2.32737775e+00  -1.58587455e-01]
 [ -2.22408272e+00   7.09118158e-01]
 [ -2.63971626e+00   9.38281982e-01]
 [ -2.19229151e+00  -1.88997851e+00]
 [ -2.25146521e+00  -2.72237108e+00]
 [ -2.20275048e+00  -1.51375028e+00]
 [ -2.19017916e+00  -5.14304308e-01]
 [ -1.89407429e+00  -1.43111071e+00]
 [ -2.33994907e+00  -1.15803343e+00]
 [ -1.91455639e+00  -4.30465163e-01]
 [ -2.20464540e+00  -9.52457317e-01]
 [ -2.77416979e+00  -4.89517027e-01]
 [ -1.82041156e+00  -1.06750793e-01]
 [ -2.22821750e+00  -1.62186163e-01]
 [ -1.95702401e+00   6.07892567e-01]
 [ -2.05206331e+00  -2.66014312e-01]
 [ -2.16819365e+00  -5.52016495e-01]
 [ -2.14030596e+00  -3.36640409e-01]
 [ -2.26879019e+00   3.14878603e-01]
 [ -2.14455443e+00   4.83942097e-01]
 [ -1.83193810e+00  -4.45266836e-01]
 [ -2.60820287e+00  -1.82847519e+00]
 [ -2.43795086e+00  -2.18539162e+00]
 [ -2.18867576e+00   4.48629048e-01]
 [ -2.21111990e+00   1.84337811e-01]
 [ -2.04441652e+00  -6.84956426e-01]
 [ -2.18867576e+00   4.48629048e-01]
 [ -2.43595220e+00   8.82169415e-01]
 [ -2.17054720e+00  -2.92726955e-01]
 [ -2.28652724e+00  -4.67991716e-01]
```

```
[ -1.87170722e+00   2.32769161e+00]
[ -2.55783442e+00   4.53816380e-01]
[ -1.96427929e+00  -4.97391640e-01]
[ -2.13337283e+00  -1.17143211e+00]
[ -2.07535759e+00   6.91917347e-01]
[ -2.38125822e+00  -1.15063259e+00]
[ -2.39819169e+00   3.62390765e-01]
[ -2.22678121e+00  -1.02548255e+00]
[ -2.20595417e+00  -3.22378453e-02]
[  1.10399365e+00  -8.63112446e-01]
[  7.32481440e-01  -5.98635573e-01]
[  1.24210951e+00  -6.14822450e-01]
[  3.97307283e-01   1.75816895e+00]
[  1.07259395e+00   2.11757903e-01]
[  3.84458146e-01   5.91062469e-01]
[  7.48715076e-01  -7.78698611e-01]
[ -4.97863388e-01   1.84886877e+00]
[  9.26222368e-01  -3.03308268e-02]
[  4.96802558e-03   1.02940111e+00]
[ -1.24697461e-01   2.65806268e+00]
[  4.38730118e-01   5.88812850e-02]
[  5.51633981e-01   1.77258156e+00]
[  7.17165066e-01   1.85434315e-01]
[ -3.72583830e-02   4.32795099e-01]
[  8.75890536e-01  -5.09998151e-01]
[  3.48006402e-01   1.90621647e-01]
[  1.53392545e-01   7.90725456e-01]
[  1.21530321e+00   1.63335564e+00]
[  1.56941176e-01   1.30310327e+00]
[  7.38256104e-01  -4.02470382e-01]
[  4.72369682e-01   4.16608222e-01]
[  1.22798821e+00   9.40914793e-01]
[  6.29381045e-01   4.16811643e-01]
[  7.00472799e-01   6.34939277e-02]
[  8.73536987e-01  -2.50708611e-01]
[  1.25422219e+00   8.26200998e-02]
[  1.35823985e+00  -3.28820266e-01]
[  6.62126138e-01   2.24346071e-01]
[ -4.72815133e-02   1.05721241e+00]
[  1.21534209e-01   1.56359238e+00]
[  1.41182261e-02   1.57339235e+00]
[  2.36010837e-01   7.75923784e-01]
[  1.05669143e+00   6.36901284e-01]
[  2.21417088e-01   2.80847693e-01]
[  4.31783161e-01  -8.55136920e-01]
[  1.04941336e+00  -5.22197265e-01]
[  1.03587821e+00   1.39246648e+00]
[  6.70675999e-02   2.12620735e-01]
```

```
[  2.75425066e-01   1.32981591e+00]
[  2.72335066e-01   1.11944152e+00]
[  6.23170540e-01  -2.75426333e-02]
[  3.30005364e-01   9.88900732e-01]
[ -3.73627623e-01   2.01793227e+00]
[  2.82944343e-01   8.53950717e-01]
[  8.90531103e-02   1.74908548e-01]
[  2.24356783e-01   3.80484659e-01]
[  5.73883486e-01   1.53719974e-01]
[ -4.57012873e-01   1.53946451e+00]
[  2.52244473e-01   5.95860746e-01]
[  1.84767259e+00  -8.71696662e-01]
[  1.15318981e+00   7.01326114e-01]
[  2.20634950e+00  -5.54470105e-01]
[  1.43868540e+00   5.00105223e-02]
[  1.86789070e+00  -2.91192802e-01]
[  2.75419671e+00  -7.88432206e-01]
[  3.58374475e-01   1.56009458e+00]
[  2.30300590e+00  -4.09516695e-01]
[  2.00173530e+00   7.23865359e-01]
[  2.26755460e+00  -1.92144299e+00]
[  1.36590943e+00  -6.93948040e-01]
[  1.59906459e+00   4.28248836e-01]
[  1.88425185e+00  -4.14332758e-01]
[  1.25308651e+00   1.16739134e+00]
[  1.46406152e+00   4.44147569e-01]
[  1.59180930e+00  -6.77035372e-01]
[  1.47128019e+00  -2.53192472e-01]
[  2.43737848e+00  -2.55675734e+00]
[  3.30914118e+00   2.36132010e-03]
[  1.25398099e+00   1.71758384e+00]
[  2.04049626e+00  -9.07398765e-01]
[  9.73915114e-01   5.71174376e-01]
[  2.89806444e+00  -3.97791359e-01]
[  1.32919369e+00   4.86760542e-01]
[  1.70424071e+00  -1.01414842e+00]
[  1.95772766e+00  -1.00333452e+00]
[  1.17190451e+00   3.18896617e-01]
[  1.01978105e+00  -6.55429631e-02]
[  1.78600886e+00   1.93272800e-01]
[  1.86477791e+00  -5.55381532e-01]
[  2.43549739e+00  -2.46654468e-01]
[  2.31608241e+00  -2.62618387e+00]
[  1.86037143e+00   1.84672394e-01]
[  1.11127173e+00   2.95986102e-01]
[  1.19746916e+00   8.17167742e-01]
[  2.80094940e+00  -8.44748194e-01]
[  1.58015525e+00  -1.07247450e+00]
```

```
[  1.34704442e+00  -4.22255966e-01]
[  9.23432978e-01  -1.92303705e-02]
[  1.85355198e+00  -6.72422729e-01]
[  2.01615720e+00  -6.10397038e-01]
[  1.90311686e+00  -6.86024832e-01]
[  1.15318981e+00   7.01326114e-01]
[  2.04330844e+00  -8.64684880e-01]
[  2.00169097e+00  -1.04855005e+00]
[  1.87052207e+00  -3.82821838e-01]
[  1.55849189e+00   9.05313601e-01]
[  1.52084506e+00  -2.66794575e-01]
[  1.37639119e+00  -1.01636193e+00]
[  9.59298576e-01   2.22839447e-02]]
```

In [42]:
```
species = ('Iris-setosa', 'Iris-versicolor', 'Iris-virginica')
colors = ('blue','green','red')

# Plotting the relationship between PCA1 and PCA2
feature1 = Y1[:, 0]
feature2 = Y1[:, 1]

species = ('Iris-setosa', 'Iris-versicolor', 'Iris-virginica')
colors = ('blue','green','red')
data = [[Y1[np.where(targets == target)][:, feature] for feature in [0, 1]] for targe

for item, color, group in zip(data, colors, species):
    plt.scatter(item[0], item[1], color=color)
    plt.title('Projection matrix')
plt.xlabel('PCA1')
plt.ylabel('PCA2')
plt.show()
```



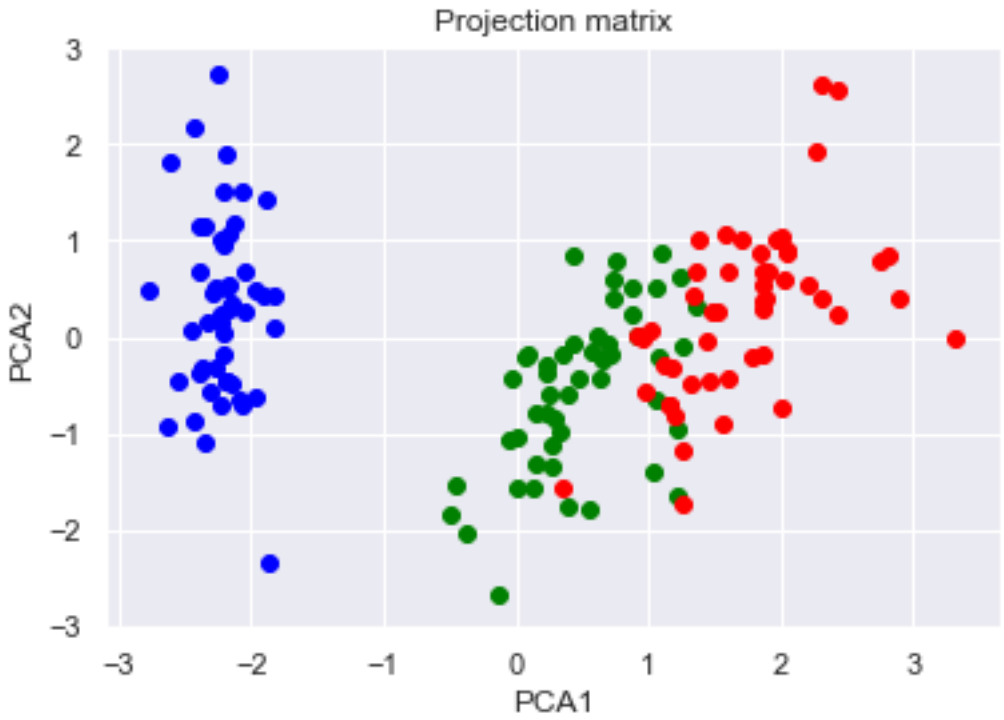
2.6.2 PCA using `scikit-learn`

```
In [43]: # importing necessary packages
         X_1 = features
         Y_1 = targets
         # importing necessary packages for standardizing
         from sklearn.preprocessing import StandardScaler

         from numpy import linalg as LA

         from sklearn.decomposition import PCA as sklearnPCA
         #from mpl_toolkits.mplot3d import Axes3D
```

Task: Perform PCA using the built-in function `sklearnPCA`

```
In [44]: X_1 = StandardScaler().fit_transform(X_1)
         pca = sklearnPCA(n_components=2)

         principalComponents = pca.fit_transform(X_1)

         #principalDf = pd.DataFrame(data = principalComponents
         #             , columns = ['principal component 1', 'principal component 2'])

         #finalDf = pd.concat([principalDf, df[['target']]], axis = 1)
```

```
#print(principalComponents)

feature1 = principalComponents[:, 0]
feature2 = principalComponents[:, 1]

species = ('Iris-setosa', 'Iris-versicolor', 'Iris-virginica')
colors = ('blue','green','red')
data = [[principalComponents[np.where(targets == target)][:, feature] for feature in

for item, color, group in zip(data, colors, species):
    plt.scatter(item[0], item[1], color=color)
    plt.title('Projection matrix')
plt.xlabel('PCA1')
plt.ylabel('PCA2')
plt.show()
```



Task: Plot the features you found using `sklearn`'s implementation of the PCA in a similar way as we showed previously using the hand-implemented method. This will also verify if our implementation is correct and gives the same result as the built-in function.

```
In [ ]: # Your code here:
```

2.7 End Remark about coding in Python ;)

```
In [ ]: # Uncomment the code below
        # import this
```

2.8 Bonus Task: Segmentation by simple thresholding

Consider the image irisNorm1 we obtained previously. Extract a segmentation mask for the flower using a basic thresholding based on the histogram of the image that we displayed. A mask is basically a binary image with pixel values as 0 for the region of interest, and all other pixels as 1.

```
In [ ]: # Your code here:
        # Please name the mask segMask
```

2.9 Feedback Cell:

I hope this Notebook gave you a good start into python and Machine Learning. Let us know how you liked it. Any suggestions/ criticism are also welcome!

Your Feedback: