Lab Session 1

«Data science libraries, KNN Model»

---

1. Introduction

Python is considered to be one of the best Programming languages when it comes to the developpement of applications/projects for data science. It does provide great libraries to deal with mathematics, statistics and data science application.

In this lab session you will put into test the different functions/functionalities of Python Libraries, followed with a K_{NN} application.

2. Learning outcome

On successful completion of this lab session you will be able to :

- Enhance your calculations using SciPy and NumPy libraries.
- Create and customize visualisations of datasets or mathematical functions, using Pandas and Matplotlib libraries.
- Handle mglearn datasets.
- Build your first K_{NN} Model.

3. Ressources

Libraries Documentation

- Python : https://docs.python.org/3/
- NumPy : https://numpy.org/doc/
- SciPy : https://docs.scipy.org/doc/scipy/
- Matplotlib : https://matplotlib.org/3.5.1/
- Panda : https://pandas.pydata.org/docs/
- mglearn : https://libraries.io/pypi/mglearn
- sklearn : https://scikit-learn.org/stable/

**to proceed with the lab session, refer to this section*

4. Getting Started

In these lab sessions, you will use Jupyter Notebook as it integrates features that targets Data science (Live code & data visualisation).

For that,you need to install Anaconda as it does contain python environement, Jupyter and datascience packages. you can refer to the installation documentation
https://docs.anaconda.com/anaconda/install/index.html

Create a new folder "DS54 Lab Sessions", where you will launch and store your jupyter notebooks by running the following comand

```
> mkdir DS54_Lab_Sessions
> cd DS54_Lab_Sessions
> jupyter notebook
```

if you have problems with your installation, you can proceed with google collab platform : https://colab.research.google.com

If you have any troubles importing libraries you can use this command :

```
>!pip install *library name* #!pip install mglearn
```

5. Python Basics

Operations

Operation	Result
x + y	addition
x - y	Substraction
x * y	Multiplication
x / y	Division
x // y	Floor Division
x % y	Modulus
-x	negation
+x	x unchanged
abs(x)	absolute value
int(x)	conversion to integer
long(x)	conversion to long
float(x)	conversion to float
complex(re,im)	complex conversion
c.conjugate()	conjugaison
divmod(x, y)	Pair (x // y, x % y)
pow(x, y)	x power y
x ** y	x power y

Numbers

In [ ]:

```
x = 3
print(type(x)) # Prints "<class 'int'>"
print(x)       # Prints "3"
print(x + 1)   # Addition; prints "4"
print(x - 1)   # Subtraction; prints "2"
print(x * 2)   # Multiplication; prints "6"
print(x ** 2)  # Exponentiation; prints "9"
x += 1
print(x)  # Prints "4"
x *= 2
print(x)  # Prints "8"
y = 2.5
print(type(y)) # Prints "<class 'float'>"
print(y, y + 1, y * 2, y ** 2) # Prints "2.5 3.5 5.0 6.25"
```

Boolean

In [ ]:

```
t = True
f = False
print(type(t)) # Prints "<class 'bool'>"
```

```
print(t and f) # Logical AND; prints "False"
print(t or f)  # Logical OR; prints "True"
print(not t)   # Logical NOT; prints "False"
print(t != f)  # Logical XOR; prints "True"
```

Strings

In [ ]:

```
hello = 'hello'    # String literals can use single quotes
world = "world"    # or double quotes; it does not matter.
print(hello)       # Prints "hello"
print(len(hello))  # String length; prints "5"
hw = hello + ' ' + world  # String concatenation
print(hw)  # prints "hello world"
hw12 = '%s %s %d' % (hello, world, 12)  # sprintf style string formatting
print(hw12)  # prints "hello world 12"
```

Strings methods

In [ ]:

```
s = "hello"
print(s.capitalize())  # Capitalize a string; prints "Hello"
print(s.upper())       # Convert a string to uppercase; prints "HELLO"
print(s.rjust(7))      # Right-justify a string, padding with spaces; prints "  hello"
print(s.center(7))     # Center a string, padding with spaces; prints " hello "
print(s.replace('l', '(ell)'))  # Replace all instances of one substring with another;
                                # prints "he(ell)(ell)o"
print('  world '.strip())  # Strip leading and trailing whitespace; prints "world"
```

Conditions

In [ ]:

```
x=10
if x < -3:
        print("ok")
print('x>=-3')
```

In [ ]:

```
x=10
if x < -3:
    print('x<-3')
elif x == 0:
    print('x=0')
elif x <= 5:
    print('x<=5')
else :
    print ('x>5')
# affectation
y = x
if x is y :
    print('x et y same objects')
# modification de x
x +=1
if x is y :
    print('x et y same objects')
else:
    print('x et y different objects')
```

Loops

In [ ]:

```
for x in range(0, 10):
  print(x)
  # Prints numbers from 0 to 9
```

In [ ]:
```
nums = [0, 1, 2, 3, 4]
squares = []
for x in nums:
    squares.append(x ** 2)
print(squares)   # Prints [0, 1, 4, 9, 16]
```

In [ ]:
```
animals = ['cat', 'dog', 'monkey']
for animal in animals:
    print(animal)
# Prints "cat", "dog", "monkey", each on its own line.
```

In [ ]:
```
animals = ['cat', 'dog', 'monkey']
for idx, animal in enumerate(animals):
    print(idx, animal)
    print('%d: %s' % (idx + 1, animal))
# Prints 0
```

Functions

In [ ]:
```
def sign(x):
    if x > 0:
        return 'positive'
    elif x < 0:
        return 'negative'
    else:
        return 'zero'

for x in [-1, 0, 1]:
    print(sign(x))
# Prints "negative", "zero", "positive"
```

In [ ]:
```
def hello(name, loud=False):
    if loud:
        print('HELLO, %s!' % name.upper())
    else:
        print('Hello, %s' % name)

hello('Bob') # Prints "Hello, Bob"
hello('Fred', loud=True)  # Prints "HELLO, FRED!"
#Optional arguments
```

Exercice

Write a function that generates fibonacci numbers, then generate a fibonacci sequence

refer to the documentation : https://en.wikipedia.org/wiki/Fibonacci_number

6. Essential libraries

6.0 Versions Used in this Book

check the libraries version by running the following code

In [ ]:
```
import sys
print("Python version:", sys.version)
import pandas as pd
print("pandas version:", pd.__version__)
import matplotlib
print("matplotlib version:", matplotlib.__version__)
```

```
import numpy as np
print("NumPy version:", np.__version__)
import scipy as sp
print("SciPy version:", sp.__version__)
import IPython
print("IPython version:", IPython.__version__)
import sklearn
print("scikit-learn version:", sklearn.__version__)
```

6.1 NumPy : Operations on arrays and matrixs

Numpy is a library to facilitate digital processing. It allows you to easily create and manipulate multidimensional matrices. The use of numpy is close to the MATLAB software.

Arrays

In [ ]:

```
import numpy as np

a = np.array([1, 2, 3])   # Create a rank 1 array
print(type(a))            # Prints "<class 'numpy.ndarray'>"
print(a.shape)            # Prints "(3,)"
print(a[0], a[1], a[2])   # Prints "1 2 3"
a[0] = 5                  # Change an element of the array
print(a)                  # Prints "[5, 2, 3]"

b = np.array([[1,2,3],[4,5,6]])    # Create a rank 2 array
print(b.shape)                     # Prints "(2, 3)"
print(b[0, 0], b[0, 1], b[1, 0])   # Prints "1 2 4"
```

In [ ]:

```
import numpy as np

a = np.array([[1,2], [3, 4], [5, 6]])

# An example of integer array indexing.
# The returned array will have shape (3,) and
print(a[[0, 1, 2], [0, 1, 0]])  # Prints "[1 4 5]"

# The above example of integer array indexing is equivalent to this:
print(np.array([a[0, 0], a[1, 1], a[2, 0]]))  # Prints "[1 4 5]"

# When using integer array indexing, you can reuse the same
# element from the source array:
print(a[[0, 0], [1, 1]])  # Prints "[2 2]"

# Equivalent to the previous integer array indexing example
print(np.array([a[0, 1], a[0, 1]]))  # Prints "[2 2]"
```

In [ ]:

```
import numpy as np

a = np.array([[1,2], [3, 4], [5, 6]])

bool_idx = (a > 2)   # Find the elements of a that are bigger than 2;
                     # this returns a numpy array of Booleans of the same
                     # shape as a, where each slot of bool_idx tells
                     # whether that element of a is > 2.

print(bool_idx)      # Prints "[[False False]
                     #          [ True  True]
                     #          [ True  True]]"

# We use boolean array indexing to construct a rank 1 array
# consisting of the elements of a corresponding to the True values
```

```
# of bool_idx
print(a[bool_idx])  # Prints "[3 4 5 6]"

# We can do all of the above in a single concise statement:
print(a[a > 2])     # Prints "[3 4 5 6]"
```

Exercises

1. Write a program that reverse a numpy array (the first element becomes the last one)

1. Replace all odd numbers in a given random array with -1

6.2 SciPy : Advanced calculations

In [ ]:
```
from scipy import sparse
# Create a 2D NumPy array with a diagonal of ones, and zeros everywhere else
eye = np.eye(4)
print("NumPy array:\n", eye)
```

In [ ]:
```
# Convert the NumPy array to a SciPy sparse matrix in CSR format
# Only the nonzero entries are stored
sparse_matrix = sparse.csr_matrix(eye)
print("\nSciPy sparse CSR matrix:\n", sparse_matrix)
```

In [ ]:
```
data = np.ones(4)
row_indices = np.arange(4)
col_indices = np.arange(4)
eye_coo = sparse.coo_matrix((data, (row_indices, col_indices)))
print("COO representation:\n", eye_coo)
```

Exercise : Scipy

1. Create a 5x5 identity matrix.
2. Create a 5x5 matrix with randomly distributed values and a density of 0,5

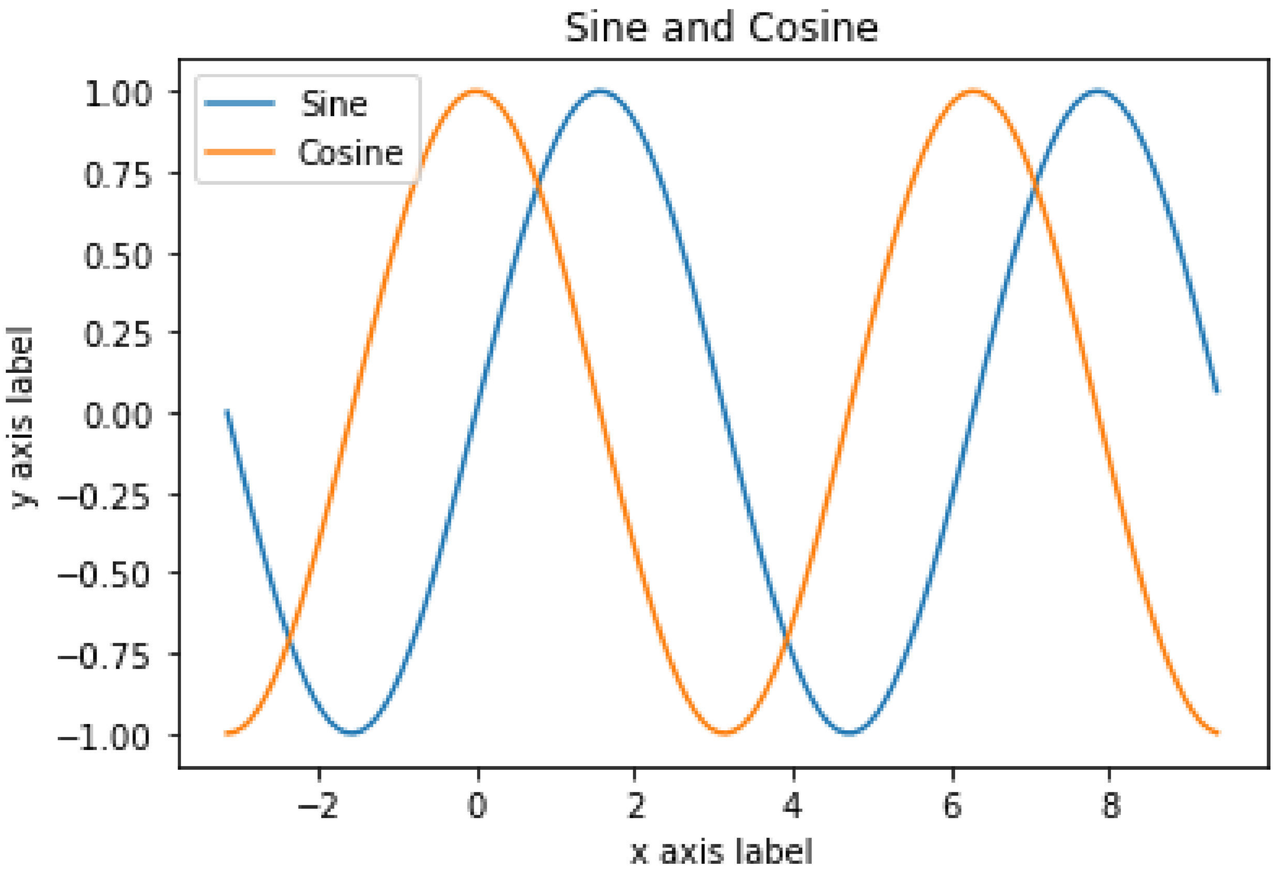
6.3 MatplotLib :

In [ ]:
```
%matplotlib inline
import matplotlib.pyplot as plt
# Generate a sequence of numbers from -10 to 10 with 100 steps in between
x = np.linspace(-10, 10, 100)
# Create a second array using sine
y = np.sin(x)
# The plot function makes a line chart of one array against another
plt.plot(x, y, marker="x")
```

Recreate the picture below with python code as best as you can.

You need to recreate the 2 curves, add the name of the x and y axis, add the legend and the name to the picture.

Matplotlib documentation : https://matplotlib.org/stable/api/matplotlib_configuration_api.html



Recreate the picture, part 2

1. Add a grid on the diagram.
2. Turn the sin curve into a dotted line.
3. Turn the cos line color red and the cos line width 5 times larger.
4. Make a new version of the picture with 2 subplots, one showing the sin curve and one showing the cos curve.

Matplotlib in 3D

With matplotlib, show a 3D curve.

The x axis is the sine function

The y axis is the cosine function.

The z axis is defined as :

```
np.linspace(0, 15, 1000)
```

6.4 Pandas :

In [ ]:

```
import pandas as pd
# create a simple dataset of people
data = {'Name': ["John", "Anna", "Peter", "Linda"],
'Location' : ["New York", "Paris", "Berlin", "London"],
'Age' : [24, 13, 53, 33]
}
data_pandas = pd.DataFrame(data)
# IPython.display allows "pretty printing" of dataframes
# in the Jupyter notebook
display(data_pandas)
```

	Name	Location	Age
0	John	New York	24
1	Anna	Paris	13
2	Peter	Berlin	53
3	Linda	London	33

In [ ]:

```
# Select all rows that have an age column greater than 30
display(data_pandas[data_pandas.Age > 30])
```

	Name	Location	Age
2	Peter	Berlin	53
3	Linda	London	33

Exercise : Pandas

1. Recreate the following dataframe :

	Fruit	Price	Stock	Like
0	Apple	0.50	20	True
1	Banana	0.30	35	True
2	Pineapple	0.90	17	False
3	Strawberry	0.85	24	True
4	Apricot	1.20	9	False
5	Mango	1.20	14	False
6	Peach	1.10	3	True
7	Cherry	1.20	16	True
8	Orange	0.65	22	False
9	Watermelon	0.20	5	False

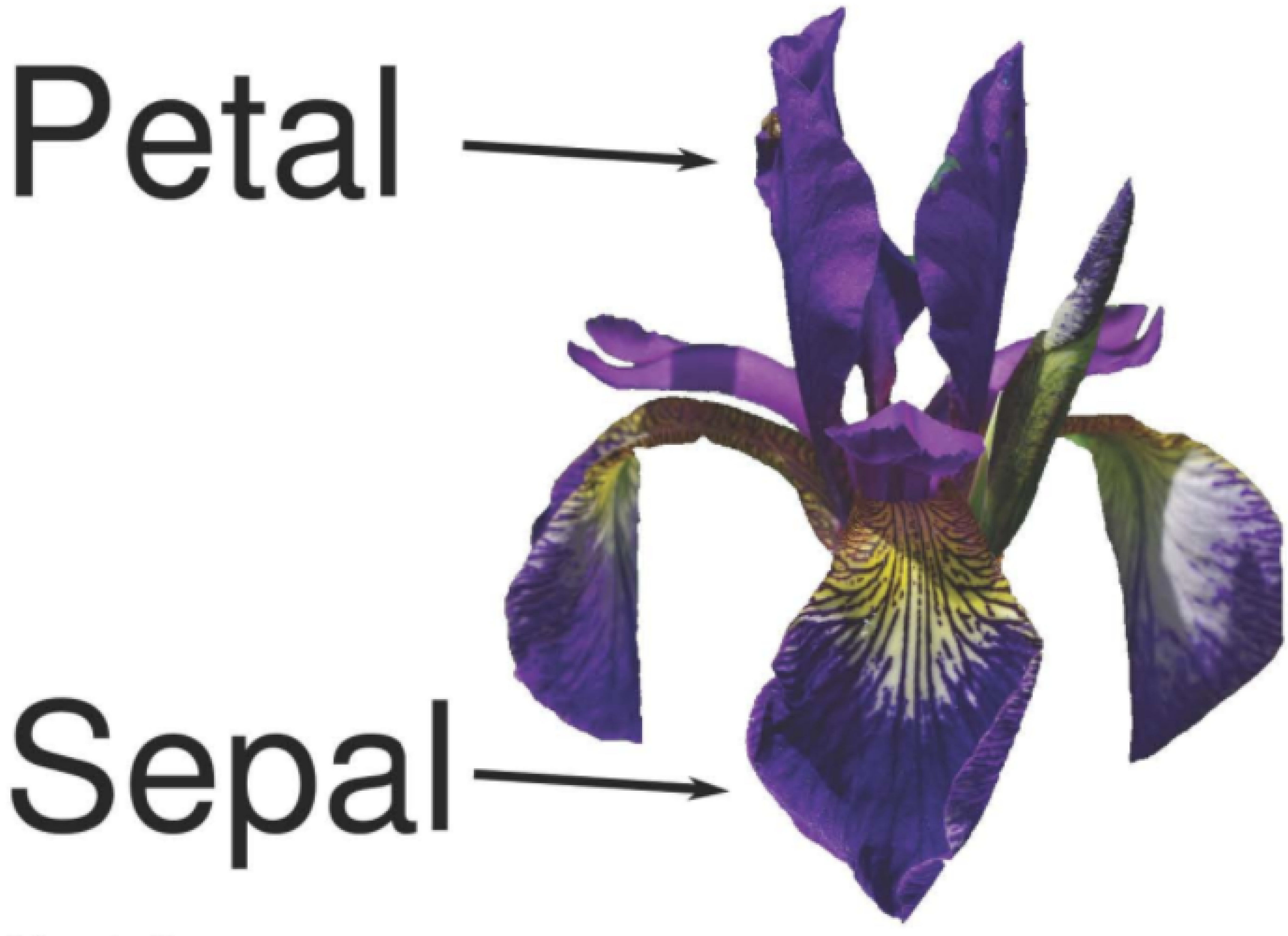
1. With Pandas, display :

- The fruits that have a price above 1.00
- The fruits that are liked
- The fruits that have a stock value that are multiples of 3
- The fruits with a name with more than 8 characters.

1. Modify the dataframe :

- Rename the column "Name" into "Fruit"
- Add a column "Color" with a color of your choice for each fruit.
- Delete the column "Like".
- Change all "Stock" value to 1.

7. A First Application : Classifying Iris Species



Meet the Data

In [ ]:
```
from sklearn.datasets import load_iris
iris_dataset = load_iris()
```

In [ ]:
```
print("Keys of iris_dataset:\n", iris_dataset.keys())
```

In [ ]:
```
print(iris_dataset['DESCR'][:193] + "\n...")
```

In [ ]:
```
print("Target names:", iris_dataset['target_names'])
```

In [ ]:
```
print("Feature names:\n", iris_dataset['feature_names'])
```

In [ ]:
```
print("Type of data:", type(iris_dataset['data']))
```

In [ ]:
```
print("Shape of data:", iris_dataset['data'].shape)
```

In [ ]:
```
print("First five rows of data:\n", iris_dataset['data'][:5])
```

In [ ]:
```
print("Type of target:", type(iris_dataset['target']))
```

In [ ]:
```
print("Shape of target:", iris_dataset['target'].shape)
```

In [ ]:
```
print("Target:\n", iris_dataset['target'])
```

Measuring success: Training and Testing Data

In [ ]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
iris_dataset['data'], iris_dataset['target'], random_state=0)
```

In [ ]:
```
print("X_train shape:", X_train.shape)
print("y_train shape:", y_train.shape)
```

In [ ]:
```
print("X_test shape:", X_test.shape)
print("y_test shape:", y_test.shape)
```

First Things First: Look at Your Data

In [ ]:
```
import mglearn
# create dataframe from data in X_train
# label the columns using the strings in iris_dataset.feature_names
iris_dataframe = pd.DataFrame(X_train, columns=iris_dataset.feature_names)
# create a scatter matrix from the dataframe, color by y_train
pd.plotting.scatter_matrix(iris_dataframe, c=y_train, figsize=(15, 15),
marker='o', hist_kwds={'bins': 20}, s=60,
alpha=.8, cmap=mglearn.cm3)
```

Building your first Model : k-Nearest Neighbors

In [ ]:
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=1)
```

In [ ]:
```
knn.fit(X_train, y_train)
```

Making Prediction

In [ ]:
```
X_new = np.array([[5, 2.9, 1, 0.2]])
print("X_new.shape:", X_new.shape)
```

In [ ]:
```
prediction = knn.predict(X_new)
print("Prediction:", prediction)
print("Predicted target name:",
iris_dataset['target_names'][prediction])
```

Evaluating the Model

In [ ]:
```
y_pred = knn.predict(X_test)
print("Test set predictions:\n", y_pred)
```

In [ ]:
```
print("Test set score: {:.2f}".format(np.mean(y_pred == y_test)))
```

In [ ]:
```
print("Test set score: {:.2f}".format(knn.score(X_test, y_test)))
```

Summary and Outlook

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(
iris_dataset['data'], iris_dataset['target'], random_state=0)
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train, y_train)
print("Test set score: {:.2f}".format(knn.score(X_test, y_test)))
```

8. Application

In [ ]:
```
from sklearn.datasets import load_breast_cancer
from sklearn.neighbors import KNeighborsClassifier

cancer = load_breast_cancer()

X=cancer.data
y=cancer.target
```

Exercise : Breast Cancer

1. Use a K-NN on this database.
2. Write a script to visualise the accuracy of your K-NN for different value of k.

Exercise : Diabetes

1. Build a Knn model of the load digits dataset and print the test score. *refer to the documentation https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_digits.html*

    1. Visualise the dataset

    1. Make a subplot of 6 digits (from 0 to 5)

    1. Write cross validation script for 1 neighbor

    1. write a script that visualise the cross validation per k neighbor. What is the best K ?