Experiment 2: Mathematical computing with Python packages like: Numpy, MatplotLib, Pandas

1. Numpy

NumPy is a Python library used for working with arrays. It also has functions for working in domain of linear algebra, fourier transform, and matrices. The array object in NumPy is called ndarray, it provides a lot of supporting functions that make working with ndarray very easy. Once NumPy is installed, import it in your applications by adding the import keyword:

```
import numpy
```

NumPy is usually imported under the np alias.

```
import numpy as np
```

Example

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
print(arr)
```

- Create a NumPy ndarray Object

NumPy is used to work with arrays. The array object in NumPy is called ndarray. We can create a NumPy ndarray object by using the array() function.

Example

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
print(arr)
print(type(arr))
```

type(): This built-in Python function tells us the type of the object passed to it. Like in above code it shows that arr is numpy.ndarray type.

- To create an ndarray, we can pass a list, tuple or any array-like object into the array() method, and it will be converted into an ndarray:

Example

Use a tuple to create a NumPy array:

```
import numpy as np
arr = np.array((1, 2, 3, 4, 5))

print(arr)
```

- Dimensions in Arrays

A dimension in arrays is one level of array depth (nested arrays). **nested array:** are arrays that have arrays as their elements.

0-D Arrays

0-D arrays, or Scalars, are the elements in an array. Each value in an array is a 0-D array.

Example

Create a 0-D array with value 42

```
import numpy as np
arr = np.array(42)
print(arr)
```

- 1-D Arrays

An array that has 0-D arrays as its elements is called uni-dimensional or 1-D array.

These are the most common and basic arrays.

Example

Create a 1-D array containing the values 1,2,3,4,5:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
print(arr)
```

- 2-D Arrays

An array that has 1-D arrays as its elements is called a 2-D array.

Example

Create a 2-D array containing two arrays with the values 1,2,3 and 4,5,6:

```
import numpy as np
arr = np.array([[1, 2, 3], [4, 5, 6]])
print(arr)
```

- Access Array Elements

Array indexing is the same as accessing an array element. You can access an array element by referring to its index number. The indexes in NumPy arrays start with 0, meaning that the first element has index 0, and the second has index 1 etc.

Example

Get the first element from the following array:

```
import numpy as np
arr = np.array([1, 2, 3, 4])
print(arr[0])
```

- Access 2-D Arrays

To access elements from 2-D arrays we can use comma separated integers representing the dimension and the index of the element. Think of 2-D arrays like a table with rows and columns, where the row represents the dimension and the index represents the column.

Example

Access the element on the first row, second column:

```
import numpy as np
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
print('2nd element on 1st row: ', arr[0, 1])
```

- Slicing arrays

Slicing in python means taking elements from one given index to another given index. We pass slice instead of index like this: [*start*:*end*]. We can also define the step, like this: [*start*:*end*:*step*].

Example

Slice elements from index 1 to index 5 from the following array:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[1:5])
```

Example

Slice elements from index 4 to the end of the array:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[4:])
```

- Checking the Data Type of an Array

The NumPy array object has a property called dtype that returns the data type of the array:

Example

Get the data type of an array object:

```
import numpy as np
arr = np.array([1, 2, 3, 4])
print(arr.dtype)
```

- Get the Shape of an Array

NumPy arrays have an attribute called shape that returns a tuple with each index having the number of corresponding elements.

Example

Print the shape of a 2-D array:

```
import numpy as np
arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])
print(arr.shape)
```

The example above returns (2, 4), which means that the array has 2 dimensions, and each dimension has 4 elements.

- Reshaping arrays

Reshaping means changing the shape of an array. The shape of an array is the number of elements in each dimension. By reshaping we can add or remove dimensions or change number of elements in each dimension.

Example

Convert the following 1-D array with 12 elements into a 2-D array.The outermost dimension will have 4 arrays, each with 3 elements:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
newarr = arr.reshape(4, 3)
print(newarr)
```

2. Pandas

Pandas is a Python library. Pandas is used to analyze data. Pandas allows us to analyze big data and make conclusions based on statistical theories. Pandas can clean messy data sets, and make them readable and relevant. Relevant data is very important in data science.

Pandas gives you answers about the data. Like:

- Is there a correlation between two or more columns?
- What is average value?
- Max value?
- Min value?

Pandas are also able to delete rows that are not relevant, or contains wrong values, like empty or NULL values. This is called *cleaning* the data.

```
import pandas as pd
mydataset = {
  'cars': ["BMW", "Volvo", "Ford"],
  'passings': [3, 7, 2]
}
myvar = pd.DataFrame(mydataset)
print(myvar)
```

- What is a DataFrame?

A Pandas DataFrame is a 2 dimensional data structure, like a 2 dimensional array, or a table with rows and columns.

Example

Create a simple Pandas DataFrame:

```
import pandas as pd
data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}

#load data into a DataFrame object:
df = pd.DataFrame(data)
print(df)
```

Result

```
   calories  duration
0       420        50
1       380        40
2       390        45
```

Locate Row

As you can see from the result above, the DataFrame is like a table with rows and columns.

Pandas use the loc attribute to return one or more specified row(s)

Example

Return row 0:

```
#refer to the row index:
print(df.loc[0])
```

Result

```
calories    420
duration     50
Name: 0, dtype: int64
```

- Read CSV Files

A simple way to store big data sets is to use CSV files (comma separated files). CSV files contains plain text and is a well know format that can be read by everyone including Pandas. In our examples we will be using a CSV file called 'data.csv'.

Example

Load the CSV into a DataFrame:

```
import pandas as pd
df = pd.read_csv('data.csv')
print(df.to_string())
```

By default, when you print a DataFrame, you will only get the first 5 rows, and the last 5 rows:

Example

Print a reduced sample:

```
import pandas as pd
df = pd.read_csv('data.csv')
print(df)
```

- Viewing the Data

One of the most used method for getting a quick overview of the DataFrame, is the head() method. The head() method returns the headers and a specified number of rows, starting from the top.

Example

Get a quick overview by printing the first 10 rows of the DataFrame:

```
import pandas as pd
df = pd.read_csv('data.csv')
print(df.head(10))
```

Note: if the number of rows is not specified, the head() method will return the top 5 rows.

There is also a tail() method for viewing the *last* rows of the DataFrame. The tail() method returns the headers and a specified number of rows, starting from the bottom.

Example

Print the last 5 rows of the DataFrame:

```
print(df.tail())
```

3. Matplotlib

- Pyplot

Most of the Matplotlib utilities lies under the pyplot submodule, and are usually imported under the plt alias:

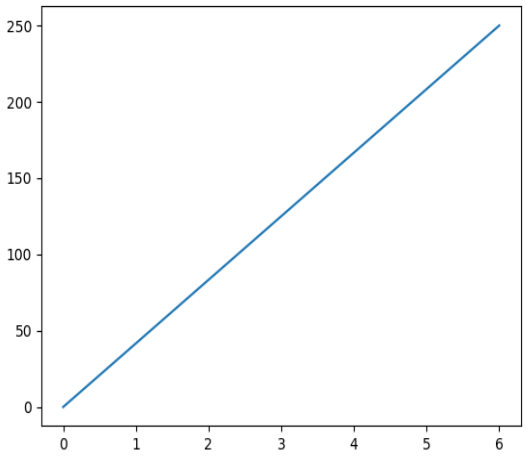
```
import matplotlib.pyplot as plt
```

Example

Draw a line in a diagram from position (0,0) to position (6,250):

```
import matplotlib.pyplot as plt
import numpy as np
xpoints = np.array([0, 6])
ypoints = np.array([0, 250])
plt.plot(xpoints, ypoints)
plt.show()
```

Result:



- Linestyle

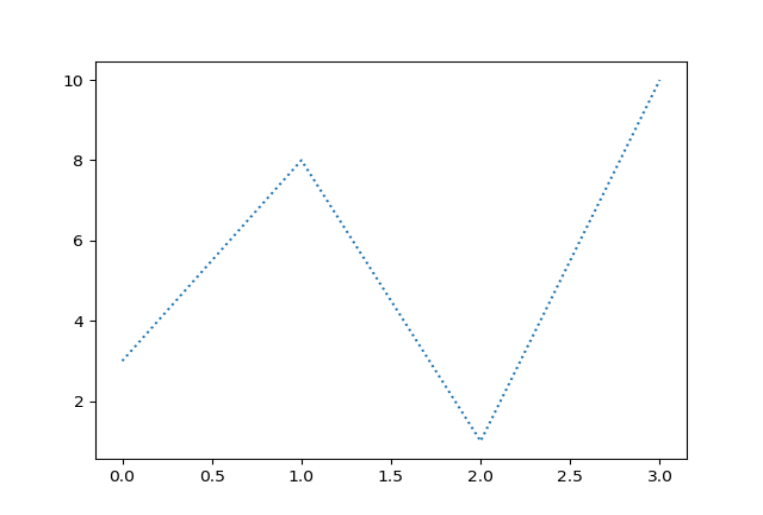
You can use the keyword argument linestyle, or shorter ls, to change the style of the plotted line:

Example

Use a dotted line:

```
import matplotlib.pyplot as plt
import numpy as np
ypoints = np.array([3, 8, 1, 10])
plt.plot(ypoints, linestyle = 'dotted')
plt.show()
```

Result:



- Create Labels for a Plot

With Pyplot, you can use the xlabel() and ylabel() functions to set a label for the x- and y-axis.

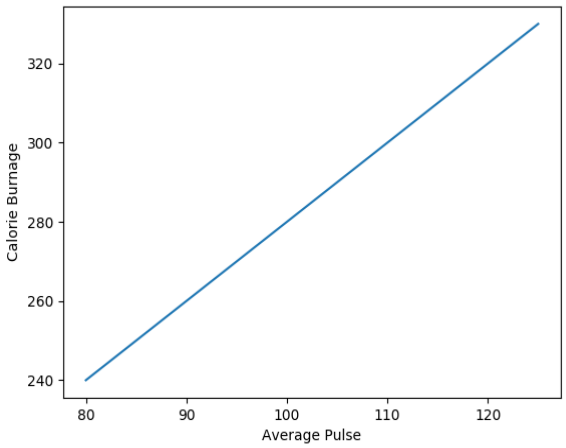
Example

Add labels to the x- and y-axis:

```
import numpy as np
import matplotlib.pyplot as plt
x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])
plt.plot(x, y)

plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")
plt.show()
```

Result:



- Creating Scatter Plots

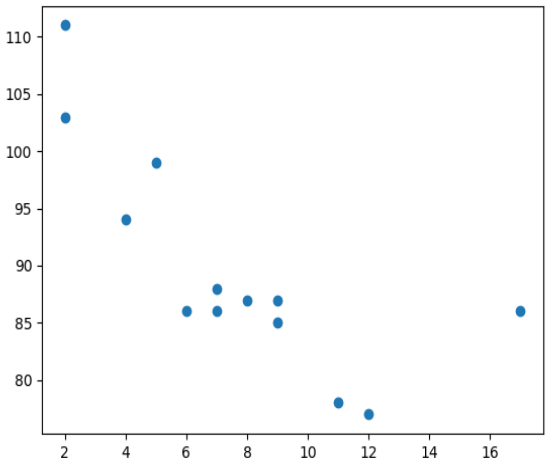
With Pyplot, you can use the scatter() function to draw a scatter plot. The scatter() function plots one dot for each observation. It needs two arrays of the same length, one for the values of the x-axis, and one for values on the y-axis:

Example

A simple scatter plot:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
plt.scatter(x, y)
plt.show()
```

Result:



- Creating Bars

With Pyplot, you can use the bar() function to draw bar graphs:

Example

Draw 4 bars:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])
plt.bar(x,y)
plt.show()
```

Result: