EP - Python Scientific Computing & Basic Plotting (Numpy & Matplotlib)

In this lab, we will study a new library in python that offers fast, memory efficient manipulation of vectors, matrices and tensors: **numpy**. We will also study basic plotting of data using the most popular data visualization libraries in the python ecosystem: **matplotlib**.

```
# Some IPython magic
# Put these at the top of every notebook, to get automatic reloading and inline pl
%reload_ext autoreload
%autoreload 2

import numpy as np
import matplotlib.pyplot as plt

!pip install scprep
```

```
Collecting scprep
  Downloading scprep-1.1.0-py3-none-any.whl (104 kB)
     |████████████████████████████████| 104 kB 10.2 MB/s
Requirement already satisfied: packaging in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: scipy>=0.18.1 in /usr/local/lib/python3.7/dist
Requirement already satisfied: numpy>=1.12.0 in /usr/local/lib/python3.7/dist
Requirement already satisfied: scikit-learn>=0.19.1 in /usr/local/lib/python3
Requirement already satisfied: pandas>=0.25 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: decorator>=4.3.0 in /usr/local/lib/python3.7/d
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/pytho
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: pyparsing>=2.0.2 in /usr/local/lib/python3.7/d
Installing collected packages: scprep
Successfully installed scprep-1.1.0
```

Python scientific computing ecosystem

For scientific computing we need an environment that is easy to use, and provides a couple of tools like manipulating data and visualizing results. Python is very easy to use, but the downside is that it's not fast at numerical computing. Luckily, we have very eficient libraries for all our use-cases.

Libraries

Core computing libraries

- `numpy` and `scipy`: scientific computing

- `matplotlib` : plotting library

Machine Learning

- `sklearn` : machine learning toolkit
- `tensorflow` : deep learning framework developed by google
- `keras` : deep learning framework on top of `tensorflow` for easier implementation
- `pytorch` : deep learning framework developed by facebook

Statistics and data analysis

- `pandas` : very popular data analysis library
- `statsmodels` : statistics

We also have advanced interactive environments:

- Ipython: advanced python console
- Jupyter: notebooks in the browser

There are many more scientific libraries available.

Check out these cheetsheets for fast reference to the common libraries:

Cheat sheets:

- python
- numpy
- matplotlib
- sklearn
- pandas

Other:

- Probabilities & Stats Refresher
- Algebra

numpy

`numpy` works with tensors of data, the main data structure is `numpy.array` or `numpy.ndarray` .

Why it is useful: Memory-efficient container that provides fast numerical operations.

Speed test

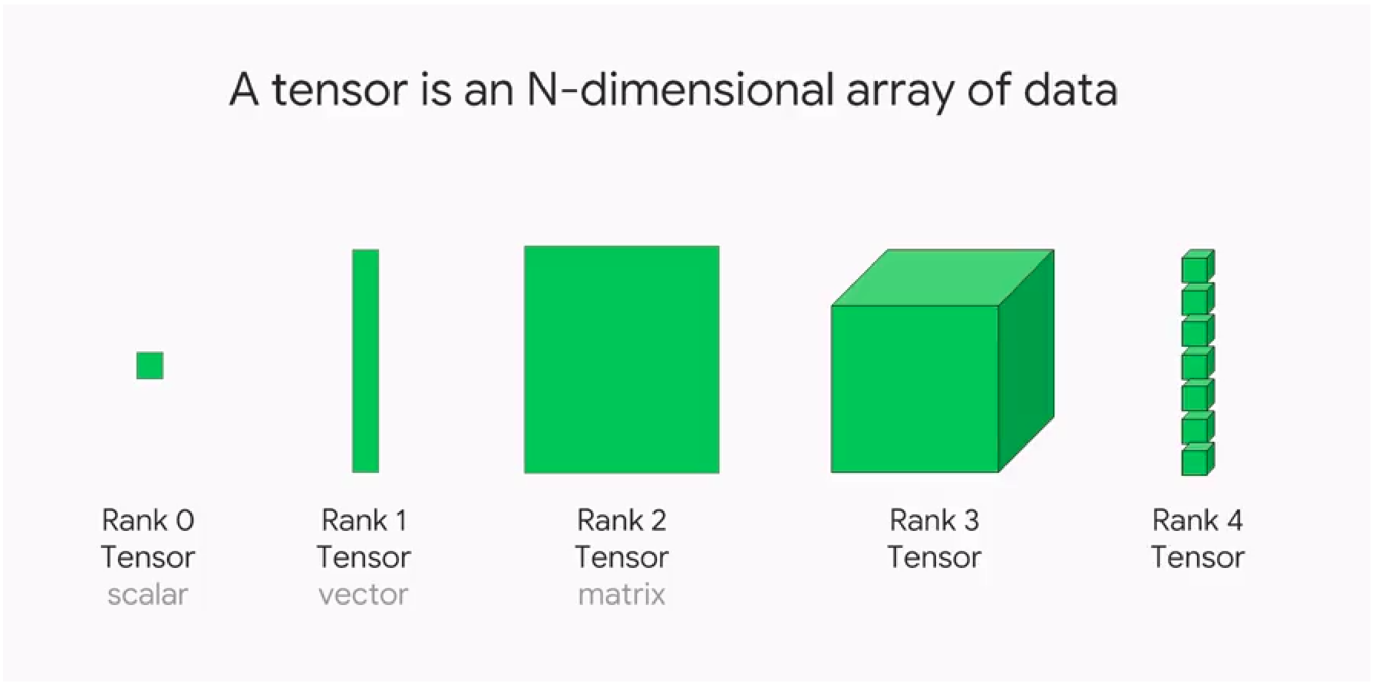
```
%timeit [x**2 for x in range(1000)]
```

```
1000 loops, best of 5: 273 µs per loop
```

```
%timeit np.arange(1000)**2
```

```
The slowest run took 1210.77 times longer than the fastest. This could mean t
100000 loops, best of 5: 3.09 µs per loop
```

It is clear that math operations using numpy arrays are far more efficient computation-wise than using plain python lists. More than that, numpy arrays offer a rich API for ***tensor*** manipulation.



Working with numpy arrays

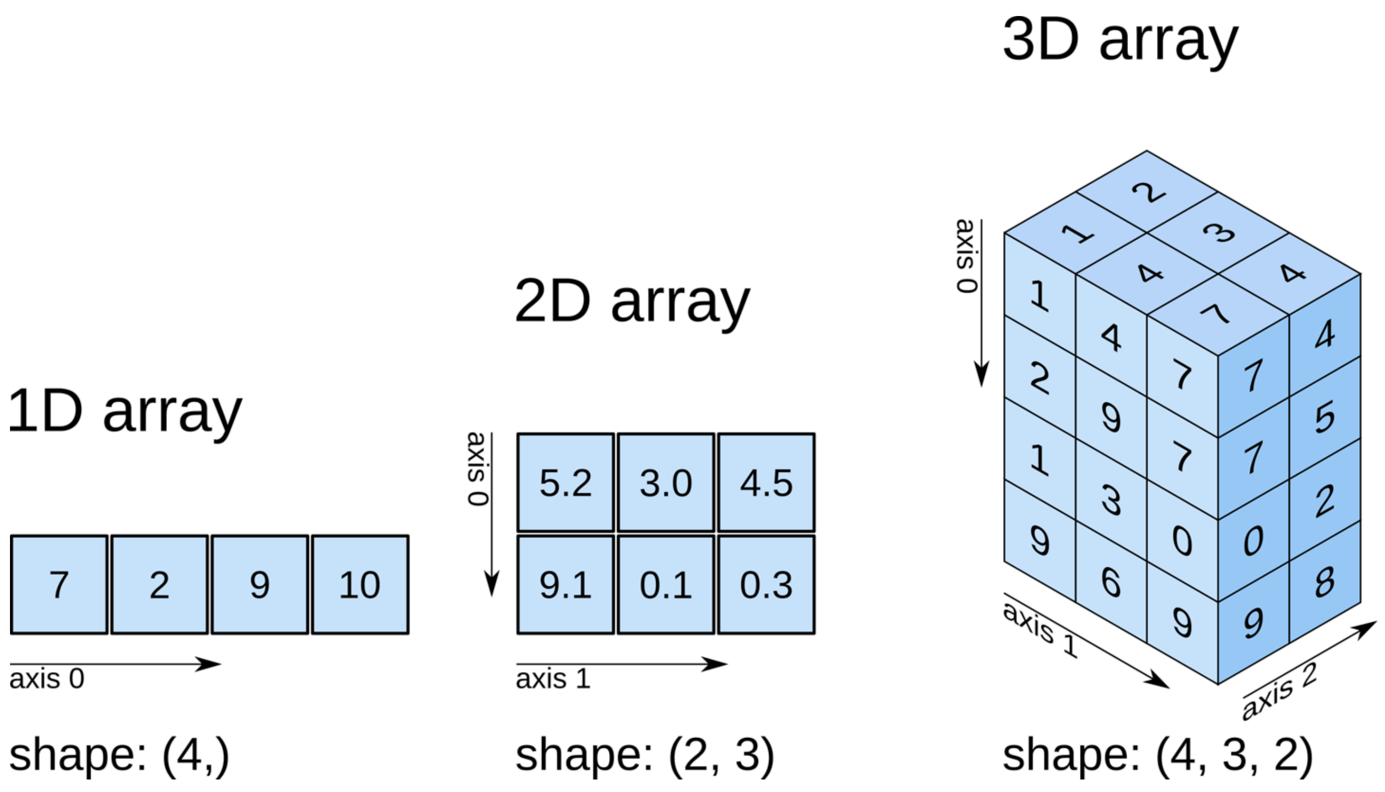
a few important attributes of the `array` structure:

- `ndarray.ndim` - number of axes (tensor rank)
- `ndarray.shape` - tuple, gives dimensions of each axes.
- `ndarray.size` - size of the array (product of elements of `ndarray.shape`)
- `ndarray.dtype` - data type

```
a = np.array([[2,7,5],[0,-1,2]])
print(a)
print("Dim:  ",a.ndim)
print("Shape:",a.shape)
print("Size: ",a.size)
print("Type: ",a.dtype)
```

```
[[ 2  7  5]
 [ 0 -1  2]]
Dim:   2
Shape: (2, 3)
Size:  6
Type:  int64
```

Shape Manipulation



```
x = np.array([1,1,2])
x.shape
```

```
    (3,)
```

```
x.reshape(1,3)
```

```
    array([[1, 1, 2]])
```

```
x.reshape(1,4) # this will fail because dimensions don't match
```

```
    ---------------------------------------------------------------------------
    ValueError                                Traceback (most recent call
    last)
    <ipython-input-7-72e7449de1fa> in <module>()
    ----> 1 x.reshape(1,4) # this will fail because dimensions don't match

    ValueError: cannot reshape array of size 3 into shape (1,4)
```

```
x.reshape(1,-1)
```

```
    array([[1, 1, 2]])
```

```
a = np.floor(10*np.random.random((3,4))) # some random matrix
a
```

```
    array([[1., 6., 1., 6.],
           [4., 4., 8., 1.],
           [6., 3., 6., 3.]])
```

```
a.shape # a tuple that describes the shape of the object
```

```
    (3, 4)
```

```
a.ravel() # flattened list
```

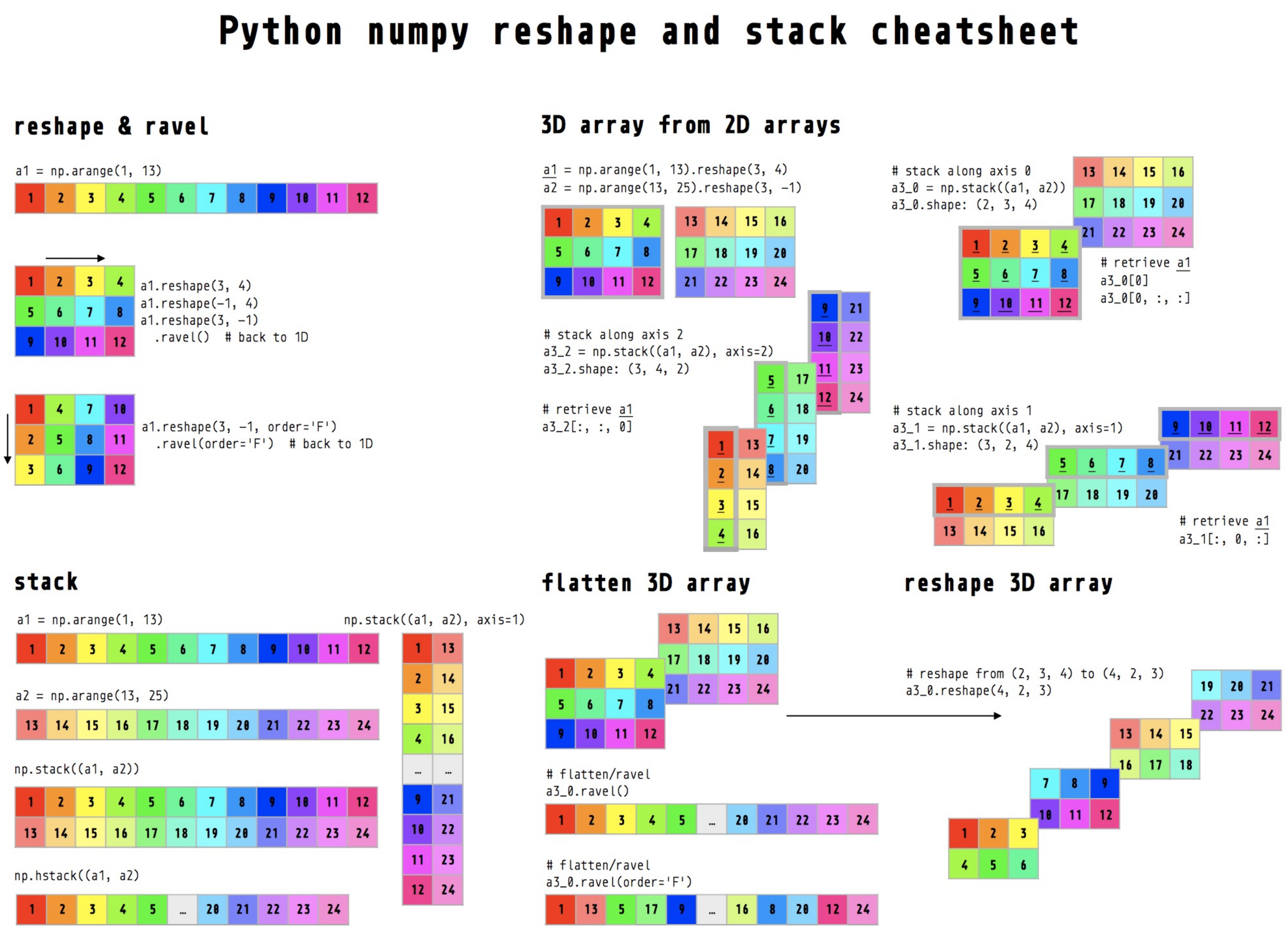
```
    array([1., 6., 1., 6., 4., 4., 8., 1., 6., 3., 6., 3.])
```

```
a.T # transpose
```

```
    array([[1., 4., 6.],
           [6., 4., 3.],
           [1., 8., 6.],
           [6., 1., 3.]])
```

```
a.reshape(2,1,-1) # -1 means "as much as you need", such that the dimensions mache
```

```
    array([[[1., 6., 1., 6., 4., 4.]],

           [[8., 1., 6., 3., 6., 3.]]])
```


Python numpy reshape and stack cheatsheet

reshape & ravel

```
a1 = np.arange(1, 13)
```

```
a1.reshape(3, 4)
a1.reshape(-1, 4)
a1.reshape(3, -1)
  .ravel()  # back to 1D
```

```
a1.reshape(3, -1, order='F')
  .ravel(order='F')  # back to 1D
```

3D array from 2D arrays

```
a1 = np.arange(1, 13).reshape(3, 4)
a2 = np.arange(13, 25).reshape(3, -1)
```

```
# stack along axis 0
a3_0 = np.stack((a1, a2))
a3_0.shape: (2, 3, 4)
```

```
# retrieve a1
a3_0[0]
a3_0[0, :, :]
```

```
# stack along axis 2
a3_2 = np.stack((a1, a2), axis=2)
a3_2.shape: (3, 4, 2)
```

```
# retrieve a1
a3_2[:, :, 0]
```

```
# stack along axis 1
a3_1 = np.stack((a1, a2), axis=1)
a3_1.shape: (3, 2, 4)
```

```
# retrieve a1
a3_1[:, 0, :]
```

stack

```
a1 = np.arange(1, 13)
```

```
a2 = np.arange(13, 25)
```

```
np.stack((a1, a2))
```

```
np.hstack((a1, a2))
```

```
np.stack((a1, a2), axis=1)
```

flatten 3D array

```
# flatten/ravel
a3_0.ravel()
```

```
# flatten/ravel
a3_0.ravel(order='F')
```

reshape 3D array

```
# reshape from (2, 3, 4) to (4, 2, 3)
a3_0.reshape(4, 2, 3)
```

Initializing

- `np.zeros(shape)`
- `np.ones(shape)`
- `np.empty(shape)`
- `np.eye(rows, cols)`

Those functions take as input the shape of an array.

Sequences

- `np.arange` - like Python's `range()`, but returns a `np.array`
- `np.linspace` - n equidistant points in interval [a,b]

```
np.arange(5)

    array([0, 1, 2, 3, 4])
```

```
np.linspace(1, 2, 5)

    array([1.  , 1.25, 1.5 , 1.75, 2.  ])
```

Math operators

Math operators are applied elementwise. (`*` is never the dot product for numpy arrays!)

```
a = np.ones((2,3), dtype=int)
a *= 3
a

    array([[3, 3, 3],
           [3, 3, 3]])
```

```
a = a - 1
a

    array([[2, 2, 2],
           [2, 2, 2]])
```

```
a = a + np.ones_like(a) / 5
a

    array([[2.2, 2.2, 2.2],
           [2.2, 2.2, 2.2]])
```

```
print("Pointwise multiplication:\n", a * np.ones_like(a) * 2 )
print("Dot Product (matrix multiplication)\n", np.dot(a.T, np.ones_like(a) * 2))
```

```
Pointwise multiplication:
 [[4.4 4.4 4.4]
 [4.4 4.4 4.4]]
Dot Product (matrix multiplication)
 [[8.8 8.8 8.8]
 [8.8 8.8 8.8]
 [8.8 8.8 8.8]]
```

Linear Algebra

numpy has implementations of some of the most common algebraic operations. The most common one, the dot product is implemented both as a method of the array object and as a numpy function.

Matrix multiplication can be performed either using the dot product or `np.matmul`. Using functions from numpy you can

- find eigenvalues and eigenvectors,
- compute some decompositions (QR, SVD),
- compute determinant of a matrix,
- norm of a vector,
- solve a linear system
- invert a matrix

See the documentation

Matplotlib

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. Almost all python plotting frameworks make use of matplotlib behind the scenes.

Plotting data is an effective way to convey the message hidden in the data. A good plot is worth a thousand words, and it can be difficult to unambiguously transmit your intended message. Different types of plots that represent the same data (i.e. scatter plots, linear plots, bar plots, pie charts etc.) can be perceived differently depending on who's looking.

Check out this article for some tips on better plotting:

Ten Simple Rules for Better Figures

NB: Do note that, regardless of the type of plot and data, ***all*** plots must have properly annotated axis ticks, axis labels, a title and a caption.

For a more comprehensive tutorial on matplotlib, check out this resource:

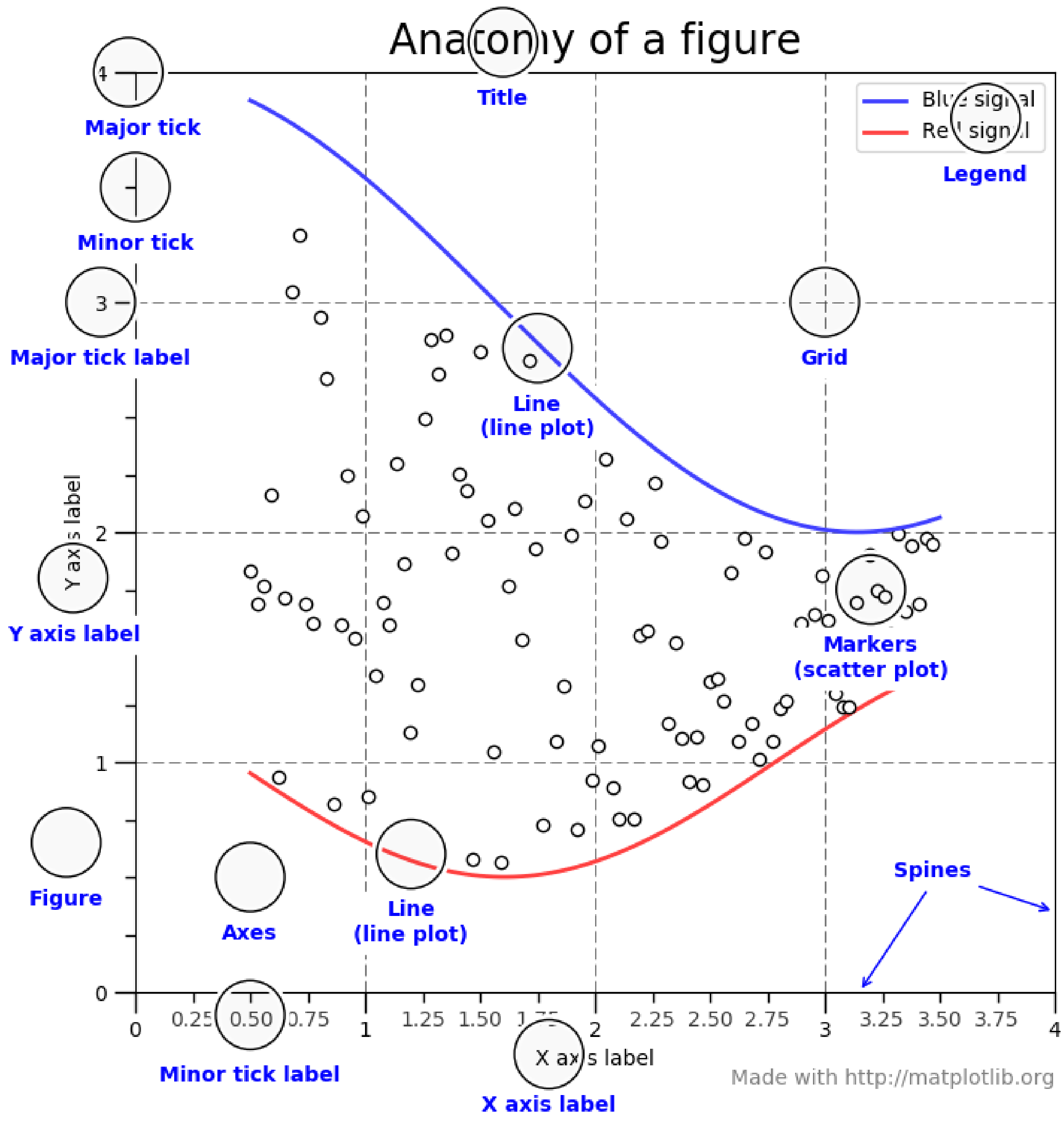
Matplotlib Tutorial – A Complete Guide to Python Plot w/ Examples

```
import matplotlib.pyplot as plt
%matplotlib inline
```

The elements of a plot

The figure and the axes

The figure is the entire image, the individual plots are called "axes". Here's a description from matplotlib's documentation.

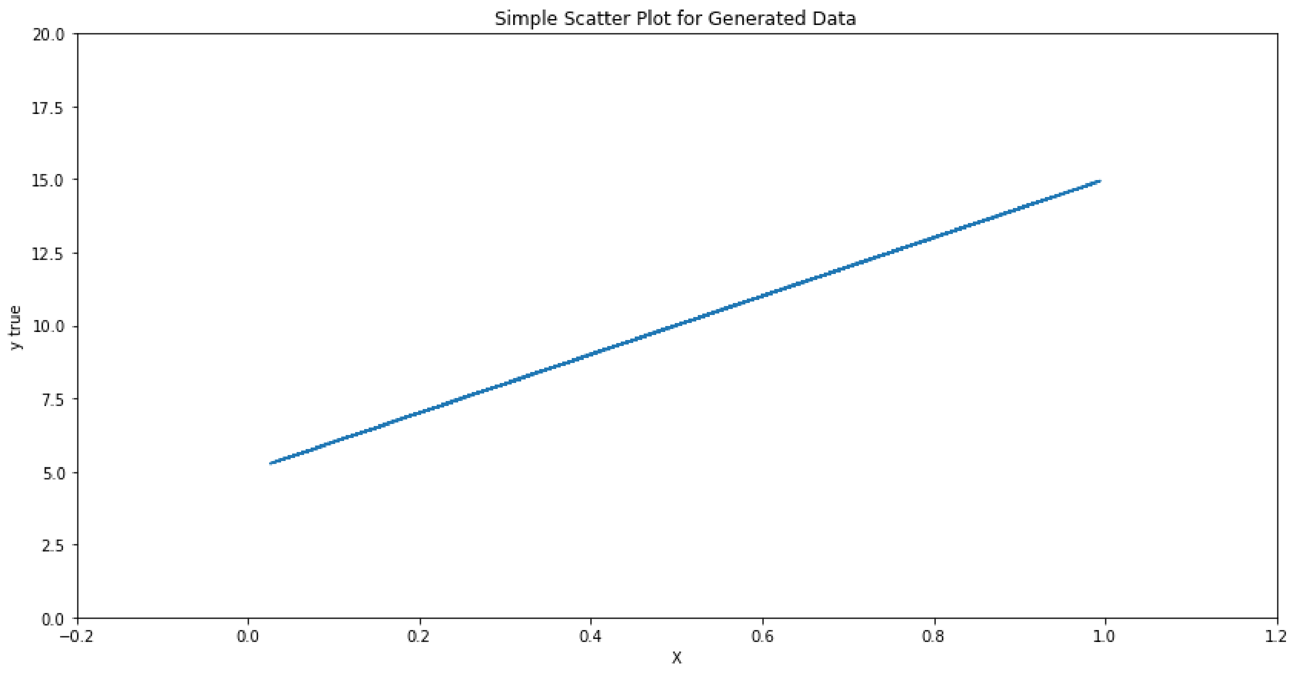


Scatter & Line Plots

Scatter plots are used to plot data points on horizontal and vertical axis in the attempt to show how much one variable is affected by another. Each row in the data table is represented by a marker the position depends on its values in the columns set on the X and Y axes. A third variable can be set to correspond to the color or size of the markers, thus adding yet another dimension to the plot.

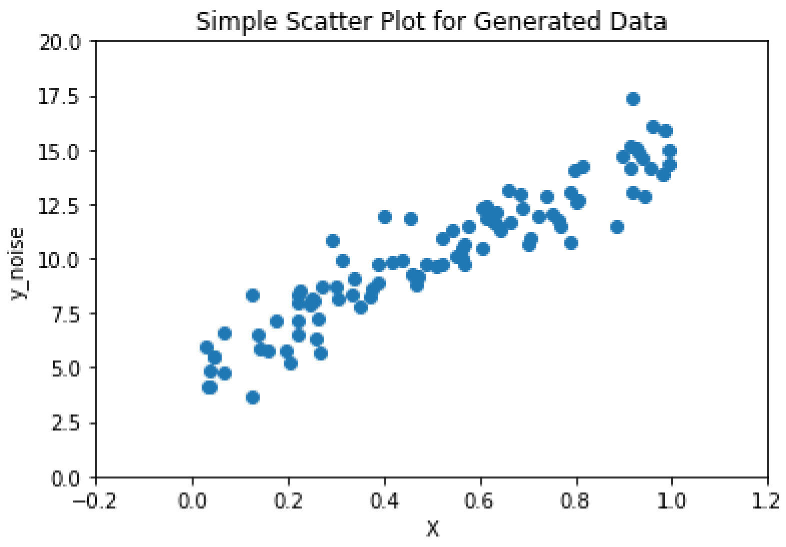
We will plot a simple regression line with generated data.

```
M = 10
N = 5

X = np.random.random(size = 100)
y_true = M * X + N

y_noise = y_true + np.random.normal(size = 100)


plt.plot(X, y_true)
plt.xlim(-0.2, 1.2)
plt.ylim(0, 20)

plt.xlabel("X")
plt.ylabel("y true")

plt.title("Simple Scatter Plot for Generated Data")
plt.gcf().set_size_inches(14, 7)
```



```
plt.scatter(X, y_noise)
plt.xlim(-0.2, 1.2)
plt.ylim(0, 20)

plt.xlabel("X")
plt.ylabel("y_noise")

plt.title("Simple Scatter Plot for Generated Data")
```

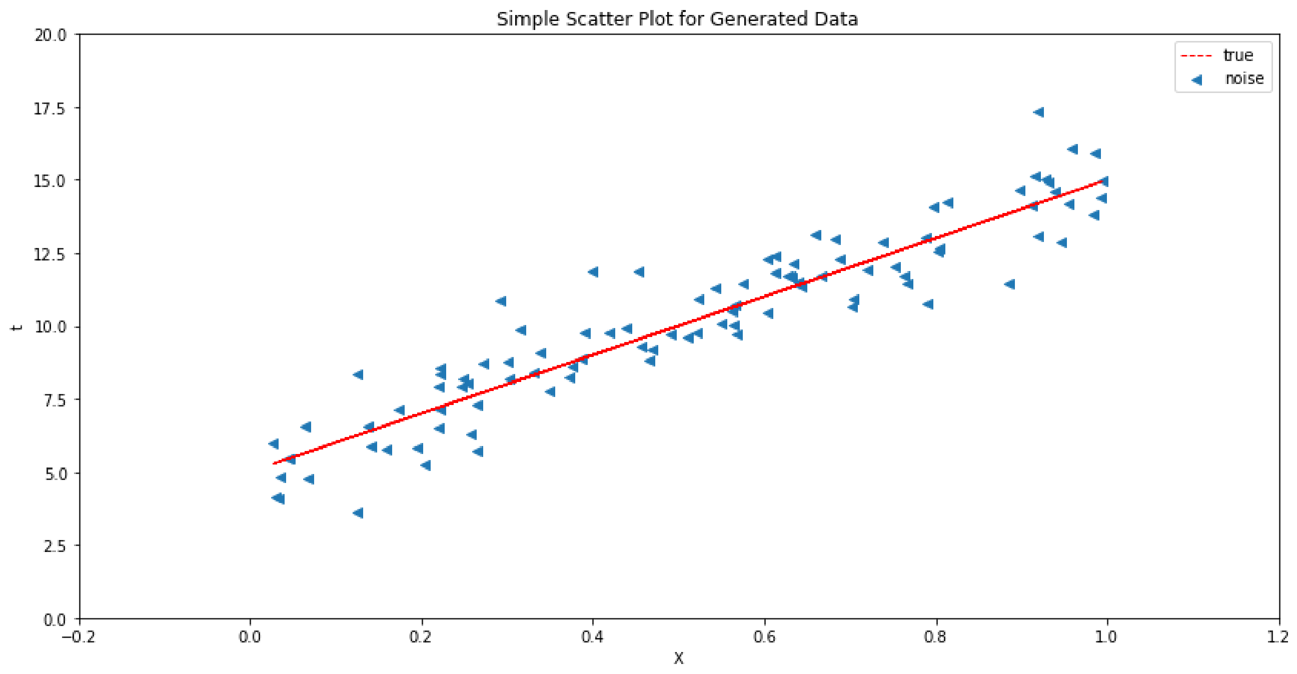
```
    Text(0.5, 1.0, 'Simple Scatter Plot for Generated Data')
```



We can plot the two variables one on top of the other, for a better visualization. Make sure to add a legend, and different coloring, such that it is clear which part of the plot belongs to which set of variables.

Matplotlib also offers a wide range of **markers** to better differentiate between multiple data categories. Check https://matplotlib.org/api/markers_api.html for a full list.

```
plt.scatter(X, y_noise, label = 'noise', marker = '<')
plt.plot(X, y_true, color = 'r', label = 'true', linestyle = 'dashed', linewidth =

plt.xlim(-0.2, 1.2)
plt.ylim(0, 20)

plt.xlabel("X")
plt.ylabel("t")

plt.title("Simple Scatter Plot for Generated Data")
plt.legend()
plt.gcf().set_size_inches(14, 7)
```

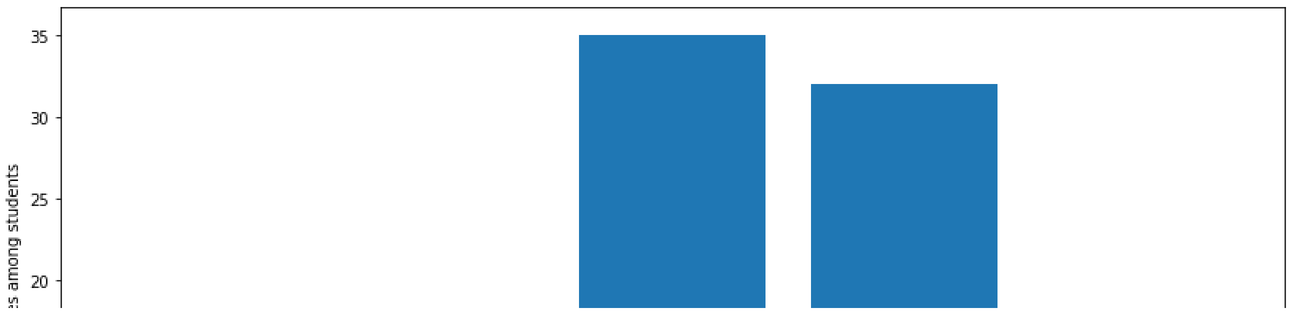


Bar Plots

A bar chart or bar graph is a chart or graph that presents categorical data with rectangular bars with heights or lengths proportional to the values that they represent. The bars can be plotted vertically or horizontally.

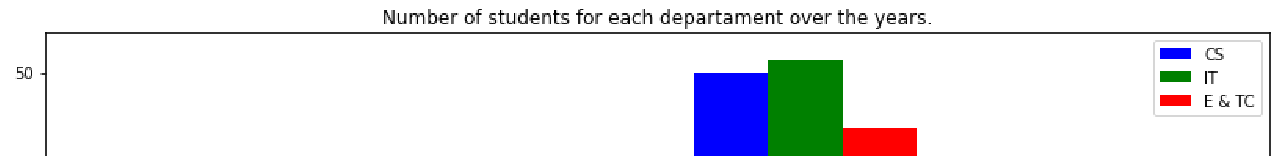
A bar graph shows comparisons among discrete categories. One axis of the chart shows the specific categories being compared, and the other axis represents a measured value.

```
languages = ['C', 'C++', 'Java', 'Python', 'PHP']
students = [10, 12, 35, 32, 5]

ax = plt.subplot()
ax.bar(languages, students)

plt.xlabel('Programming Language')
plt.ylabel("Number of favorites among students")
plt.gcf().set_size_inches(14, 7)
```



We can make multiple bars for each category. It makes it easier for comparing different quantities with a common category.

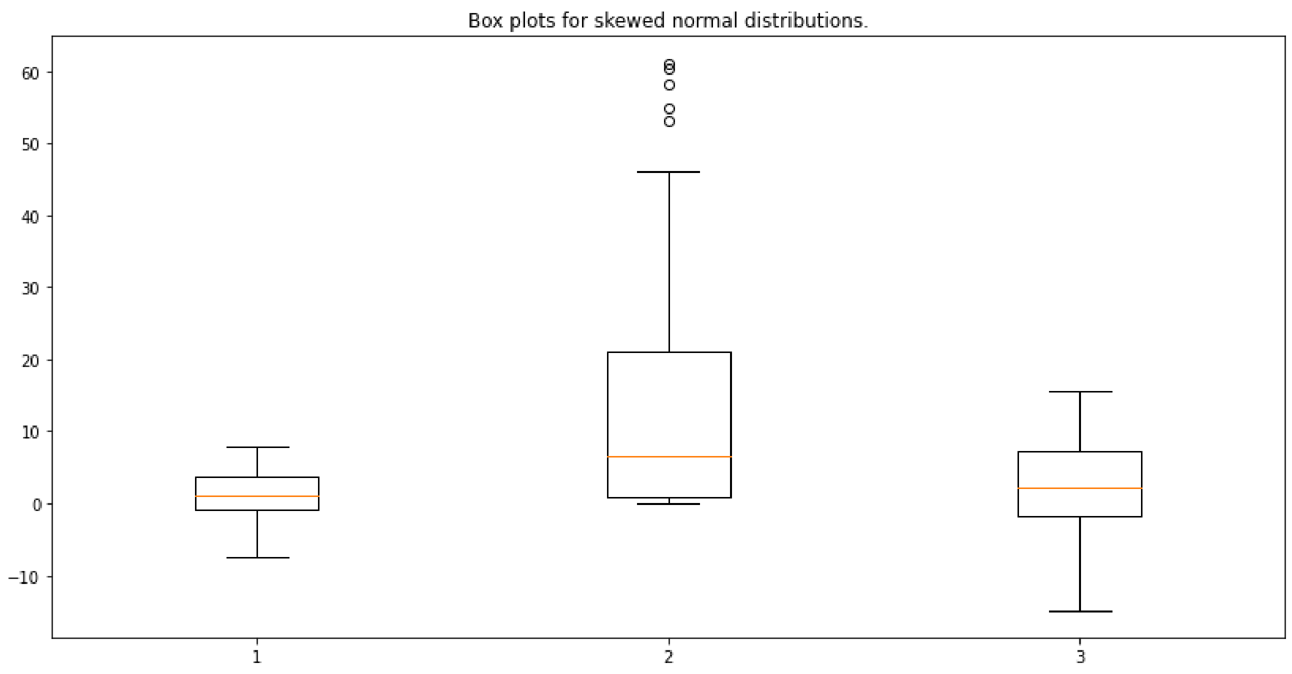
```
data = [
  [30, 25, 50, 20],
  [40, 23, 51, 17],
  [35, 22, 45, 19]
]

X = np.arange(4)
ax = plt.subplot()

ax.bar(X + 0.00, data[0], color = 'b', width = 0.25, label = 'CS')
ax.bar(X + 0.25, data[1], color = 'g', width = 0.25, label = 'IT')
ax.bar(X + 0.50, data[2], color = 'r', width = 0.25, label = 'E & TC')

ax.set_xticks(X + 0.25)
ax.set_xticklabels(X + 2017)

plt.xlabel("Year")
plt.ylabel("Number of students")
plt.title("Number of students for each departament over the years.")
ax.legend()
plt.gcf().set_size_inches(14, 7)
```



Box Plots

A boxplot is a standardized way of displaying the distribution of data based on a five number summary (“minimum”, first quartile (Q1), median, third quartile (Q3), and “maximum”). It can tell you about your outliers and what their values are. It can also tell you if your data is symmetrical, how tightly your data is grouped, and if and how your data is skewed.

```
data = np.random.normal(size = 100, scale = 3.0, loc = 1.0)

ax = plt.subplot()
ax.boxplot([data, data ** 2, data * 2])

plt.title("Box plots for skewed normal distributions.")
plt.gcf().set_size_inches(14, 7)
```



Exercises

After you finish your lab exercises, you should export this notebook as **pdf** and upload it to Moodle. (i.e. **File -> Print**, Destination: Save as PDF).

1. System of linear equations

Using numpy, solve the following linear equation:

$$\begin{bmatrix} 1 & 2 & 3.3 & 2 \\ 3 & 3.6 & 7 & 0 \\ 1 & 3 & -1 & 12 \\ 2 & 11 & 4 & 16 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 10 \\ 15 \\ 3 \\ -1 \end{bmatrix}$$

```
# Your code here
A = np.array([[1, 2, 3.3, 2], [3, 3.6, 7, 0], [1, 3, -1, 12], [2, 11, 4, 16]])
B = np.array([10, 15, 3, -1])

print(np.linalg.solve(A,B))
```

```
[-1.28937298 -4.34942596  4.9322932   1.85582867]
```

2. Simple numpy exercises

1. Given an matrix, calculate the sum for each row
2. Given an vector, normalize the vector using the l_{10} norm

```
M = np.array([
    [1, 2, 3],
    [9, 3, 10],
    [3, 5, 0],
    [1, 7, -3],
    [0, -2, 3],

])

v = np.array([100, 2302, 2, -10, 134, -1])

# Your code here
# 1
print(np.sum(M,axis=1))

# 2
norm = np.linalg.norm(v)
normal_v = v/norm
print(normal_v)
```

```
[ 6 22  8  5  1]
[ 4.33259256e-02  9.97362807e-01  8.66518512e-04 -4.33259256e-03
  5.80567403e-02 -4.33259256e-04]
```

3. Harder numpy exercises

Using the matrix and vector defined above:

1. Subtract the mean of each column in a given matrix.
2. Get the 3rd largest value in an array.

```
# Your code here

# 1
row_means = M.mean(axis=0)
print(row_means)

# 2
v.sort()
print(v[-3])
```

```
    [2.8 3.  2.6]
    100
```

4. Plot some functions

Using numpy and matplotlib, plot the following functions (you can choose the colors and styling, interval of interest and other parameters):

$f_1(x) = max(0, x)$

$f_2(x) = \frac{e^{2x}-1}{e^{2x}+1}$

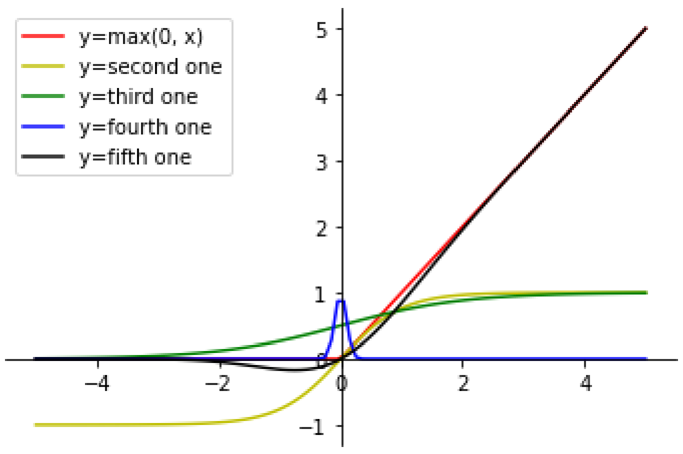
$f_3(x) = \frac{1}{1+e^{-x}}$

$f_4(x) = \frac{1}{\mu^2+1} * e^{-\frac{(x-\mu)^2}{b^2}}$

$f_5(x) = 0.5x(1 + tanh(0.797885x + 0.035677x^3))$

Note that all plots should have relevant ticks, labels and a title.

```
# Your code here
import scipy.constants as scipy

x = np.linspace(-5,5,100)
y1 = np.maximum(0, x)
y2 = (np.exp(2 * x) - 1) / (np.exp(2 * x) + 1)
y3 = 1 / (1 + np.exp(-1 * x))
### Who is b ??
y4 = 1 / (scipy.mu_0 ** 2 + 1) * np.exp((-1 * ((x - scipy.mu_0) ** 2 ) / (0.134534
y5 = 0.5 * x * (1 + np.tanh(0.797885 * x + 0.035677 * (x ** 3)))

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.xaxis.set_ticks_position('bottom')
ax.yaxis.set_ticks_position('left')
ax.spines['left'].set_position('center')
ax.spines['bottom'].set_position('zero')
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')

plt.plot(x,y1, 'r', label='y=max(0, x)')
plt.plot(x,y2, 'y', label='y=second one')
plt.plot(x,y3, 'g', label='y=third one')
plt.plot(x,y4, 'b', label='y=fourth one')
plt.plot(x,y5, 'k', label='y=fifth one')


plt.legend(loc='upper left')
plt.show()
```



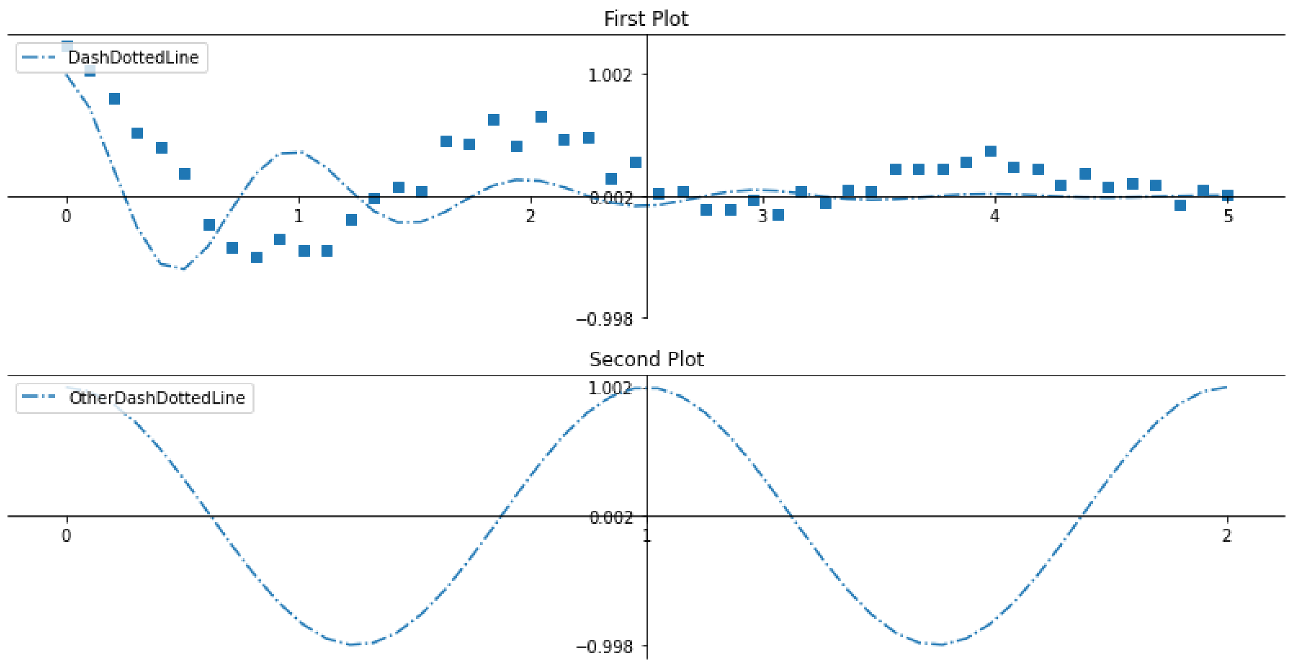
5. Prettify Plot

Given the below plot, make it look presentable. Add **square** markers to scatter plots, **dashdotted** lines, add relevant axis **limits** and **ticks**, a **legend** and a **title**.

NB: A plot should be self-contained. Any reader should understand it without have to refer to its text description.

```
# Data Synthetization
x1 = np.linspace(0.0, 5.0)
x2 = np.linspace(0.0, 2.0)

y1 = np.cos(2 * np.pi * x1) * np.exp(-x1)
y2 = np.cos(2 * np.pi * x2)
y3 = np.cos(1 * np.pi * x1) * np.exp(-0.5 * x1) + 0.3 * np.random.random(size = x1

# Plotting code
# Took some "artistic liberties"

fig, (ax1, ax2) = plt.subplots(2, 1)

ax1.plot(x1, y1, '-.', label="DashDottedLine")
```

```
ax1.title.set_text('First Plot')
ax1.scatter(x1, y3, marker="s")
ax1.get_xaxis().set_ticks(np.arange(min(x1), max(x1)+1, 1.0))
ax1.get_yaxis().set_ticks(np.arange(min(y2), max(y2)+1, 1.0))
ax1.xaxis.set_ticks_position('bottom')
ax1.yaxis.set_ticks_position('left')
ax1.spines['left'].set_position('center')
ax1.spines['bottom'].set_position('zero')
ax1.spines['right'].set_color('none')
ax1.spines['top']
ax1.legend(loc='upper left')


ax2.plot(x2, y2, '-.', label="OtherDashDottedLine")
ax2.title.set_text('Second Plot')
ax2.get_xaxis().set_ticks(np.arange(min(x2), max(x2)+1, 1.0))
ax2.get_yaxis().set_ticks(np.arange(min(y2), max(y2)+1, 1.0))
ax2.xaxis.set_ticks_position('bottom')
ax2.yaxis.set_ticks_position('left')
ax2.spines['left'].set_position('center')
ax2.spines['bottom'].set_position('zero')
ax2.spines['right'].set_color('none')
ax2.spines['top']
ax2.legend(loc='upper left')

plt.gcf().set_size_inches(14, 7)
```



6. Plotting a dataset

Plot different aspects of the Wine Dataset from sklearn. These data are the results of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars.

6.1 Create a plot for visualizing the distribution of one features (e.g. alchohol level / malic acid / ash etc.) for each class of wine.

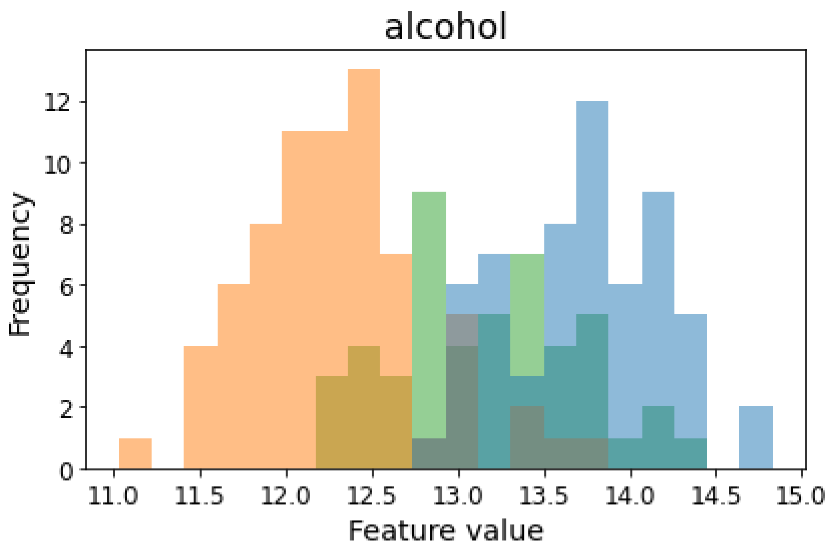
6.2 Create 3 subplots highlighting different aspects of the dataset. Choose the appropriate plot types (scatter plot, line plot, bar plot, box plot etc.). You can choose various combinations of columns to gain a better understanding of this dataset.

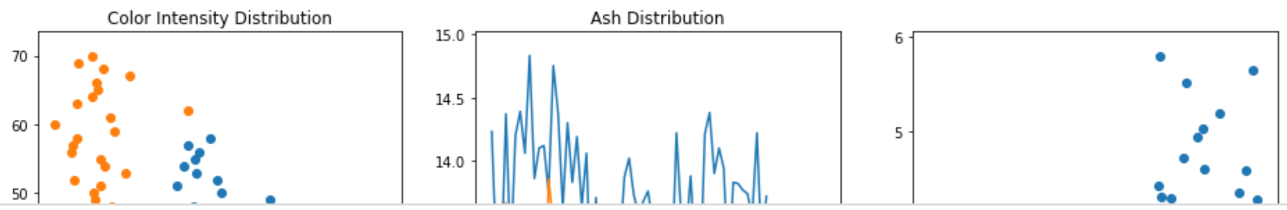
Be creative! You can color your points using the `target` array. Remember to always have labels on your axes, appropriate ticks, a legend where necessary, and a plot title

```
#Source of inspiration: https://colab.research.google.com/github/KrishnaswamyLab/S
from sklearn.datasets import load_wine
import pandas as pd
import scprep

wine_dataset = load_wine()

data = wine_dataset['data']
target = wine_dataset['target']

column_names = wine_dataset['feature_names']

# Load cultivar information about each wine
cultivars = np.array(['Cultivar{}'.format(cl) for cl in wine_dataset['target']])

# Create nice names for each row
wine_names = np.array(['Wine{}'.format(i) for i in range(data.shape[0])])

# Gather all of this information into a DataFrame
data = pd.DataFrame(data, columns=column_names, index=wine_names)


### Features that differentiate two cultivars
feature = data['alcohol']
# Desired cultivars here
a_group = 'Cultivar0'
b_group = 'Cultivar1'
c_group = 'Cultivar2'
scprep.plot.histogram([feature[cultivars == a_group], feature[cultivars == b_group
                      title=feature.name, bins=20,
                      xlabel='Feature value',ylabel='Frequency')


### Color intensity variation for wines of diffrent cultivars
feature = data['color_intensity']
y1 = np.arange(feature[cultivars == a_group].size)
y2 = np.arange(feature[cultivars == b_group].size)
y3 = np.arange(feature[cultivars == c_group].size)

fig, (ax1, ax2, ax3) = plt.subplots(1, 3)
ax1.title.set_text('Color Intensity Distribution')
ax1.scatter(feature[cultivars == a_group], y, label="OtherDashDottedLine")
```

```
ax1.scatter(feature[cultivars == b_group], y2, label="OtherDashDottedLine")
ax1.scatter(feature[cultivars == c_group], y3, label="OtherDashDottedLine")

### Ash variation for wines of diffrent cultivars accross wines
feature = data['alcohol']
ax2.title.set_text('Ash Distribution')
ax2.plot(y, feature[cultivars == a_group], label="OtherDashDottedLine")
ax2.plot(y2, feature[cultivars == b_group], label="OtherDashDottedLine")

### Malic acid of each class of wine FOR 6.1
plt.scatter(wine_names, data['malic_acid'])

plt.gcf().set_size_inches(16, 7)
```

```
/usr/local/lib/python3.7/dist-packages/numpy/core/_asarray.py:83: VisibleDepr
  return array(a, dtype, copy=False, order=order)
```





✓ 0s completed at 11:05 AM