



MatchIT: Python for Data Science

Lecture: NumPy

Lecture 1: Content

- About the course
- Review of Programming Concepts
- Introduction to Data Analysis
- NumPy: Overview

Course

- Python for Data Science course web page in Moodle:
 - MatchIT_2019/20: Python for Data Science
 - Enrollment key: py_DS
- Duration 3 weeks (Week 1: June 03, Week 2: June 10, Week 3: June 17)
 - Check Moodle Calendar for any changes in Lecture and Lab dates
 - **Lectures:** Two occasions a week on Mondays and Tuesdays 13:00-15:00, except for the first week when the lectures will be held Monday and Wednesday 13:00-15:00
 - **Labs:** One lab per week, labs due at the end of each week

Week 1	Week 2	Week 3
M, Tue, W : 15-17	M, Tue : 15-17	M, Tue : 15-17
F: 13-15	Th, Fr : 13-15	W, Th: 13-15

- **Quiz:**
 - Week 1 and Week 2: Friday, 10:00 – 11:30
 - Week 3: Thursday, 10:00-11:30

Course Personnel

- Teachers:
 - Lecturer Dr. Alma Orucevic-Alagic (office hours M, T: 10:30-11:30)
 - Prof. Dr. Martin Höst
- Student Assistents:
 - Jesper Grahm
 - Mikael Murstam
 - Viktor Claeson

Literature

- Book: Python Data Science Handbook (online)
 - <https://jakevdp.github.io/PythonDataScienceHandbook/index.html>
- Labs and accompanying materials will be added each week to the course page in Moodle

Code of Conduct

- Come to lectures on time, to minimize disruptions. Course attendance is taken at the beginning of each class.
- Lab attendance is noted. There will be 2 to 3 assistants available at each lab session – ask questions if you get stuck!
- Each student is expected to participate. Everyone is assigned a number that will be called to answer a question by selection via random number generated.
- If you do not understand – ask and speak up! We are here to learn.

Three week plan

- Week 1: NumPy review
- Week 2: Pandas
- Week 3: Machine learning – Linear Regression

Review of Basic Programming Concepts in Python

- Variable
- Data type
- Control structures:
 - If statements
 - For loops,
 - While Loop
 - Nested Loop
- Data and operations:
 - Functions
 - List (append, pop, sort, range)
 - Slices
 - Variable Scope
- Object orientation

Data Analysis: Introduction

- The process of extracting information from raw data
- Domain: Biology, Physics, Finance, Population Statistics....
- Aspects:
 - Data Domain
 - Computer Science
 - Mathematics and Statistics
 - Machine Learning and Artificial Intelligence

Raw Data → Information → Knowledge

- Types of Data
 - Categorical (nominal and ordinal)
 - Numerical (discrete and continuous)

Data Analysis: The Process

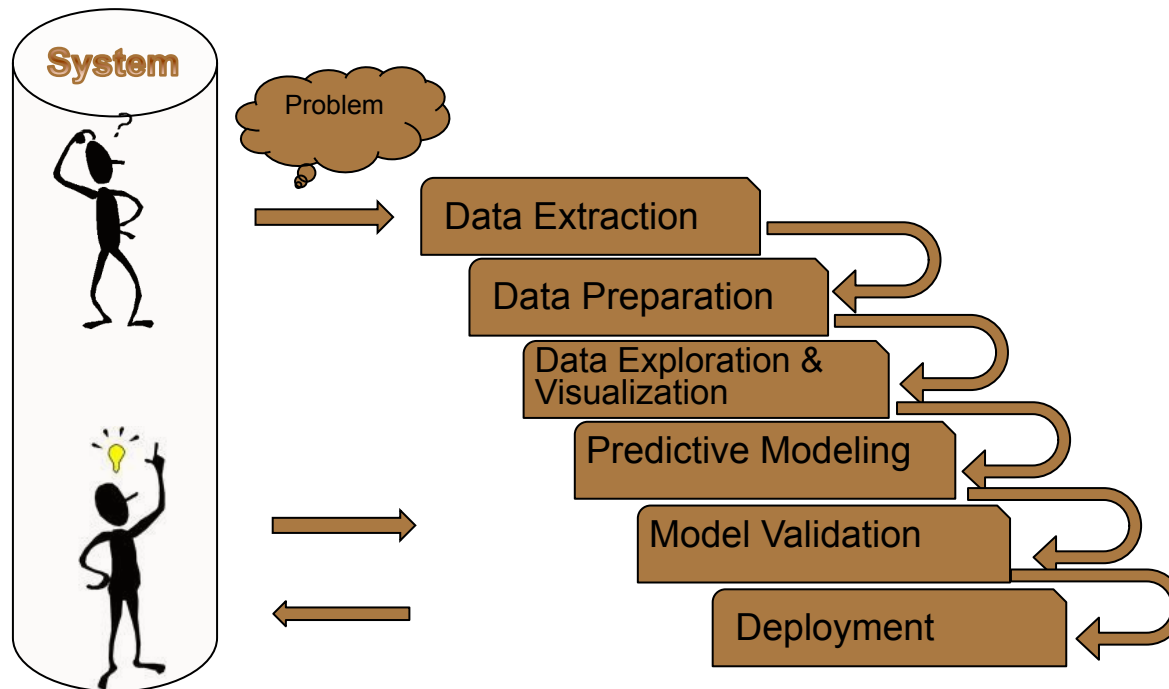


Figure 1: Data Analysis Process

Predictive Modeling

- Predictive modeling is important step in data analysis and it is a process concerned with creation or selection of an appropriate statistical model to predict the probability of the result.
- Pre-requirements: Data Extraction, Data Preparation, Data Exploration & Validation steps of Data Analysis process have been completed.
- Division is based on results produced:
 - *Classification models*: The result produced is **categorical**.
 - *Regression models*: The result produced is **numeric**.
 - *Clustering models*: The result produced is **descriptive**.
- Purpose:
 1. To make predictions about the data values produced by the system (Regression models)
 2. To classify (Classification models or clustering models)

Predictive Modeling: Continued

- Some methods used to generate the predictive models:
 - Linear Regression, Logistic Regression, Classification and Regression Trees, K-nearest neighbors
 - Many analysis methods, important to understand which of the methods' characteristics best fit a desired analysis of a dataset, i.e. each method produces a specific predictive model.
- Some models explain characteristics of a studied system, others can provide predictions.

Model Validation and Deployment

- A set of data used to build the model is called *training set*, while dataset used to validate the model is called *validation set*.

Data Produced by the Model == Data Produced by the System ???

- Evaluate error and estimate the limits of validity of the generated model.
- Deployment: Presentation of results of the model, e.g. value for business client, design solution, scientific publication...

Quantitative and Qualitative Data Analysis

- Quantitative Data Analysis suitable for numeric and categorical data structures.
- Quantitative Data Analysis suitable for descriptive/natural language data
- Open Data Data Sources:
 - World Health Organization (<http://www.who.int/research/en>)
 - EU Open Data Portal (<http://open-data.europa.eu/en/data/>)
 - FB Graph (<http://developers.facebook.com/docs/graph-api>)
 - many more....
- Python contains large number of libraries that provide a complete set of tools for data analysis and manipulation.

SciPy

- Set of Open Source Python Libraries for Scientific Computing
- Set of Tools for calculating and displaying data
- We will be working with:
 - NumPy
 - Pandas

NumPy: Numerical Python

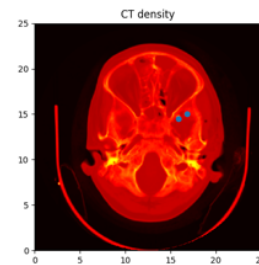
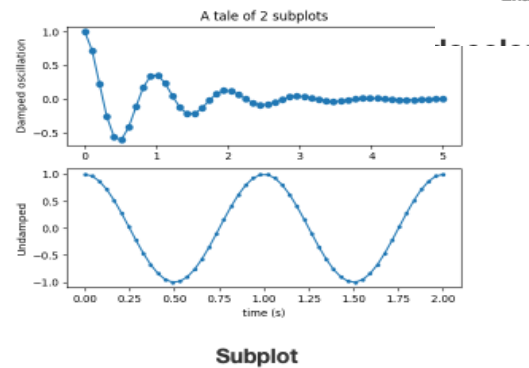
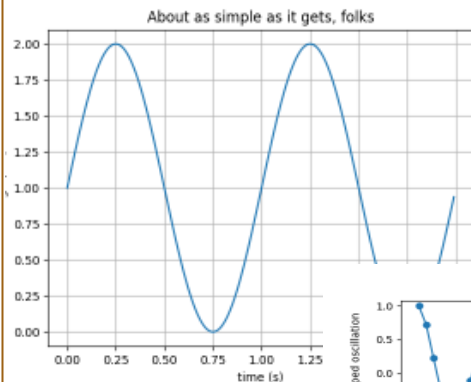
- NumPy: Foundation Library for Scientific Computing
- Correct usage of this library can significantly improve performance
- Some of the most significant NumPy features:
 - Ndarray: A more efficient multidimensional array
 - Element-wise computation: A set of functions for arrays
 - Reading-writing datasets
 - Integration with other languages (C, C++, Fortran..)

Pandas

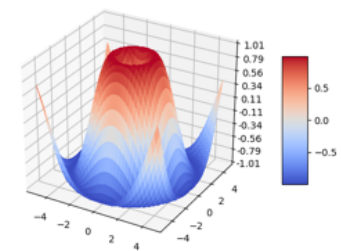
- Core data analysis package in Python
- Study of this package is the main goal of this part of the course
- *DataFrame*, a two-dimensional tabular data structure with row and column labels.
- Pandas utilizes NumPy library to manipulate data in spreadsheets or relational databases.
- Uses sophisticated indexing to manipulate data structure:
 - Reshaping
 - Slicing
 - Aggregation
 - Selection of subsets

matplotlib

- Python library package to produce plots and other 2D, 3D data visualizations



Example of using `imshow()` to display a CT scan



Surface3d

NumPy : ndarray and dtype

- *ndarray*: Multi (*N*) *dimensional array*
 - Homogeneous: all of the items in ndarray are *virtually* of the same type and size
- *dtype*: *Data Type Object*
 - belongs to NumPy library also, and specifies which type of data within the ndarray. (data-type)
- *shape*:
 - Specifies the number of the dimensions and items in an array
- *axes*:
 - dimensions
- *number of axes*:
 - rank

```
a = np.array([1,2,3])
```

```
: a
: array([1, 2, 3])
```

```
: type(a)
: numpy.ndarray
```

```
: a.dtype
: dtype('int64')
```

```
: a.ndim
: 1
```

```
: a.size
: 3
```

```
: a.shape
: (3,)
```

```
: b = np.array(['one','two','three'],['four', 'five', 'six'])
```

```
: type(b)
: numpy.ndarray
```

```
: b.dtype
: dtype('<U5')
```

```
: b.ndim
: 2
```

```
: b.size
: 6
```

```
: b.shape
: (2, 3)
```

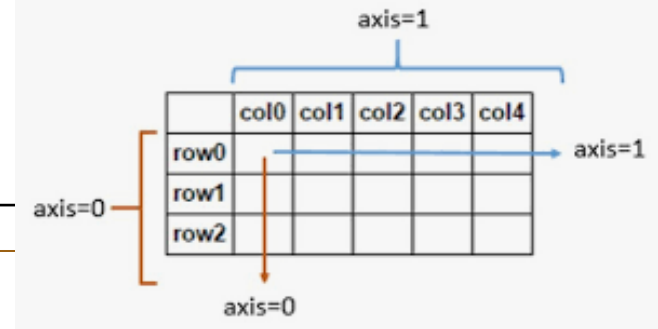
NumPy : Slicing and Iteration

- *Slicing* – Extracting portion of an array to generate new arrays , e.g `A[0,:]` , `A[0:2,0:2]` ...
 - In NumPy slicing returns a view of the original array. If values of some elements within the original array are changed, and the elements are included in the sliced view, the view will reflect the changes.
- *Iteration* – Traversing elements within array

```
In [76]: for row in a:  
        ...:     print(row)  
        ...:  
[10 11 12]  
[13 14 15]  
[16  1 18]
```

```
In [79]: for item in a.flat:  
        ...:     print(item)  
        ...:  
10  
11  
12  
13  
14  
15  
16  
1  
18
```

NumPy: Aggregate function



- *Aggregate function:*
- Used when we need to operate on a column, row, or individual elements:
 - *apply_along_axis()*
 - takes three argumentes (*function_name*, *row/column-1/1 indicator*, *array*)

• Example 1: Using mean

```
: a=np.arange(0,20).reshape(4,5)
: a
:
[ 0,  1,  2,  3,  4],
[ 5,  6,  7,  8,  9],
[10, 11, 12, 13, 14],
[15, 16, 17, 18, 19]]

: np.apply_along_axis(np.mean, axis=0, arr=a)
: array([ 7.5,  8.5,  9.5, 10.5, 11.5])

: np.apply_along_axis(np.mean, axis=1, arr=a)
: array([ 2.,  7., 12., 17.])
```

Example 2: Using user defined foo

```
: def foo(x):
:     return x/2
:
:
: np.apply_along_axis(foo, axis=0, arr=a)
:
[0. , 0.5, 1. , 1.5, 2. ],
[2.5, 3. , 3.5, 4. , 4.5],
[5. , 5.5, 6. , 6.5, 7. ],
[7.5, 8. , 8.5, 9. , 9.5]]
```

NumPy: Continued

- *Conditions and Boolean Arrays*

Example 1:

```
In [94]: a
Out[94]:
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19]])

In [95]: a<5
Out[95]:
array([[ True,  True,  True,  True,  True],
       [False, False, False, False, False],
       [False, False, False, False, False],
       [False, False, False, False, False]])
```

Example 2:

```
|: a[a<10]
|: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

Numpy: Structured Arrays

- Convenient way to keep data closely related together, e.g magnitude of an earthquake, together with coordinates, date, time, a
- Structured arrays contain *structs* of records. *dtype* data structure is used to declare a list of comma-separated elements that define the *struct* data type and order.

```
In [98]: x = np.array([('Rex', 9, 81.0), ('Fido', 3, 27.0)],  
    ...: ...          dtype=[('name', 'U10'), ('age', 'i4'), ('weight', 'f4')])
```

```
In [99]: x
```

```
Out[99]:  
array([('Rex', 9, 81.), ('Fido', 3, 27.)],  
      dtype=[('name', '<U10'), ('age', '<i4'), ('weight', '<f4')])
```

```
In [100]: x[1]
```

```
Out[100]: ('Fido', 3, 27.)
```

Numpy: Loading and Saving Data in Binary Files

```
In [104]: m= np.arange(10,50).reshape((8, 5))
```

```
In [105]: m
```

```
Out[105]:
```

```
array([[10, 11, 12, 13, 14],  
       [15, 16, 17, 18, 19],  
       [20, 21, 22, 23, 24],  
       [25, 26, 27, 28, 29],  
       [30, 31, 32, 33, 34],  
       [35, 36, 37, 38, 39],  
       [40, 41, 42, 43, 44],  
       [45, 46, 47, 48, 49]])
```

```
In [106]: np.save('saved_data',m)
```

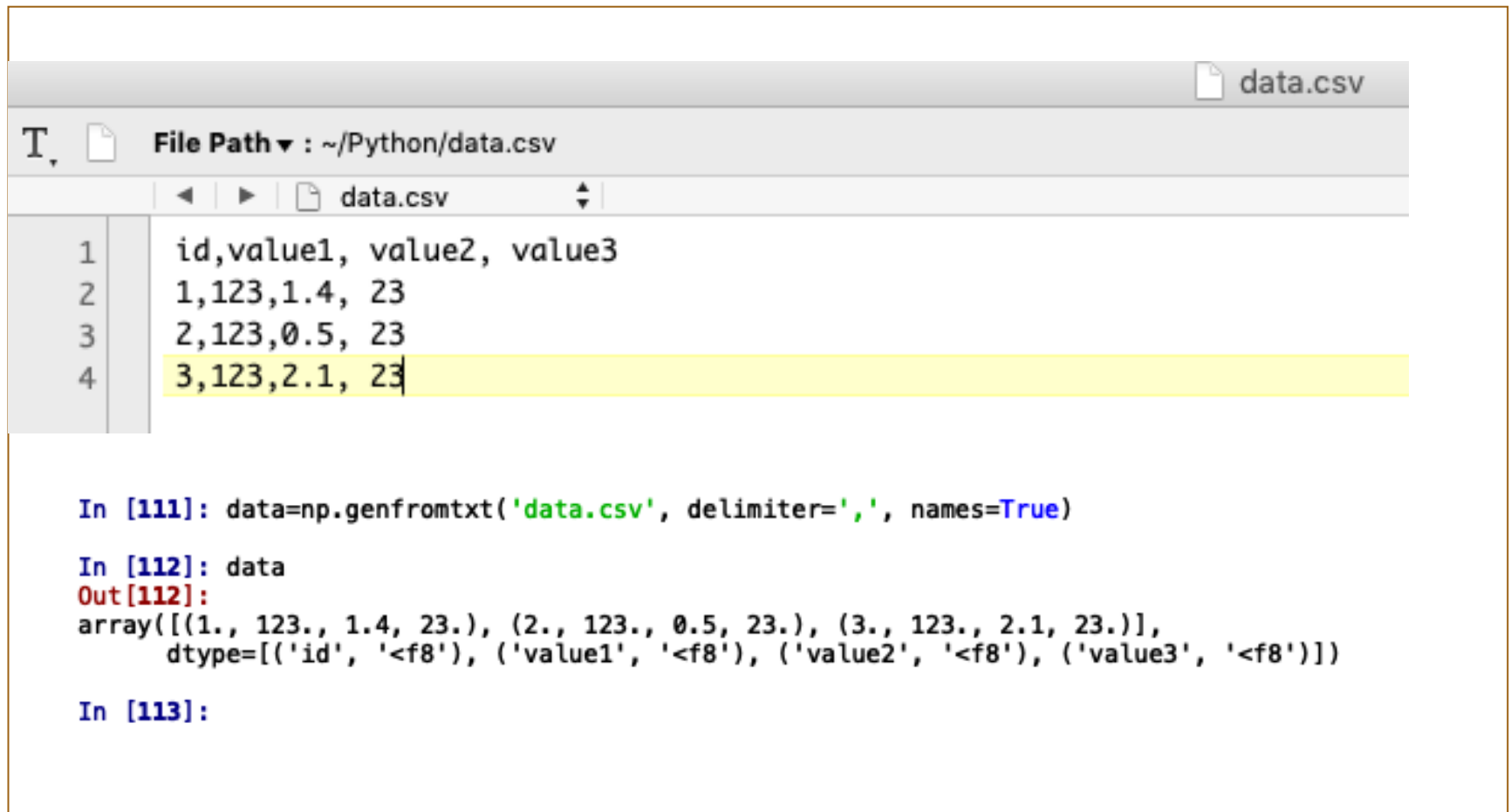
```
In [107]: loaded_data=np.load('saved_data.npy')
```

```
In [108]: loaded_data
```

```
Out[108]:
```

```
array([[10, 11, 12, 13, 14],  
       [15, 16, 17, 18, 19],  
       [20, 21, 22, 23, 24],  
       [25, 26, 27, 28, 29],  
       [30, 31, 32, 33, 34],  
       [35, 36, 37, 38, 39],  
       [40, 41, 42, 43, 44],  
       [45, 46, 47, 48, 49]])
```


Numpy: Reading Files with Tabular Data



The screenshot shows a Jupyter Notebook interface. At the top, a file browser shows a file named `data.csv` located at `~/Python/data.csv`. The file is open, showing its contents:

	id,value1, value2, value3
1	1,123,1.4, 23
2	2,123,0.5, 23
3	3,123,2.1, 23

The third row is highlighted in yellow. Below the file viewer, the Jupyter Notebook code cells are shown:

```
In [111]: data=np.genfromtxt('data.csv', delimiter=',', names=True)

In [112]: data
Out[112]:
array([(1., 123., 1.4, 23.), (2., 123., 0.5, 23.), (3., 123., 2.1, 23.)],
      dtype=[('id', '<f8'), ('value1', '<f8'), ('value2', '<f8'), ('value3', '<f8')])

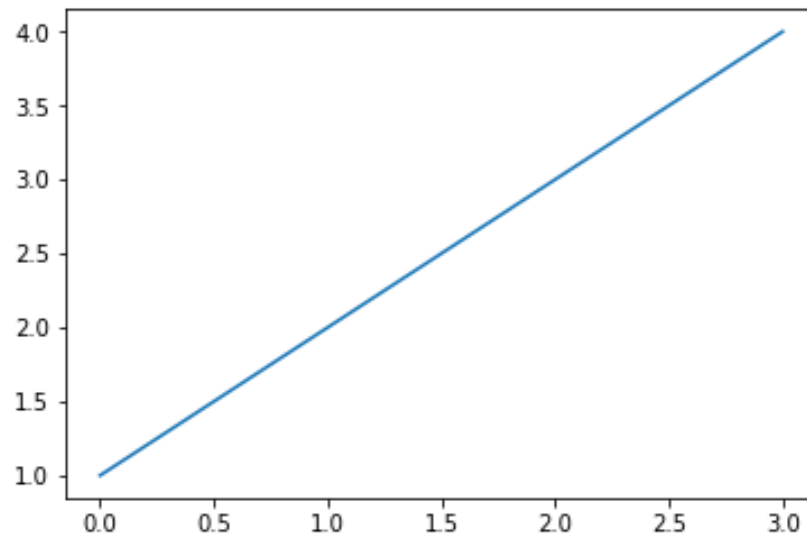
In [113]:
```

Data Visualization:pyplot

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

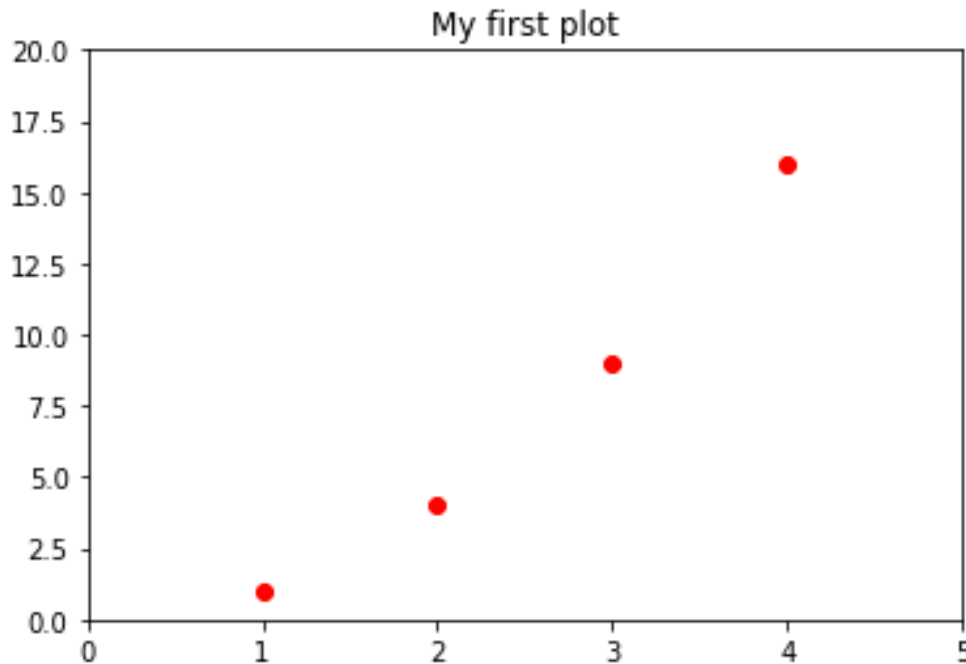
- Pass the values into plot function: e.g.

```
In [117]: plt.plot([1,2,3,4])  
Out[117]: [<matplotlib.lines.Line2D at 0x1182ab128>]
```



Data Visualization:plyplot

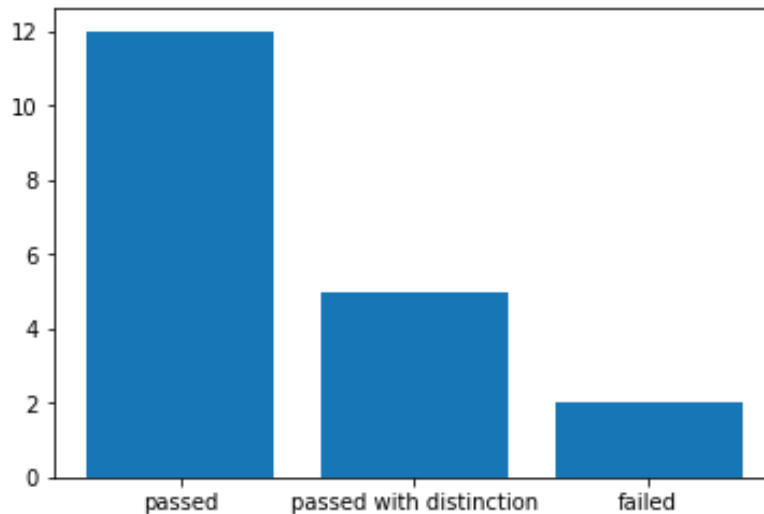
```
In [120]: plt.axis([0,5,0,20])  
...: plt.title('My first plot')  
...: plt.plot([1,2,3,4],[1,4,9,16],'ro')  
• Out[120]: [<matplotlib.lines.Line2D at 0x1192cf860>]
```



Data Visualization:plyplot

- Generate a bar graph from a dictionary

```
results = {}  
results['passed']=12  
results['passed with distinction']= 5  
results['failed']=2  
plt.bar(range(len(results)), list(results.values()), align='center')  
plt.xticks(range(len(results)), list(results.keys()))
```



TO DO: Week 1

- Read chapter 2: “Introduction to NumPy” in Python Data Science Hand Book:
<https://jakevdp.github.io/PythonDataScienceHandbook/index.html>
- Read sections “Simple Line Plots” and “Simple Scatter Plots” of chapter 4 “Visualization with Matplotlib”
- Start working on the lab 1. Lab 1 due on Friday, 7.6