# Scientific Programming in Python crash course

Big Data

Kristiania University College

## Objectives

1. Introduce scientific programming problems
2. Efficient matrix computations in Python
3. Visualize data in Python

## Bibliography

Jake VanderPlas. *Python Data Science Handbook*. Chapter 1. O'Reilly. (Link).

# Table of Contents

1. The data scientist toolkit
    - Motivation
    - Overview
    - Anaconda
    - Conda
2. NumPy
    - Understanding data types in Python
    - Introduction
    - NumPy array attributes
    - NumPy data types
    - NumPy notebook
3. Pandas
    - Introduction
    - The Pandas `Series` object
    - The Pandas `DataFrame` object
    - Pandas notebook
4. Data visualization
    - Introduction
    - Matplotlib
    - Matplotlib notebook
    - Searborn
    - Seaborn notebook

# The data scientist toolkit

## Motivation

Data science is about manipulating data

- Need of specialized tools
- Two main languajes: R and Python

Python is a general purpose programming language

- Easy integration
- Huge ecosystem of packages and tools

Need of data-oriented tools

- Features provided by third-party tools

# The data scientist toolkit

## Overview

| Tool | Type | Description |
|---|---|---|
| `conda` | Software | Python environments and package management |
| `iPython` | Software | Advaced Python interpreter |
| `Jupiter` | Software | Python notebooks (Python interpreter) |
| `Numpy` | Package | Efficient array operations |
| `Pandas` | Package | Dataframe support |
| `Matplotlib` | Package | Data visualization |
| `Seaborn` | Package | Data visualization with dataframes |
| `Scikit-learn` | Package | AI/ML package for Python |

# The data scientist toolkit

## Anaconda

Most of those tools are packaged in *Anaconda*

- Python distribution for Data Science
- Environment management for Python
- Package management system

Anaconda provides `conda`

- Packages management tool
- Environment management for Python

In addition, Anaconda provides `Spyder`

- Python IDE designed for Data Science

# The data scientist toolkit

## Conda crush introduction

### Conda environment for Data Science

1. `conda create --name ml seaborn=0.9.0`
2. `source activate ml`
3. `conda install ipython`
4. `conda install nb_conda`
5. `conda install scikit-learn`

List environments:

```
conda info --envs
```

Activate environment:

```
source activate <env>
```

Install package:

```
conda install <package>
```

List packages:

```
conda list
```

Exit environment:

```
conda deactivate
```

# NumPy

## Understanding data types in Python (I)

### Static typing

```c
/* C code */
int result = 0;
for(int i=0; i<100; i++){
    result += i;
}
```

- Data types must be declared
- Data types cannot change
- Error detection in compilation
- Variables names are, basicly, labels

### Dynamic typing

```python
# Python code
result = 0
for i in range(100):
   result += i
```

- Data types are not declared
- Data types can change
- Error detection in run-time
- Variables are complex data structures (even for simple types)

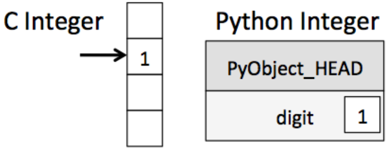
# NumPy

## Understanding data types in Python (II)

Dynamic typing must be implemented somewhere ...

**Python 3.4 source code**

```
struct _longobject {
    long ob_refcnt;
    PyTypeObject *ob_type;
    size_t ob_size;
    long ob_digit[1];
};
```

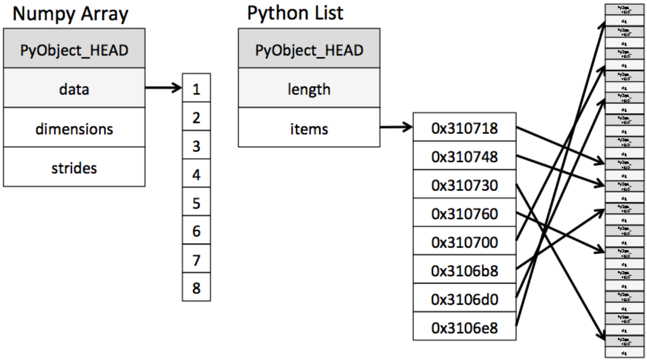


# NumPy

## Understanding data types in Python (III)

A Python list may contain different types

```
In [1]: L3 = [True, "2", 3.0, 4]
   ...: [type(item) for item in L3]
Out[1]: [bool, str, float, int]
```



# NumPy

## Understanding data types in Python (IV)

Standard Python data types are powerful and flexible

- Flexibility has a price: Reduced performance
- Not an big issue in generic programming
- A big issue in scientific programming
- We require efficient data manipulation mechanisms: NumPy

NumPy: Python package for numeric computation

- Efficient array implementation
- Fast mathematical functions
- Random numbers generation
- Static data types: Less flexibility

Most Python modules for AI/ML depend on NumPy, in particular

- Pandas (dataframes), Scikit-learn (ML), Seaborn (data visualization)

# NumPy

## Introduction

NumPy must be imported in order to be available

- You can use `np?` or `np.<TAB>`

The main component of NumPy is ndarray

- Python object
- Efficient matrix representation
- Homogeneus elements

Convention

```
import numpy as np
```

```
In [1]: array = np.array
    ([1,2,3])
In [2]: array
Out[1]: array([1, 2, 3])
In [3]: array = np.array
    ([[1,2],[3,4]])
```

# NumPy

## NumPy array attributes

Ndarray objects expose several attributes

- `ndim`: Dimensions
- `shape`: Size of each dimension
- `size`: Number of elements
- `dtype`: Data type
- `itemsize`: Size of each element (in bytes)
- `nbytes`: Size of the array (in bytes)

```
x = np.random.randint(10, size
    =(3, 4))
print("x ndim: ", x.ndim)
print("x shape:", x.shape)
print("x size: ", x.size)
print("dtype:", x.dtype)
print("itemsize:", x.itemsize)
print("nbytes:", x.nbytes)
```

# NumPy

## NumPy data types

Python is implemented in C

- Data types in NumPy are based on those in C

Two styles to declare types

- String: `np.zeros(10, dtype='int16')`
- NumPy object: `np.zeros(10, dtype=np.int16)`

| Data type | Description |
|---|---|
| `bool_` | Boolean (True or False) stored as a byte |
| `int_` | Default integer type |
| `intc` | Identical to C |
| `intp` | Integer used for indexing |
| `int8` | Byte |
| `int16` | Integer |
| `int32` | Integer |
| `int64` | Integer |
| `uint8` | Unsigned integer |
| `uint16` | Unsigned integer |
| `uint32` | Unsigned integer |
| `uint64` | Unsigned integer |
| `float_` | Shorthand for float64 |
| `float16` | Half precision float |
| `float32` | Single precision float |
| `float64` | Double precision float |
| `complex_` | Shorthand for complex128 |
| `complex64` | Complex number |
| `complex128` | Complex number |

# NumPy

## NumPy notebook

### NumPy notebook

(Link to notebook)

# Pandas

## Introduction

A DS/ML workflow needs more features

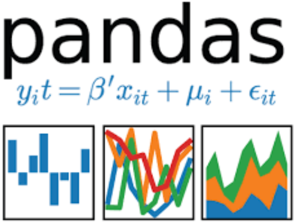
- Missing data
- Data input
- Operations on groups
- Label columns and rows

Pandas provides all those features, and more

- Pandas = **PAN**el **DA**ta **S**ystem
- Built on NumPy's ndarray
- Provides dataframes

Pandas provides two main objects

- `Series` and `DataFrame`



**Convention**

```
import numpy as np
import pandas as pd
```

# Pandas

## The Pandas `Series` object (I)

A `Series` is a one-dimensional array of indexed data

- NumPy arrays indices are implicit (i.e. its position)
- Series indices are explicit, and can be any type

Two attributes

- `values`: ndarray
- `index`: `pd.Index` object

Two indices

- Implicit: Regular index
- Explicit: Custom index

| Index | Values |
|---|---|
| ’a’ | 0.25 |
| ’b’ | 0.5 |
| ’c’ | 0.75 |
| ’d’ | 0.99 |

```
data = pd.Series([0.25,
     0.5, 0.75, 1.0])
data.values
data.index
data[1:3]
```

# Pandas

## The Pandas Series object (II)

### Custom indices

```
In[1] : data = pd.Series([0.25, 0.5, 0.75, 1.0],
    index=['a', 'b', 'c', 'd'])
In [2]: data
Out[1]:
    a     0.25
    b     0.50
    c     0.75
    d     1.00
    dtype: float64

    In [3]: data['a']
    Out[2]: 0.25
    In [4]: data[0]
    Out[3]: 0.25
```

# Pandas

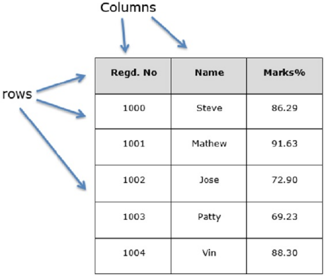
## The Pandas `DataFrame` object (I)

A `DataFrame` is a 2-D tabular data structure

- Similar to a spreadsheet
- Homogeneous columns
- Heterogeneous rows

Two read-only attributes, both `pd.Index`

- `index`: Rows
- `columns`: Columns



| Regd. No | Name | Marks% |
|---|---|---|
| 1000 | Steve | 86.29 |
| 1001 | Mathew | 91.63 |
| 1002 | Jose | 72.90 |
| 1003 | Patty | 69.23 |
| 1004 | Vin | 88.30 |

(Source)

# Pandas

## The Pandas DataFrame object (II)

DataFrame example

```
In [1]:  import seaborn as sns

In [2]:  iris = sns.load_dataset('iris')
In [3]:  iris.head()

Out[1]:
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
In [246]: iris.columns
Out[246]:
Index(['sepal_length', 'sepal_width', 'petal_length',
       'petal_width', 'species'], dtype='object')
```

# Pandas

## The Pandas `DataFrame` object (III)

Read from a file

- Excel:
  `pd.read_excel('filename.xlsx', sheetname='mysheet')`
- CSV (very common!!!): `pd.read_csv('filename.csv')`

### CSV example

```
# This CSV file contains data about weights and heights
"id", "weight","height","sex",  "race"
1,     143.5,    81.6,     "Female","White"
2,     109.1,    83.7,     "Female","Black"
4,     104.8,    54.6,     "Female","Hisp"
7,     130.2,    81.7,     "Male",  "White"
```

CVS can be exported from MS Excel or programatically

# Pandas

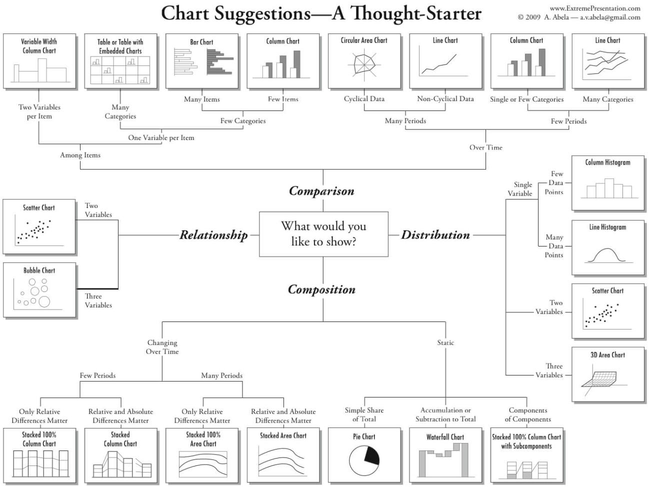
## Pandas notebook

**NumPy notebook**

(Link to notebook)

# Data visualization

## Motivation



(Source)
(Alternative resource)

# Data visualization

## Matplotlib (I)

Matplotlib is a Python package

- Based on NumPy
- Imitates Matlab

Three operation modes

- Scripts.
  Must use `plt.show()` to enter event loop. Use it once!
- IPython shell.
  Must use `%matplotlib`
- IPython notebook. Two modes
  - `%matplotlib inline`
  - `%matplotlib notebook`

Convention

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

myplot.py

```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0, 10, 100)

plt.plot(x, np.sin(x))
plt.plot(x, np.cos(x))

plt.show()
```

# Data visualization

## Matplotlib (II)

Matplotlib comes with two interfaces

- Matlab-like. Old-fashioned function-oriented API.
- Object-oriented. Object-oriented and more powerfull API.

### Matlab API

```
plt.figure()  # create a plot

# create the first of two panels
     and set current axis
plt.subplot(2, 1, 1) # (rows,
    columns, panel number)
plt.plot(x, np.sin(x))

# create the second panel and
     set current axis
plt.subplot(2, 1, 2)
plt.plot(x, np.cos(x));
```

### OO API

```
plt.figure()  # create a plot

# create the first of two panels
     and set current axis
plt.subplot(2, 1, 1) # (rows,
    columns, panel number)
plt.plot(x, np.sin(x))

# create the second panel and
     set current axis
plt.subplot(2, 1, 2)
plt.plot(x, np.cos(x));
```

# Pandas

## Matplotlib notebook

### Matplotlib notebook

(Link to notebook)

# Data visualization

## Seaborn (I)

Seaborn is a modern data-visualization Python package

- Based on matplotlib
- ... it uses matplotlib indeed
- Pandas-aware
- High level
- Advanced visualizations
- Easy to use

Still under development! (v. 0.9)

**Convention**

```
import seaborn as sns
```

This documentation is for Seaborn 0.9 or newer

# Data visualization

## Seaborn (II)

Display initialization

- `plt.show()`
- `%matplotlib`

Style initialization

- Default Seaborn style `sns.set()`
- By default, same style than matplotlib

Several functions ...

- ... similar parameters

### Parameters

- x: Data axis x
- y: Data axis Y
- data: Dataframe name
- hue: Color
- style: Style
- sizes: Size
- kind: Alternate representation

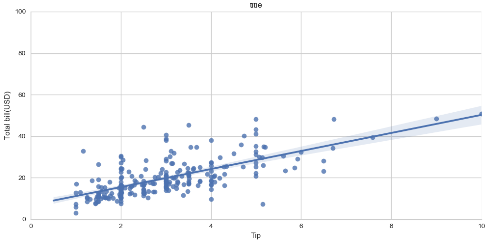
# Data visualization

## Seaborn (III)

### Typical Seaborn usage

1. Prepare data
2. Set up aesthetics
3. Plot
4. Customize the plot

```
import matplotlib.pyplot as plt
import seaborn as sns
# Prepare data
tips = sns.load_dataset(”tips”)
# Set up aesthetics
sns.set_style(”whitegrid”)
# Plot
g = sns.lmplot(x=”tip”,y=”total_bill”, data=tips,aspect=2)
# Plot customization
g = (g.set_axis_labels(”Tip”,”Total bill(USD)”).set(xlim
        =(0,10),ylim=(0,100)))
plt.title(”title”)
plt.show(g)
```



# Seaborn

## Seaborn notebook

### Seaborn notebook

(Link to notebook)