

# **Data Analysis in Python**

## **Course for SEA/UAB 2017-2018**

José María Lago Alonso

[https://github.com/jmlago/DA\\_python](https://github.com/jmlago/DA_python)

# Structure of the Course

## 1 Aerial perspective

Basic concepts of Data Analysis and Python. A global view of the environment...

## 2.4.5 Data Gathering

How to extract real and quality data from Internet...

## 3 Exploratory Analysis and Cleaning

Now with real data, how to prepare this data, understand the data, and do visualizations...

## 4 Algorithm Selection

A tiny view of algorithms world and how to use them...

## 5 Performance Engineering

Usually, when we work with big amounts of data, we need to speed up our algorithms ...

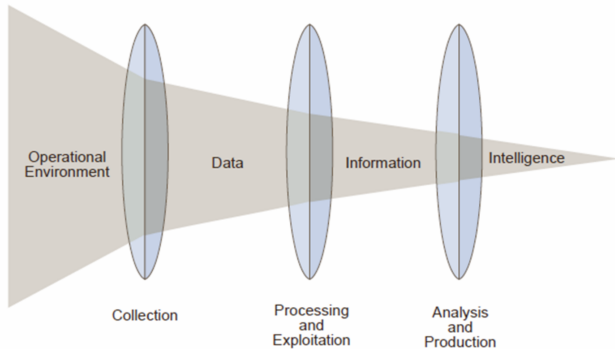
## 6 Deep Learning

A tiny view of the deep learning world...

# What is Data Analysis?

Analysis of data is a process of inspecting, cleansing, transforming, and modeling data with the goal of discovering useful information, suggesting conclusions, and supporting decision-making.

## Relationship of Data, Information and Intelligence



# What is Python?

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics.

- ▶ Interpreted:  
Compile and execute by any size blocks in any time.
- ▶ Object Oriented:  
Uses classes objects and attributes.
- ▶ High-Level:  
Can not allocate memory manually.
- ▶ Dynamic semantics:  
Python doesn't have static types.

# Why use Python?

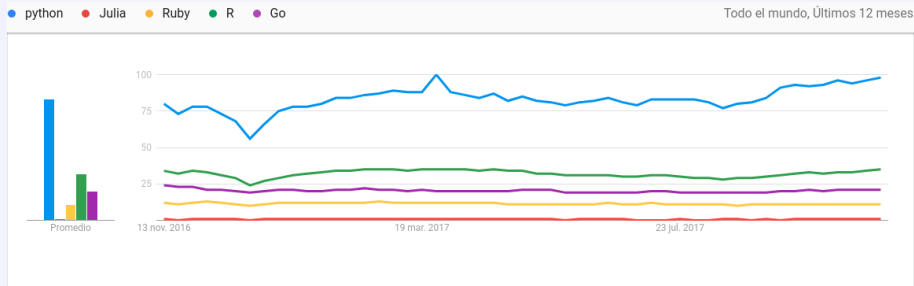
## Strengths

- ▶ Glue language
- ▶ Simple and easy to learn
- ▶ Program modularity and code reuse
- ▶ Edit-test-debug really fast
- ▶ General purpose language
- ▶ Cross platform

## Weaknesses

- ▶ Slower than compiled languages
- ▶ Python2 not compatible with Python3
- ▶ Lack of static types
- ▶ Can't free memory in usual way

# Python vs Others



**Figure:** Python vs Julia vs R vs Go (Google Trends)

- ▶ **Community**
- ▶ Multi-purpose
- ▶ Easy to learn
- ▶ Julia is faster? And Go? (Numpy/Numba/Torch/CuPy...)
- ▶ Frameworks and wrappers

# Conda

## Anaconda

**We are going to use Python3 in all this course !!!**

### Definition

Anaconda is an easy-to-install free package manager, environment manager, Python distribution, and collection of over 720 open source packages offering free community support.

### Why?

Because Anaconda's packages are for data analytics, data science, and scientific computing.

Also Conda makes sure that all packages and environments works fine together.

# Installation

- ▶ Go to: <https://www.continuum.io/downloads>
- ▶ Choose your OS
- ▶ Download the installer
- ▶ Follow the steps on the webpage

## Warning

For Linux users, you may use:

```
bash Anaconda3-5.0.1-Linux-x86_64.sh
```

To install Anaconda.



## Anaconda tips

For Windows users, you should open AnacondaPrompt terminal. For Mac and Linux users, you can open the regular terminal.

- ▶ List available python:

```
conda search "^python$"  
source ~/.bashrc #(Linux users)
```

- ▶ Create and activate the environment:

```
conda create --name my_env python=3  
source activate my_env
```

- ▶ Install packages:

```
conda install some_package  
pip install some_package
```

- ▶ Remove environment:

```
conda remove --name my_env --all
```

# Alternative Installations

## Linux

### Linux

```
sudo apt-get install python3-pip python3-dev
↪ python-virtualenv
pip3 install --upgrade pip3
cd /usr/local/share
sudo mkdir virtualenvs
sudo chown -R root:sudo virtualenvs/
sudo chmod -R g+w virtualenvs/
virtualenv --system-site-packages -p python3
↪ /usr/local/share/virtualenvs/v1
source /usr/local/share/virtualenvs/v1/bin/activate
pip3 install -r requirements.txt
```

# Alternative Installations

## Mac and Windows

### Mac

Same script that in Linux, but changing the `sudo apt-get install` for `brew install`

### Windows

- ▶ Download Python3 from <https://www.python.org/downloads/windows/>
- ▶ open the cmd and type `python get-pip.py`
- ▶ After that, execute:
  - ▶ `pip install virtualenv`
  - ▶ `pip install virtualenvwrapper-powershell`
  - ▶ `mkdir '~\.virtualenvs'`
  - ▶ `Import-Module virtualenvwrapper`
- ▶ To see the lists of commands that we can use, just type:  
`Get-Command *virtualenv*`

# Spyder

## Integrated Development Enviroment

### Definition

Scientific PYthon Development EnviRonment

- ▶ Similar to RStudio and MATLAB IDE's
- ▶ IDE for Science
- ▶ Exploratory
- ▶ Easy debugging

spyder

## Other IDEs



**Figure:** Atom + Hydrogen



**Figure:** Jupyter



**Figure:** Eclipse + PyDev

All IDEs have advantages and disadvantages, you will need to choose what suits you most. In this course we will use SPYDER for simplicity, but each IDE has different purposes.

# Verify that all is working OK

- ▶ Open SPYDER
- ▶ Try to execute the next code:

```
import pandas as pd
import os
import mxnet as mx
import scipy as sp
```

```
print("Everything is working OK!!!")
```

Type the code, select all the code and press CTRL+INTRO

# Python Basics 1

## Libraries

In Python we need to set the modules that we are going to use at the beginning of the script. We do in the following way:

```
import somelibrary as somename  
somename.somefunctioninthelib()
```

## Generic Data Types

Python really has dynamic semantics so a variable is somehow dynamic type, for instance:

```
a = [1,2,3] #--> list of numbers  
a = a[1] #--> position in a list  
a = "abcd" #--> string  
...
```

# Python Basics 2

## Functions

We can use functions very easy because of the dynamic semantics. Also in Python the most important thing is **INDENTATION** this is how we determine the loops and the range of the functions.

Beautiful and readable code.

```
def somefunction(param1,param2):  
    a = param1*param2  
    return a
```



# Python Basics 3

## Classes

Here is the OO part.

Easy example:

```
class Complex:
    def __init__(self, realpart, imagpart):
        self.r = realpart
        self.i = imagpart
```

```
x = Complex(3.0, -4.5)
```

```
x.r, x.i
(3.0, -4.5)
```

**Let's practice a little bit !!!**

Open the practice0.py

After this open oo.py

# Python Basics 4

## Reserved words

```
if __name__ == "__main__":  
    print("Hello main")  
#####  
class C2(C1):  
    def __len__(self,...):
```

## Relative paths and modules

What is a python module? What is `__init__.py`?

```
from ..outer.inner import foo
```

hello/

\_\_init\_\_.py

params.py

bye/

\_\_init\_\_.py

params2.py

**Exercise:** Try to import in `params2.py` some variable defined in `params.py` without executing the main script.

# Help!

- ▶ StackOverFlow and How To "do something" in Python
- ▶ Python documentation
- ▶ External libraries documentation

## Warning

Be careful of StackOverFlow because it's easy to copy and paste the code but if you don't understand what are you doing you'll have several problems.



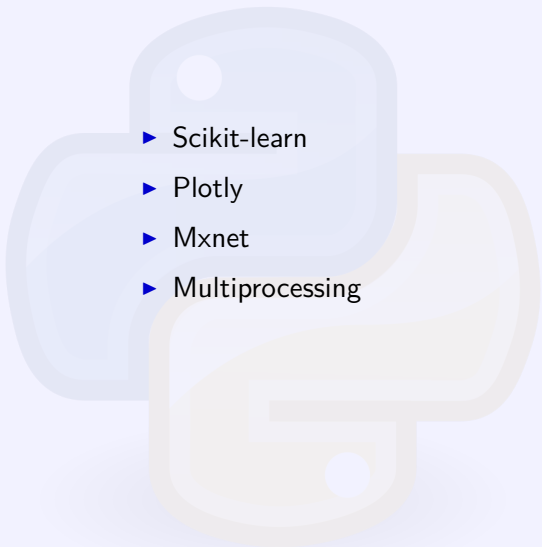
stack overflow

# Libraries that we need

- ▶ Pandas
- ▶ BeautifulSoup
- ▶ Matplotlib
- ▶ NumPy
- ▶ SciPy

And many others...

- ▶ Scikit-learn
- ▶ Plotly
- ▶ Mxnet
- ▶ Multiprocessing



# Numpy 1

## What is numpy ?

NumPy is the fundamental package for scientific computing with Python. It contains among other things:

- ▶ A powerful N-dimensional array object
- ▶ Sophisticated (broadcasting) functions
- ▶ Tools for integrating C/C++ and Fortran code
- ▶ Useful linear algebra, Fourier transform, and random number capabilities

## Numpy 2

Now we have numeric arrays (not lists) and we can apply operations:

### Numpy arrays

```
import numpy as np
a = np.array(1,2,3,4)      # WRONG
a = np.array([1,2,3,4])    # RIGHT
b = np.array([(1.5,2,3), (4,5,6)])
print(b)
      array([[ 1.5,  2. ,  3. ],
             [ 4. ,  5. ,  6. ]])
c = np.array( [ [1,2], [3,4] ], dtype=complex )
print(c)
      array([[ 1.+0.j,  2.+0.j],
             [ 3.+0.j,  4.+0.j]])
```

# Numpy 3

As we can see in the previous slide there are many dtypes for an array of NumPy. More precisely a dtype follow the next properties:

## Properties

- ▶ Type of the data (integer, float, Python object, etc.)
- ▶ Size of the data (how many bytes is in e.g. the integer)
- ▶ Byte order of the data (little-endian or big-endian)
- ▶ If the data type is structured (an aggregate of other data types).

## Dtypes sample

int16,bool,float64,complex,str,unicode,buffer,object,...

# Numpy 4

Some useful algebraic functions, better to see as example:

## Numpy useful functions

```
a = np.array([[1.0, 2.0], [3.0, 4.0]])  
a.transpose() #transposed matrix  
np.linalg.inv(a) #inverse matrix  
np.dot(a, a) #matrix product  
np.trace(a) #trace of the matrix  
y = np.array([[5.], [7.]]) #independent term  
np.linalg.solve(a, y) #solver of a system
```



## Numpy 5

Polynomial fit functions that minimize the SSE:

### Polynomial fit functions

```
x = np.array([0.0, 1.0, 2.0, 3.0, 4.0, 5.0])
y = np.array([0.0, 0.8, 0.9, 0.1, -0.8, -1.0])
z = np.polyfit(x, y, 3)#3 is the degree
#of the fitting polynomial
print(z)
      array([ 0.08703704, -0.81349206,
              1.69312169, -0.03968254])
p = np.poly1d(z)
p(0.5)
      0.6143849206349179
p30 = np.poly1d(np.polyfit(x, y, 30))
      RankWarning: Polyfit may be poorly conditioned
```

**Practice again!!!**

Open the `numpy_practice.py`

# Matplotlib 1

## What is matplotlib ?

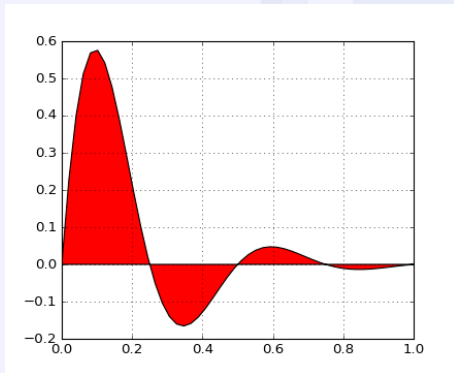
Matplotlib is a python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms.

## Basic example

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0, 1)
y = np.sin(4 * np.pi * x) * np.exp(-5 * x)
plt.fill(x, y, 'r') #fills the curve with red color
plt.grid(True) #draws the grid
plt.show() #shows the plot
```

## Matplotlib 2

Show of the previous plot:



## Matplotlib 3

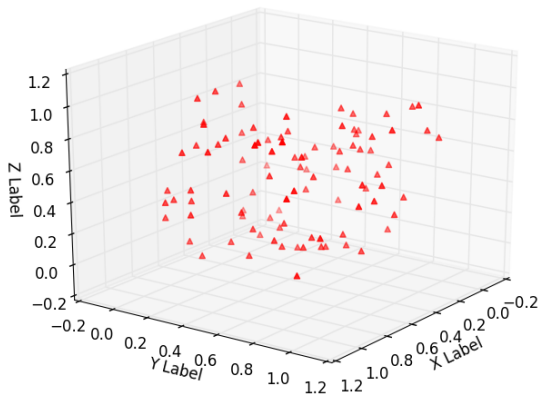
Now we are going to draw some data in 3D just as an example:

### 3D example

```
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
n = 100
xs = np.random.uniform(0,1,n)
ys = np.random.uniform(0,1,n)
zs = np.random.uniform(0,1,n)
ax.scatter(xs, ys, zs,color="r",marker="^")
ax.set_xlabel('X Label')
ax.set_ylabel('Y Label')
ax.set_zlabel('Z Label')
plt.show()
```

# Matplotlib 4

Show of the previous plot:



# Matplotlib 5

The most important part of matplotlib is...

The gallery: <http://matplotlib.org/gallery.html>

Images, contours, and fields



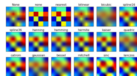
contourf\_log



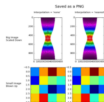
image\_demo



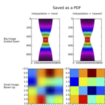
image\_demo\_clip\_path



interpolation\_methods



interpolation\_none\_vs\_nearest



interpolation\_none\_vs\_nearest



pcolormesh\_levels



streamplot\_demo\_features



streamplot\_demo\_features



streamplot\_demo\_masking



streamplot\_demo\_start\_points

Pie and polar charts



pie\_demo\_features



pie\_demo\_features



polar\_bar\_demo



polar\_scatter\_demo

**Practice again !!!**

Open practice1.py

# Pandas 1

This library is the CORE of Data Analysis in Python.  
And It's most powerful tool...

## DataFrame class

This is our table and our database structure in Python.  
Has many many attributes and most libraries were build using this structure as a link.

## Example

```
import pandas as pd
d = [['Pepito',22],['Juanito',43],['Pablito',20]]
c = ["Name","Age"]
df = pd.DataFrame(data=d,columns = c)
```

## Pandas 2

Knowing a little bit more about DataFrame class:

### Continuing Example

```
df.Age #--> Sometimes correct  
df["Age"] #--> Correct  
df["Age"].mean()  
df["Age"].var()
```

Let's take a look at the documentation:

<http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.html>



# Pandas 3

Getting information in different formats:

## Some format to DataFrame

```
##All this sentences return a DataFrame object  
pd.read_csv(file)  
pd.read_excel(file)  
pd.read_html(path) # directly from the web  
pd.read_json(path) # directly from the web  
pd.read_sas(file)  
pd.read_sql(file)  
pd.read_sql_table(file)  
pd.read_stata(file)  
df.from_csv(file)  
df.from_dict(dict)  
...
```

To see how it works, just open `easy_plot.py`

# Pandas 4

Query inside DataFrame class:

## Query in Python

```
x = df[(df["label1"] == "some1") &  
(df["label2"] == "some2")]["label3"]
```

This could be a set of numbers coming from the DataFrame as a request. The request is like we want all elements on the DataFrame that have in the column named label1 the value some1 **and** that have in the column named label2 the value some2. And from this set of rows that follow the answer we want the values of column named label3. Then put all this values in a variable named x.

## DataFrame to L<sup>A</sup>T<sub>E</sub>X

```
df2.to_latex().replace("\n", "")  
# generates a table in LaTeX format
```

# Pandas 5

## Few more possibilities

```
df.dropna(axis=1, how='any', inplace=True) #drop NA values  
agg_col = df.groupby('label1').aggregate(sum)  
agg_col.index = ['bar1', 'bar2', 'bar3']  
agg_col.plot(kind='bar') #see other kinds in documentation
```

# Pandas 6

```
df = pd.concat(df1,df2,df3)
```

df1				
	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

df2				
	A	B	C	D
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7

df3				
	A	B	C	D
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11

Result					
		A	B	C	D
x	0	A0	B0	C0	D0
x	1	A1	B1	C1	D1
x	2	A2	B2	C2	D2
x	3	A3	B3	C3	D3
y	4	A4	B4	C4	D4
y	5	A5	B5	C5	D5
y	6	A6	B6	C6	D6
y	7	A7	B7	C7	D7
z	8	A8	B8	C8	D8
z	9	A9	B9	C9	D9
z	10	A10	B10	C10	D10
z	11	A11	B11	C11	D11

# Pandas 7

```
result = pd.concat([df1, df4], axis=1)
```

df1					df4				Result									
										A	B	C	D	B	D	F		
		A	B	C	D			B	D	F	0	A0	B0	C0	D0	NaN	NaN	NaN
0		A0	B0	C0	D0	2		B2	D2	F2	1	A1	B1	C1	D1	NaN	NaN	NaN
1		A1	B1	C1	D1	3		B3	D3	F3	2	A2	B2	C2	D2	B2	D2	F2
2		A2	B2	C2	D2	6		B6	D6	F6	3	A3	B3	C3	D3	B3	D3	F3
3		A3	B3	C3	D3	7		B7	D7	F7	6	NaN	NaN	NaN	NaN	B6	D6	F6
											7	NaN	NaN	NaN	NaN	B7	D7	F7

# Pandas 8

```
result = pd.concat([df1, df4], axis=1, join='inner')
```

df1					df4				Result							
	A	B	C	D		B	D	F		A	B	C	D	B	D	F
0	A0	B0	C0	D0	2	B2	D2	F2	2	A2	B2	C2	D2	B2	D2	F2
1	A1	B1	C1	D1	3	B3	D3	F3	3	A3	B3	C3	D3	B3	D3	F3
2	A2	B2	C2	D2	6	B6	D6	F6								
3	A3	B3	C3	D3	7	B7	D7	F7								

# Pandas 9

```
result = df1.append(df4)
```

df1					Result						
	A	B	C	D		A	B	C	D	F	
0	A0	B0	C0	D0	0	A0	B0	C0	D0	NaN	
1	A1	B1	C1	D1		1	A1	B1	C1	D1	NaN
2	A2	B2	C2	D2		2	A2	B2	C2	D2	NaN
3	A3	B3	C3	D3		3	A3	B3	C3	D3	NaN
df4						2	NaN	B2	NaN	D2	F2
	B	D	F	3		NaN	B3	NaN	D3	F3	
2	B2	D2	F2	6		NaN	B6	NaN	D6	F6	
3	B3	D3	F3	7	NaN	B7	NaN	D7	F7		
6	B6	D6	F6								
7	B7	D7	F7								

# Pandas 10

```
result = pd.merge(left, right,  
how='left', on=['key1', 'key2'])
```

left					right					Result						
	A	B	key1	key2		C	D	key1	key2		A	B	key1	key2	C	D
0	A0	B0	K0	K0	0	C0	D0	K0	K0	0	A0	B0	K0	K0	C0	D0
1	A1	B1	K0	K1	1	C1	D1	K1	K0	1	A1	B1	K0	K1	NaN	NaN
2	A2	B2	K1	K0	2	C2	D2	K1	K0	2	A2	B2	K1	K0	C1	D1
3	A3	B3	K2	K1	3	C3	D3	K2	K0	3	A2	B2	K1	K0	C2	D2
										4	A3	B3	K2	K1	NaN	NaN



# Pandas 11

```
result = pd.merge(left, right,  
how='outer', on=['key1', 'key2'])
```

left					right					Result						
	A	B	key1	key2		C	D	key1	key2		A	B	key1	key2	C	D
0	A0	B0	K0	K0	0	C0	D0	K0	K0	0	A0	B0	K0	K0	C0	D0
1	A1	B1	K0	K1	1	C1	D1	K1	K0	1	A2	B2	K1	K0	C1	D1
2	A2	B2	K1	K0	2	C2	D2	K1	K0	2	A2	B2	K1	K0	C2	D2
3	A3	B3	K2	K1	3	C3	D3	K2	K0							

# Pandas 12

```
result = pd.merge(left, right,  
how='inner', on=['key1', 'key2'])
```

left					right					Result						
											A	B	key1	key2	C	D
	A	B	key1	key2		C	D	key1	key2	0	A0	B0	K0	K0	C0	D0
0	A0	B0	K0	K0	0	C0	D0	K0	K0	1	A1	B1	K0	K1	NaN	NaN
1	A1	B1	K0	K1	1	C1	D1	K1	K0	2	A2	B2	K1	K0	C1	D1
2	A2	B2	K1	K0	2	C2	D2	K1	K0	3	A2	B2	K1	K0	C2	D2
3	A3	B3	K2	K1	3	C3	D3	K2	K0	4	A3	B3	K2	K1	NaN	NaN
										5	NaN	NaN	K2	K0	C3	D3

... and many other ways to fusion DataFrames.

**Practice again!!!**

Open the practice2.py

# Titanic Dataset

In this first real example, we are going to study and analyze the data corresponding to the titanic. Open `titanic_preprocessing.py`!!!