Intro to Python for Machine Learning

Dr Vincent Croft - 1st Terascale School of Machine Learning

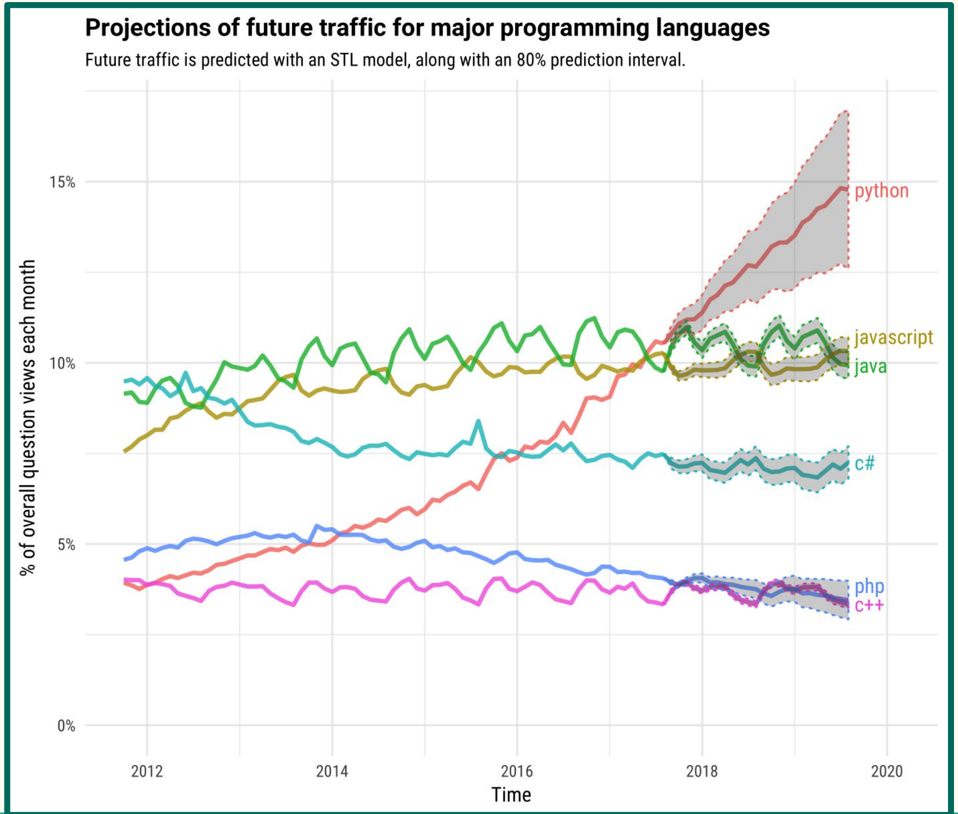
22nd October 2018 Desy -Hamburg

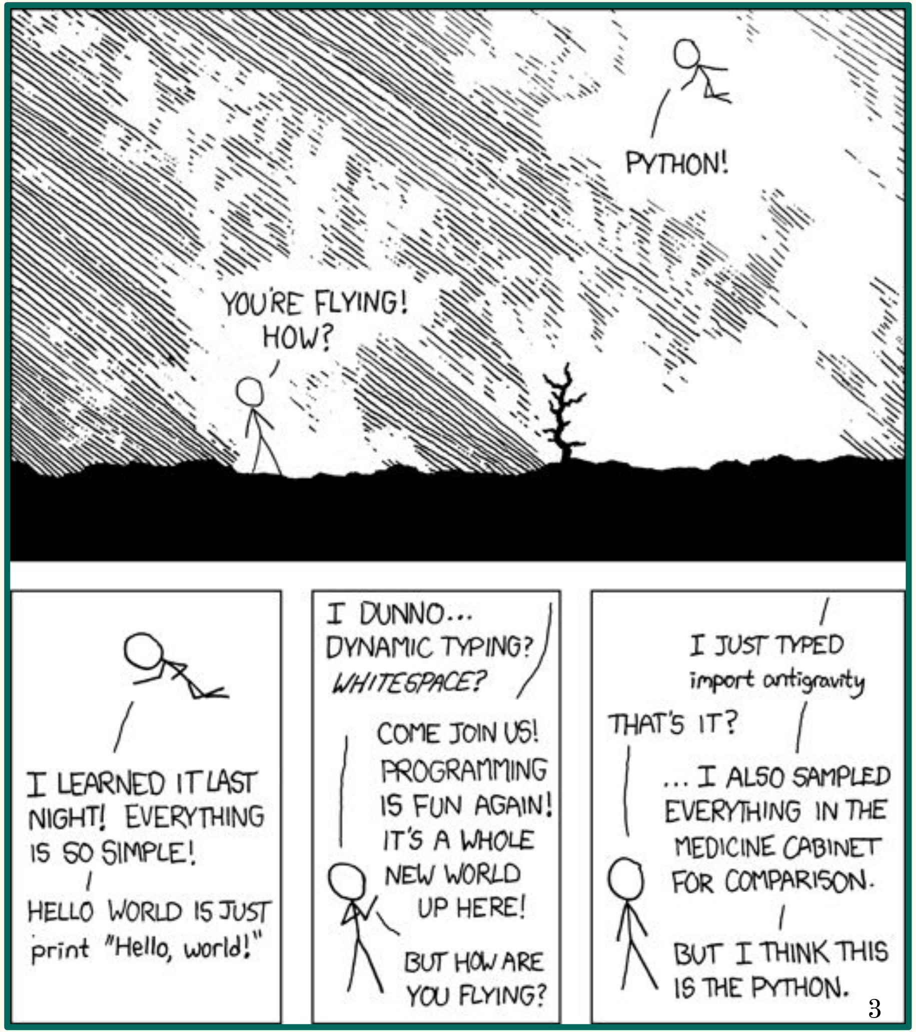
Part One - Welcome to python

- Why python?
- Getting and using python
  - Locally
  - Docker
  - Cloud
- Mastering the basics
  - Types
  - Logic
  - Containers
  - Loops
- Structures
  - Functions
  - Classes

Why python?

Python is easy to understand.





Step one: print(“hello world”)

https://www.python.org/downloads/

https://www.anaconda.com/download/

brew/apt-get/microsoft-store install python

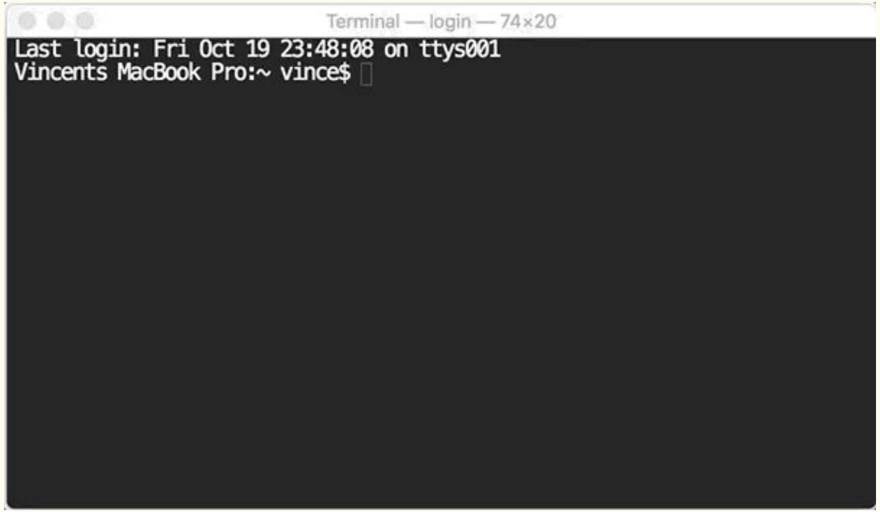
While you’re at it make sure you have a github account and git installed locally

And Docker https://docs.docker.com/install/

Step one: print(“hello world”)

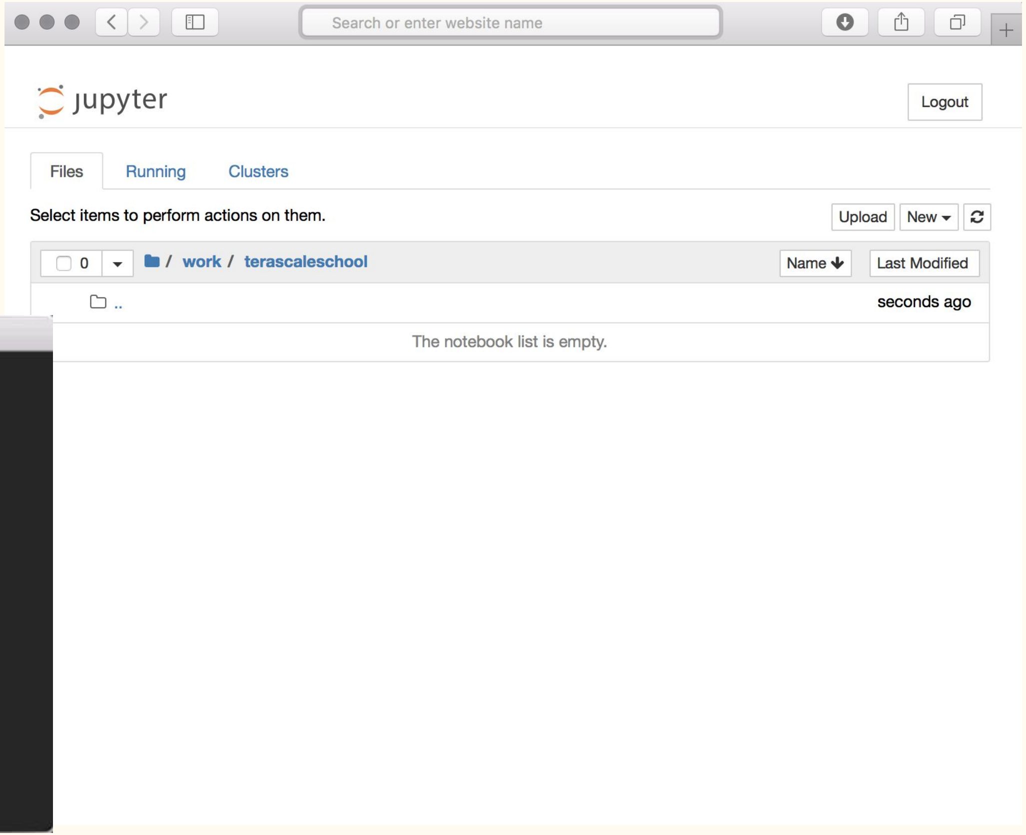
Get python running locally!

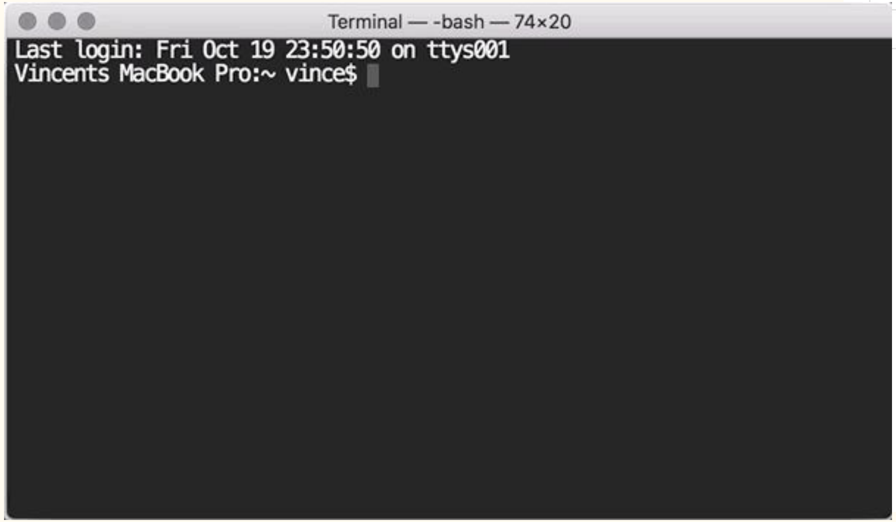
print “hello world”



Step Two: Run Jupyter

pip install jupyter





print “hello world” in notebook

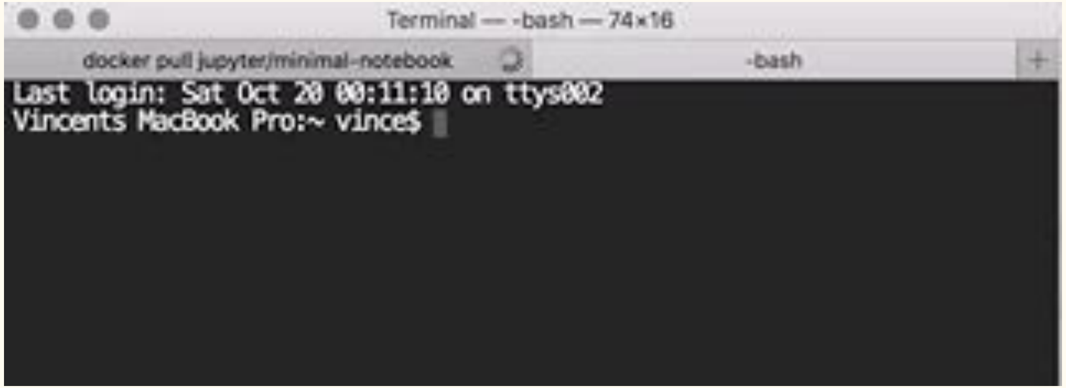
Easy? Ok now do it in docker - OPTIONAL

Simply run python - `docker run -it python:3.6`

Then a notebook - `docker run -p 8888:8888 jupyter/minimal-notebook`

Then mount a local dir in the container - `mkdir ~/notebooks`

```
docker run --rm -p 8888:8888 -v "$PWD":/home/jovyan/work jupyter/datascience-notebook
```



Step Three: Simple git-Did you log into github.com? No? do it now!

Fork this course material at https://github.com/vincecr0ft/terascale_python

Clone your fork of the code

git checkout -b a_new_branch_name

Edit you name in README.md

git add README.md

git commit -m “adding my name”

git push origin a_new_branch_name

In browser merge your new branch with master (your master)

Chapter Two: Actual python

Python is simple and flexible

Variable assignment is automatic

Try some maths with simple integers, floats, boolians and strings!

A = 3.142

B = “bananas”

print(A+B) etc

Logic

```
N_counts = “count of three”

if “four” in N_counts:

    print “ney! Four is too many”

elif “three” in 1234:

    print “ok three’s fine

else:

    do_something()
```

Containers

```
words = ["first", "second","third"]

words.append("forth")

print words

print words[2]
```

```
numbers = {"first":1,
"second":2,"third":3}

numbers["forth"] = 4

print numbers

print numbers["second"]
```

Exercise 1: Prime Numbers

The goal is to make a dictionary of important prime numbers

In the range 1-100:

- What is the largest prime number?
- What is the most common factor (it should be two)
- What is the largest prime factor

Chapter 3: Structuring your code

Python is great for simple maths but is it more than a scripting language?

```
def some_function(some_interpreted_inputs):
    result = some_interpreted_inputs * 2        #Some calculation
    return result
```

Classes - Containers for your functions

```
class MyClass:
    def __init__(self, initial,variables):
        self.name = initial+variables
    def who_am_i(self):
        print(self.name)
```

Want to know more?

projecteuler.net

Project Euler is a series of challenging mathematical/computer programming problems that will require more than just mathematical insights to solve. Although mathematics will help you arrive at elegant and efficient methods, the use of a computer and programming skills will be required to solve most problems.

Part Two - Python for Machine Learning

- Intro to Numpy
- Numpy Algebra
- Data with Pandas
- Generating data with Numpy
- Plotting with Matplotlib
- Machine Learning with Scikit-learn

Numpy!

Numpy makes python fast!

Python is considered slow because it is an interpreted language

In each loop type comparisons and function overloading factors into the run time

Numpy type casts the array elements and pushes the overloading deep down into the compiled (fortran) core of the library.

This allows for a roughly 100x speed up on all iterative, low level operations.

Numpy for loops- Example One: ufuncs

Pure python lists

```
a = [1 , 2, 3, 4, 5, 6, 7 , 8, 9, 10]

b = [i + 5 for i in a]

print(b)

[6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
```

Numpy arrays

```
Import numpy as np

a = np.array(a)

b = a + 5

print(b)

[6 7 8 9 10 11 12 13 14 15]
```

Numpy for loops- Example One: ufuncs

Arithmetic operators: + - * / // % **

Bitwise operators: & | ~ ^ >> <<

Comparison operators: < > <= >= == !=

Trigonometric functions: np.sin, np.cos, np.tan, etc

Exponents: np.exp, np.log, np.log10, etc

...and more...

Numpy for loops- Example Two: agregations

Aggregations summarise the data in an array

min(a) -> a.min()

max(b) -> b.max()

sum(c) -> c.max()

sum(d)/len(d) -> d.mean()

Also works on multidimensional arrays.

Lots of aggregations available. All more than 50x faster than pure python

Numpy for loops- Example Three: broadcasting

Ufuncs for weird size arrays

Add a row to a matrix or add a column vector to a row vector - linear algebra!

No more loop indexes

Numpy for loops- Example Four: slicing and more

Pure python

```
a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

print a[3]

4

print a[2:6]

[3, 4, 5, 6]
```

But that's it!

```
a = np.array([1, 2, 3, 4])

mask = np.array([True, False, True, False])

print a[mask]

array([1,3])

b = np.array([5, 6, 7, 8, 9, 10, 11, 12, 13])

mask = (b%2 == 0 ) & (b < 10 )

print(b[mask])

array([6, 8])
```

Numpy for loops - Example Four: slicing and more

a = np.array([1,2,3,4,5])

Indexes = [2, 3]

print(a[indexes])

array([3,4])

a = np.array([1, 2, 3, 4])

mask = np.array([True, False, True, False])

print a[mask]

array([1,3])

b = np.array([5, 6, 7, 8, 9, 10, 11, 12, 13])

mask = (b%2 == 0) & (b < 10)

print(b[mask])

array([6, 8])

Numpy for Loops - combinations are endless!

```
m = np.arange(6).reshape(2,3)

print(m)

array([[0, 1, 2],

       [3, 4, 5]])

print(m[m.sum(axis=1) > 4, 1:])

array([[4, 5]])
```

Want to know more?

Experience. Numpy is neat, efficient and useful.

machinelearningplus.com has 101 numpy problems and solutions to test yourself

Pandas!

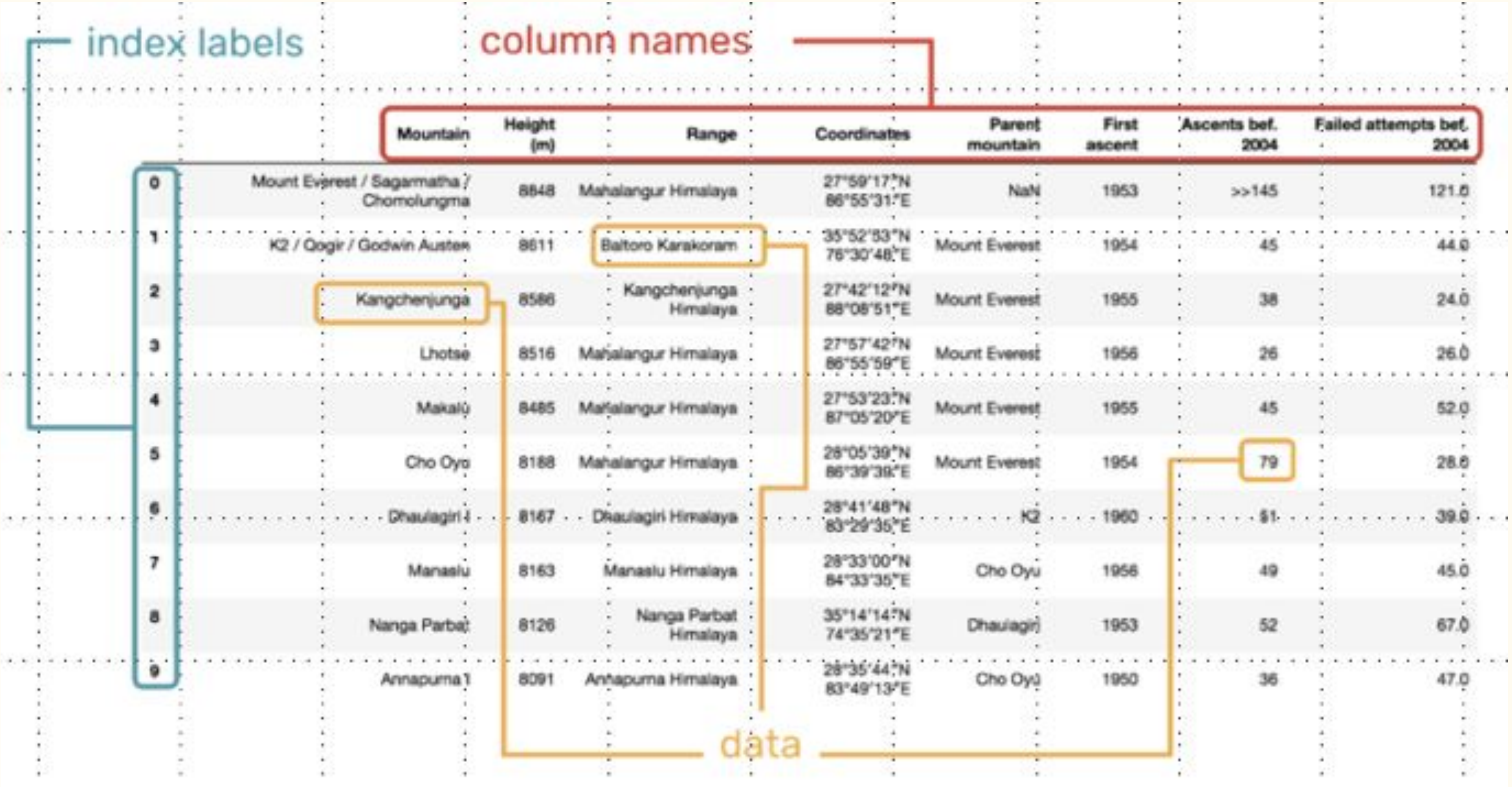
Name comes from “panel data”

Built an top of numpy!

Standard tool for reading, and manipulating data.

Format recently adopted (and improved) by ROOT RDataFrame

What does it look like?



	Mountain	Height (m)	Range	Coordinates	Parent mountain	First ascent	Ascents bef. 2004	Failed attempts bef. 2004
0	Mount Everest / Sagarmatha / Chomolungma	8848	Mahalangur Himalaya	27°59′17″N 86°55′31″E	NaN	1953	>>145	121.0
1	K2 / Qogir / Godwin Austen	8611	Baltoro Karakoram	35°52′53″N 76°30′48″E	Mount Everest	1954	45	44.0
2	Kangchenjunga	8586	Kangchenjunga Himalaya	27°42′12″N 88°08′51″E	Mount Everest	1955	38	24.0
3	Lhotse	8516	Mahalangur Himalaya	27°57′42″N 86°55′59″E	Mount Everest	1956	26	26.0
4	Makalu	8485	Mahalangur Himalaya	27°53′23″N 87°05′20″E	Mount Everest	1955	45	52.0
5	Cho Oyu	8188	Mahalangur Himalaya	28°05′39″N 86°39′39″E	Mount Everest	1954	79	28.0
6	Dhaulagiri I	8167	Dhaulagiri Himalaya	28°41′48″N 83°29′35″E	K2	1960	51	39.0
7	Manaslu	8163	Manaslu Himalaya	28°33′00″N 84°33′35″E	Cho Oyu	1956	49	45.0
8	Nanga Parbat	8126	Nanga Parbat Himalaya	35°14′14″N 74°35′21″E	Dhaulagiri	1953	52	67.0
9	Annapurna I	8091	Annapurna Himalaya	28°35′44″N 83°49′13″E	Cho Oyu	1950	36	47.0

Why pandas? - not essential but convenient

- Tools for reading and writing data
- Data alignment and integrated handling of missing data
- Ability to perform arithmetic operations
- Easy reshaping and pivoting of datasets
- User-friendly operations for merging and joining data
- Ability to handle time series

A return to Numpy!

Numpy is especially effective for generating data points e.g:

```
np.arange(6)              ->  array([0, 1, 2, 3, 4, 5])

np.linspace(9, 12, 6)     ->  array([9., 9.6, 10.2, 10.8, 11.4, 12.])
```

Or whole random distributions

```
np.random.normal(0, 1, 1000)              #  1000 random gaussian distributed points

gauss = np.random.normal(mean, sigma, 1000)   #  lots of data

sample = np.random.choice(gauss, 5)       #  5 random points from the data
```

pyplot!

Lots of programs available. Most common is pyplot from matplotlib

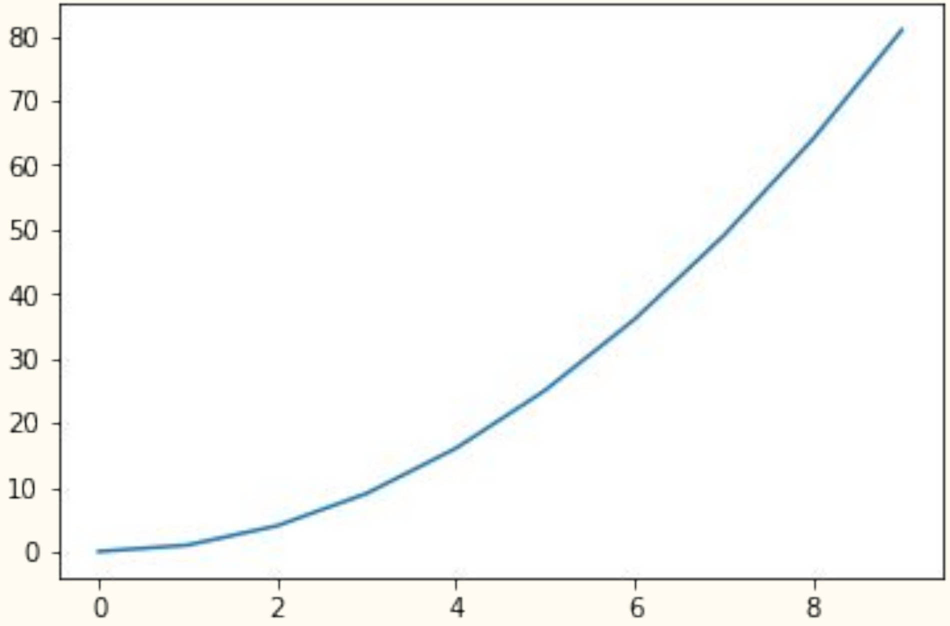
```
import matplotlib.pyplot as plt

x = range(10)

y = [i*i for i in x]

plt.plot(x,y)

plt.show()
```



Machine Learning with Sklearn

There are literally hundreds of classifiers and regressors in the scikit-learn package

```
from sklearn import classifier

clf = classifier(options)

clf.fit(training_data, training_target)

prediction = clf.predict(test_data)

rms_error = np.mean(np.pow((prediction - test_target), 2))
```

Want to know more?

https://www.kaggle.com

Kaggle is the world's largest community of data scientists and machine learners. Kaggle offers machine learning competitions and now also offers a public data platform, a cloud-based workbench for data science, and short form AI education.

pyROOT

- ROOT Website: https://root.cern
- Material online: https://github.com/root-project/training
- More material: https://root.cern/getting-started
  - Includes a booklet for beginners: **the “ROOT Primer”**
- Reference Guide: https://root.cern/doc/master/index.html
- Forum: https://root-forum.cern.ch

ROOT in a Nutshell

- ROOT is a software framework with building blocks for:
  - Data processing
  - Data analysis
  - Data visualisation
  - Data storage

An Open Source Project

We are on github

github.com/root-project

All contributions are warmly welcome!

- ROOT is written mainly in C++ (C++11/17 standard)
  - Bindings for Python the focus here!
- Adopted in High Energy Physics and other sciences (but also industry)
  - 1 EB of data in ROOT format
  - Fits and parameters' estimations for discoveries (e.g. the Higgs)
  - Thousands of ROOT plots in scientific publications

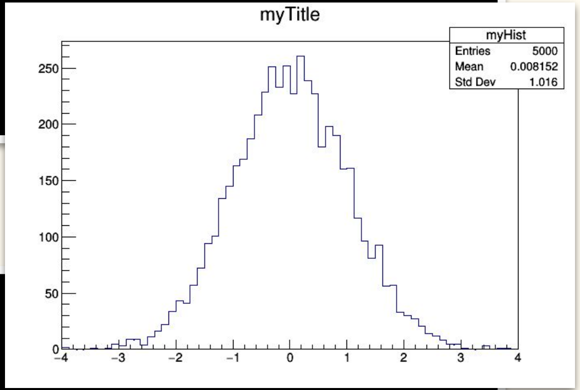
Interpreter

- ROOT has a built-in interpreter : CLING
    - C++ interpretation: highly non trivial and not foreseen by the language!
    - One of its kind: Just In Time (JIT) compilation
    - A C++ interactive shell
- Can interpret "macros" (non compiled programs)
    - Rapid prototyping possible
- ROOT provides also Python bindings
    - Will use Python interpreter directly after a simple *import ROOT*

```
$ root
root[0] 3 * 3
(const int) 9
```

Example: C++ to Python

```
> root
root [0] TH1F h("myHist", "myTitle", 64, -4, 4)
root [1] h.FillRandom("gaus")
root [2] h.Draw()
```




```
> python
>>> import ROOT
>>> h = ROOT.TH1F("myHist", "myTitle", 64, -4, 4)
>>> h.FillRandom("gaus")
>>> h.Draw()
```

Dynamic C++ (JITting)

```python
import ROOT
cpp_code = """
int f(int i) { return i*i; }
class A {
public:
  A() { cout << "Hello PyROOT!" << endl; }
};
"""
# Inject the code in the ROOT interpreter
ROOT.gInterpreter.ProcessLine(cpp_code)

# We find all the C++ entities in Python!
a = ROOT.A()    # this prints Hello PyROOT!
x = ROOT.f(3)  # x = 9
```

C++ code we want to invoke from Python