Scientific Programming with Python (2018 Edition)

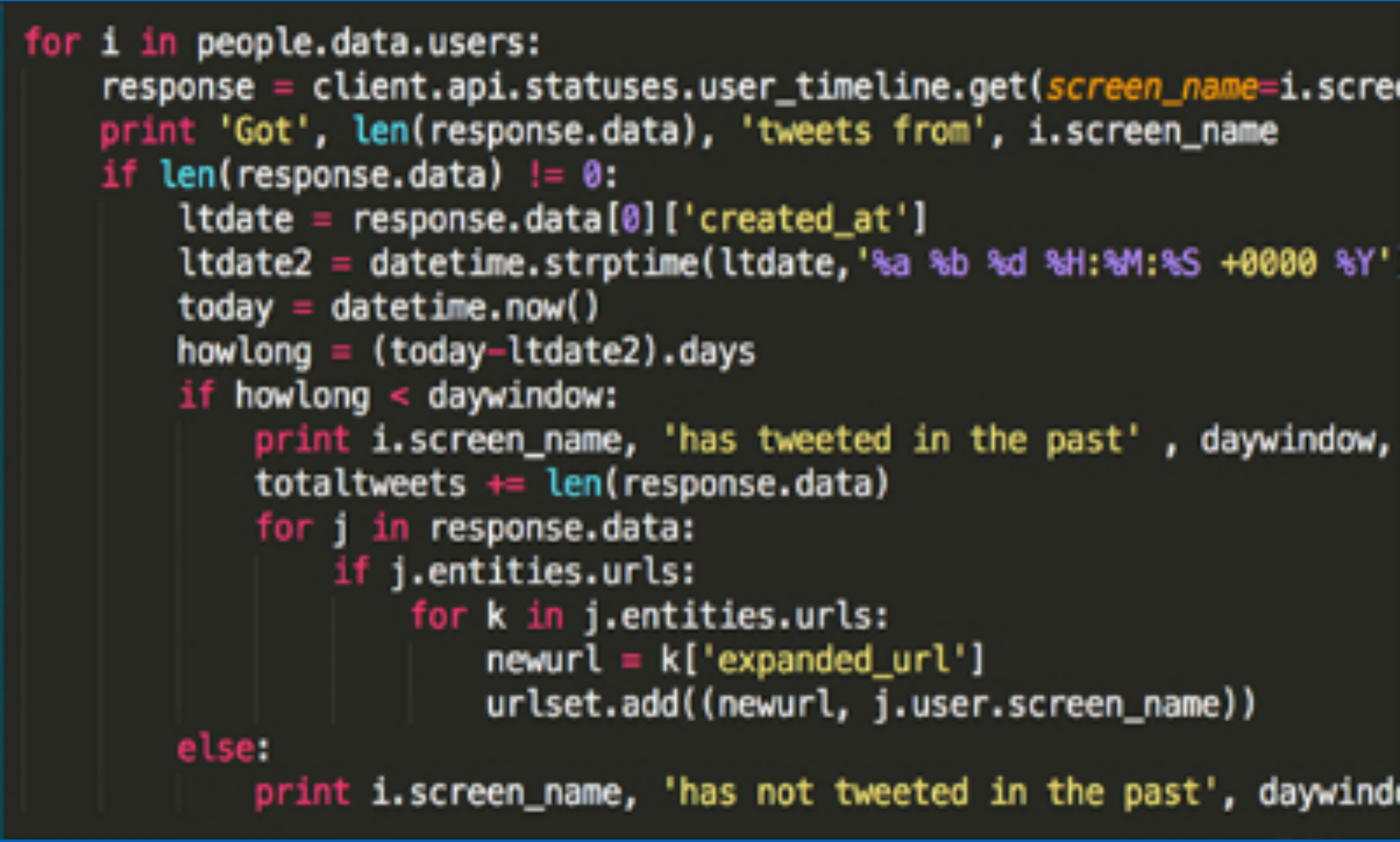
https://gdfa.ugr.es/python

Pedro Magaña (pmagana@ugr.es)

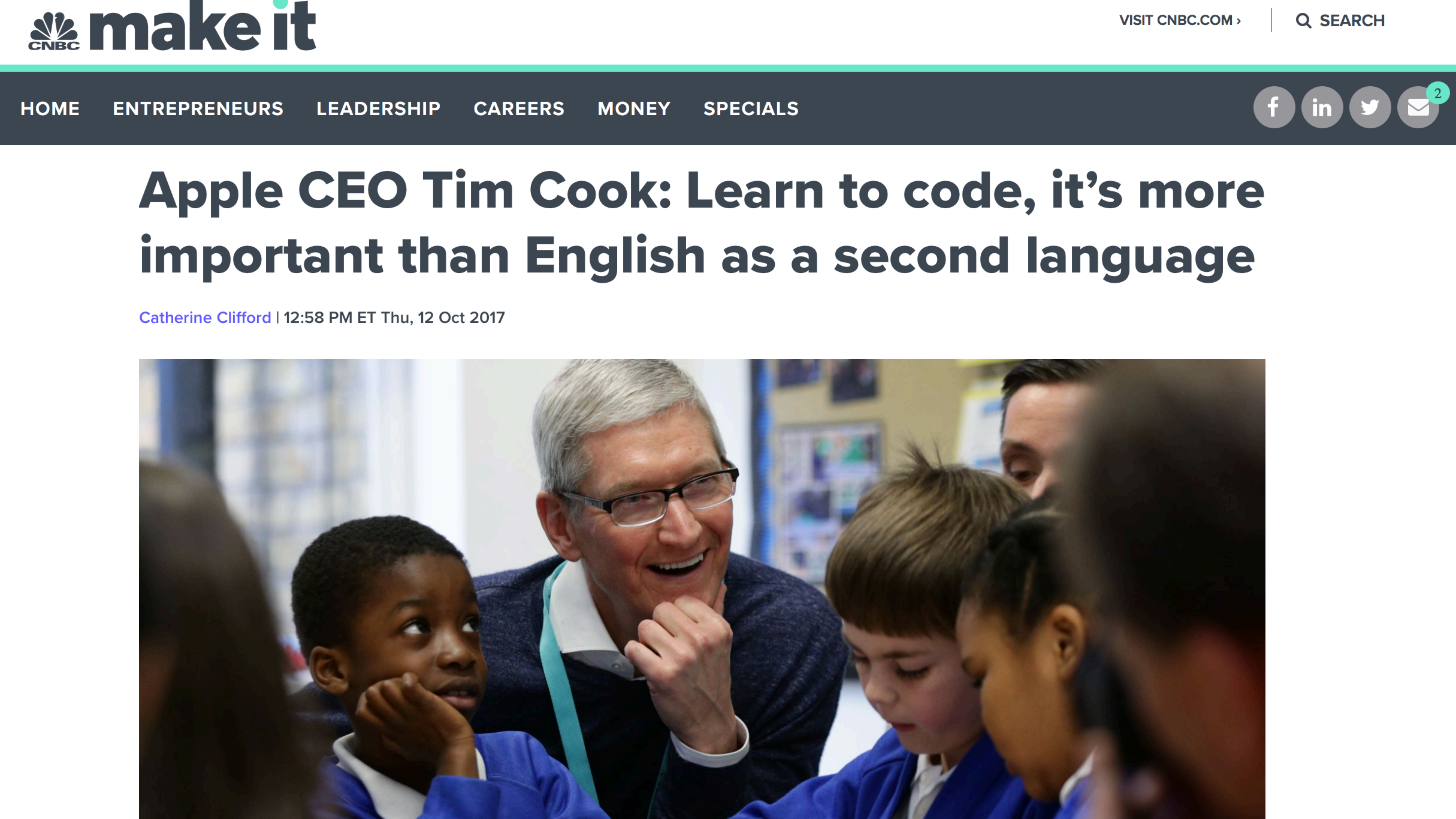
Outline

- Why learn to code?
- Introduction to Python
- Python for science, where to begin?
- Python language
- Scientific libraries

Why learn to code?

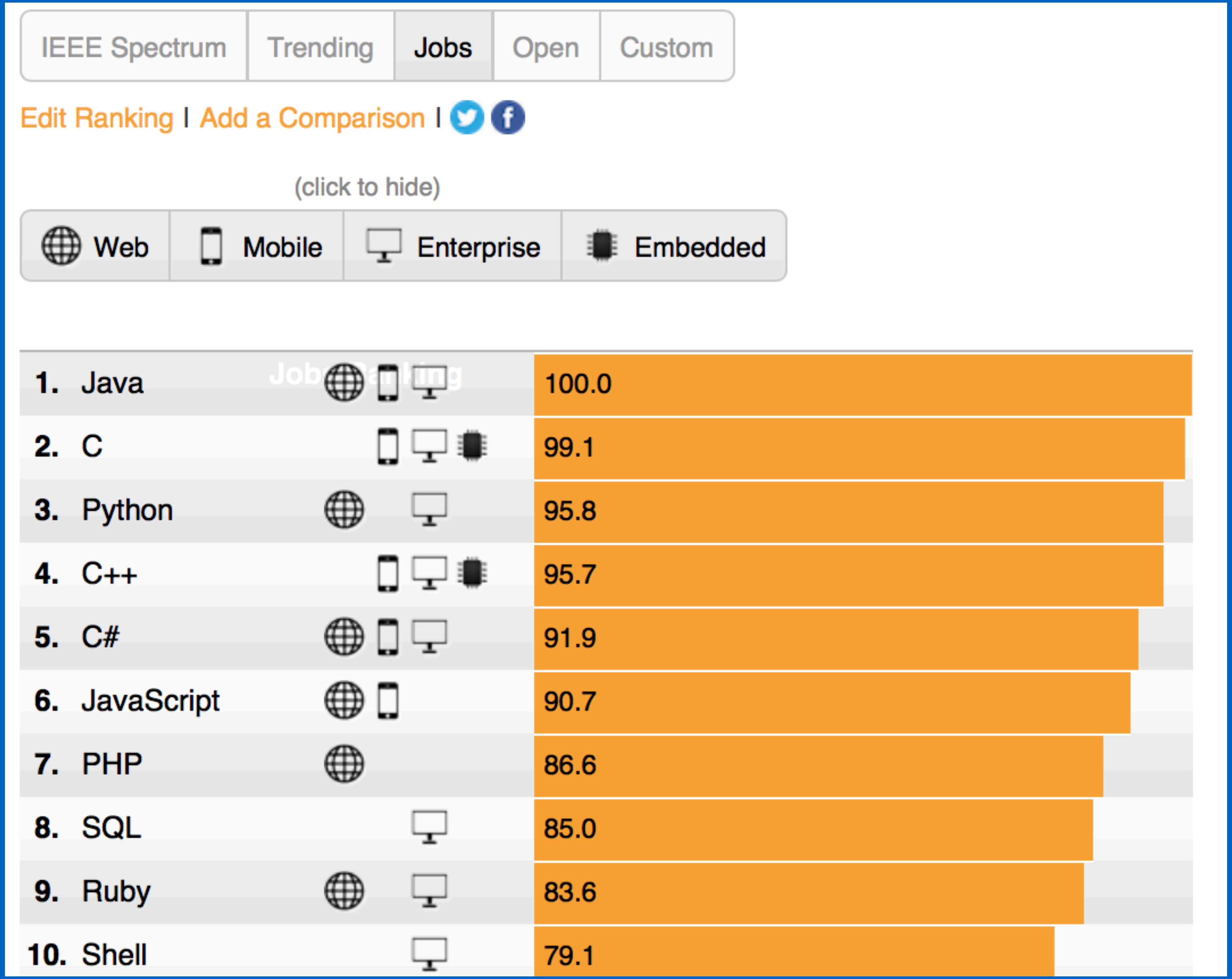


```
for i in people.data.users:
    response = client.api.statuses.user_timeline.get(screen_name=i.scre
    print 'Got', len(response.data), 'tweets from', i.screen_name
    if len(response.data) != 0:
        ltdate = response.data[0]['created_at']
        ltdate2 = datetime.strptime(ltdate,'%a %b %d %H:%M:%S +0000 %Y'
        today = datetime.now()
        howlong = (today-ltdate2).days
        if howlong < daywindow:
            print i.screen_name, 'has tweeted in the past' , daywindow,
            totaltweets += len(response.data)
            for j in response.data:
                if j.entities.urls:
                    for k in j.entities.urls:
                        newurl = k['expanded_url']
                        urlset.add((newurl, j.user.screen_name))
        else:
            print i.screen_name, 'has not tweeted in the past', daywind
```

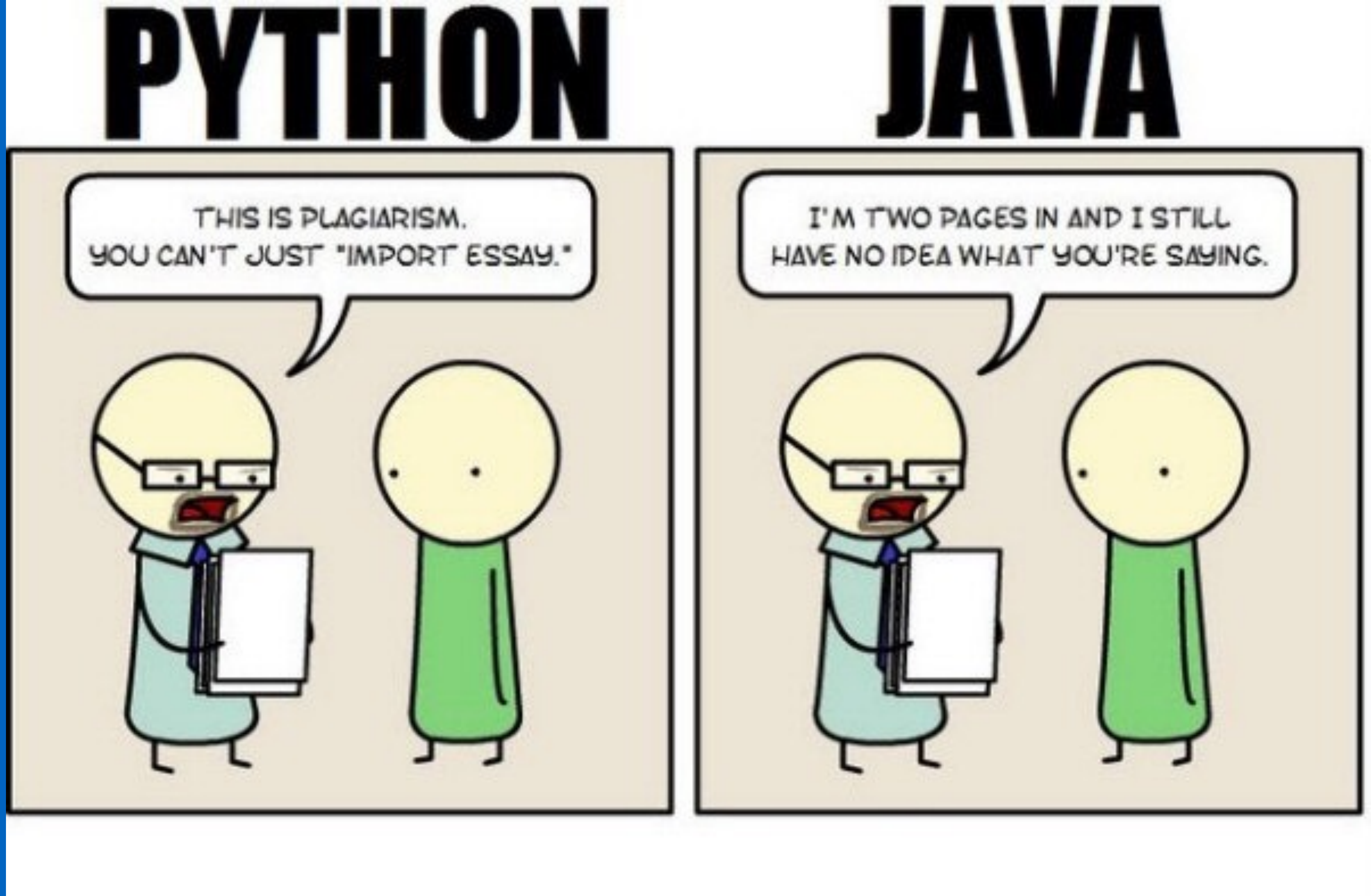


Apple CEO Tim Cook: Learn to code, it's more important than English as a second language

Catherine Clifford | 12:58 PM ET Thu, 12 Oct 2017



1.	Java	🌐 📱 🖥	100.0
2.	C	📱 🖥 ▦	99.1
3.	Python	🌐 🖥	95.8
4.	C++	📱 🖥 ▦	95.7
5.	C#	🌐 📱 🖥	91.9
6.	JavaScript	🌐 📱	90.7
7.	PHP	🌐	86.6
8.	SQL	🖥	85.0
9.	Ruby	🌐 🖥	83.6
10.	Shell	🖥	79.1


PYTHON
THIS IS PLAGIARISM. YOU CAN'T JUST "IMPORT ESSAY."
JAVA
I'M TWO PAGES IN AND I STILL HAVE NO IDEA WHAT YOU'RE SAYING.

TOOLBOX

PICK UP PYTHON

A powerful programming language with huge community support.

[ILLUSTRATION BY THE PROJECT TWINS

BY JEFFREY M. PERKEL

Last month, Adina Howe took up a post at Iowa State University in Ames. Officially, she is an assistant professor of agricultural and biosystems engineering. But she works not in the greenhouse, but in front of a keyboard. Howe is a programmer, and a key part of her job is as a 'data professor' — developing curricula to teach the next generation of graduates about the mechanics and importance of scientific programming.

Howe does not have a degree in computer science, nor does she have years of formal training. She had a PhD in environmental engineering and expertise in running enzyme assays when she joined the laboratory of Titus Brown at Michigan State University in East Lansing. Brown specializes in bioinformatics and uses computation to extract meaning from genomic data sets, and Howe had to get up to speed on the computational side. Brown's recommendation: learn Python.

Among the host of computer-programming languages that scientists might choose to pick up, Python, first released in 1991 by Dutch programmer Guido van Rossum, is an increasingly popular (and free) recommendation. It combines simple syntax, abundant online resources and a rich ecosystem of scientifically focused toolkits with a heavy emphasis on community.

HELLO, WORLD

With the explosive growth of 'big data' in disciplines such as bioinformatics, neuroscience and astronomy, programming know-how is becoming ever more crucial. Researchers who can write code in Python can deftly manage their data sets, and work much more efficiently on a whole host of research-related tasks — from crunching numbers to cleaning up, analysing and visualizing data. Whereas some programming languages, such as MATLAB and R, focus on mathematical and statistical operations, Python is a general-purpose language, along the lines of C and C++ (the languages in which much commercial software and operating systems are written). As such, it is perhaps more complicated, Brown says, but also more capable: it is amenable to everything from automating small sets of instructions, to building websites, to fully fledged applications. Jessica Hamrick, a psychology PhD student at the University of California, Berkeley, has been ▶

Programming: Pick up Python

A powerful programming language with huge community support.

Jeffrey M. Perkel

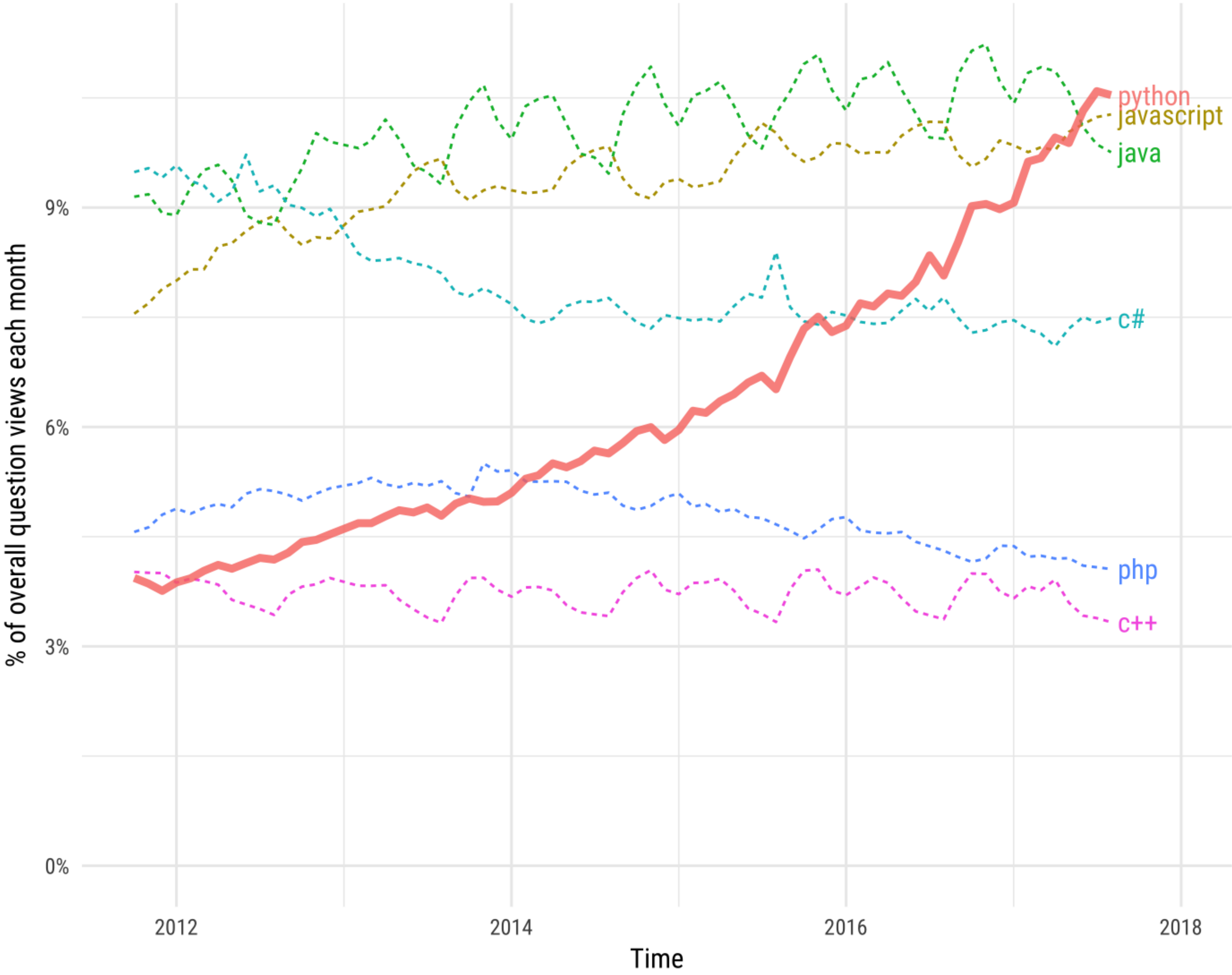
04 February 2015

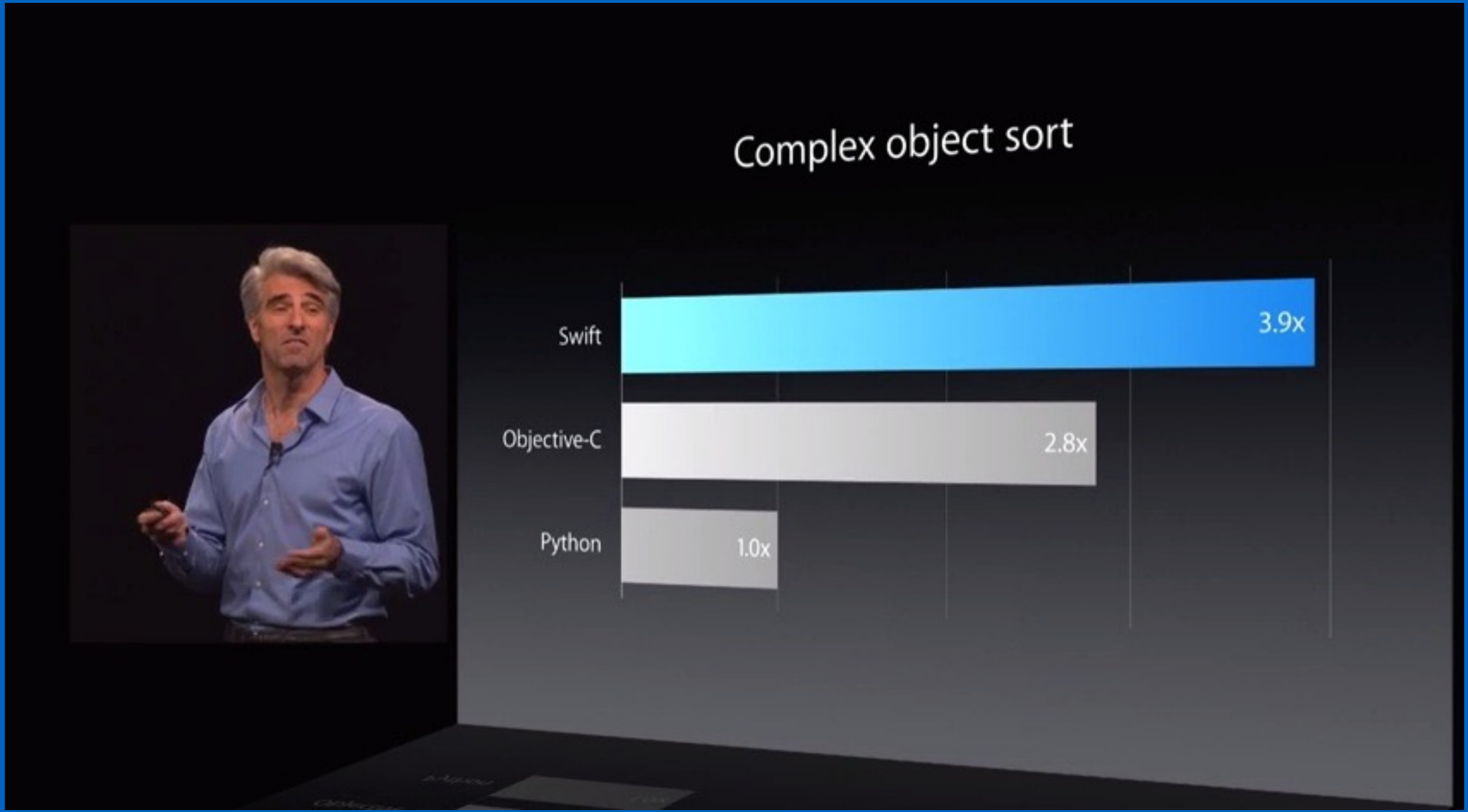
Nature

http://doi.org/10.1038/518125a

Growth of major programming languages

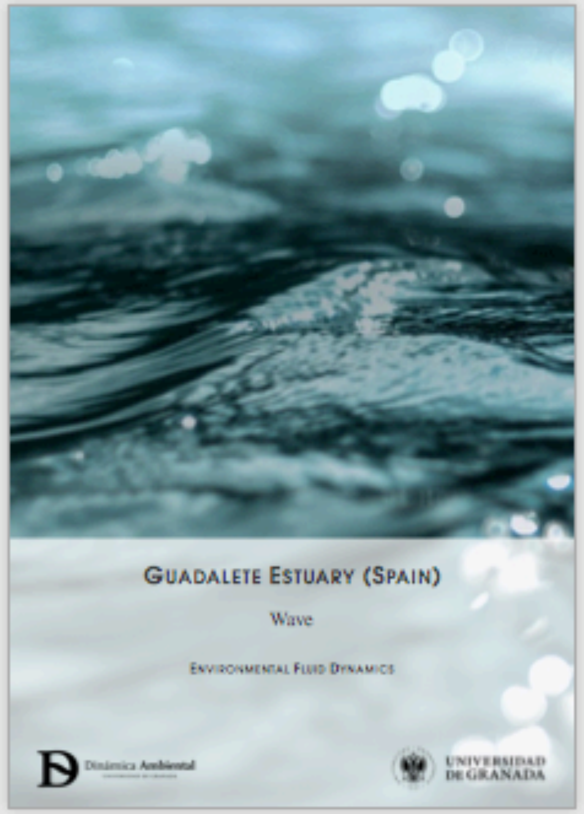
Based on Stack Overflow question views in World Bank high-income countries





Complex object sort

Samples



Guadalete Estuary (Spain)

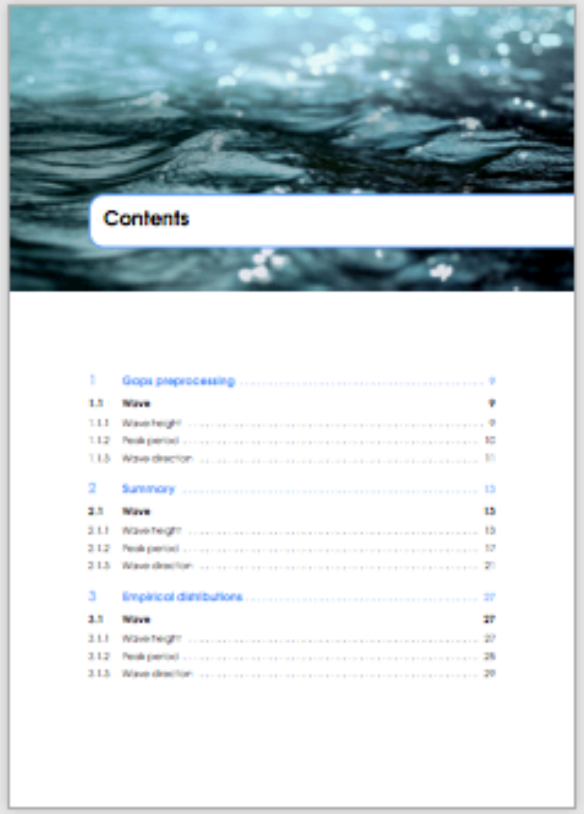
Wave

Environmental Fluid Dynamics

Dinámica Ambiental

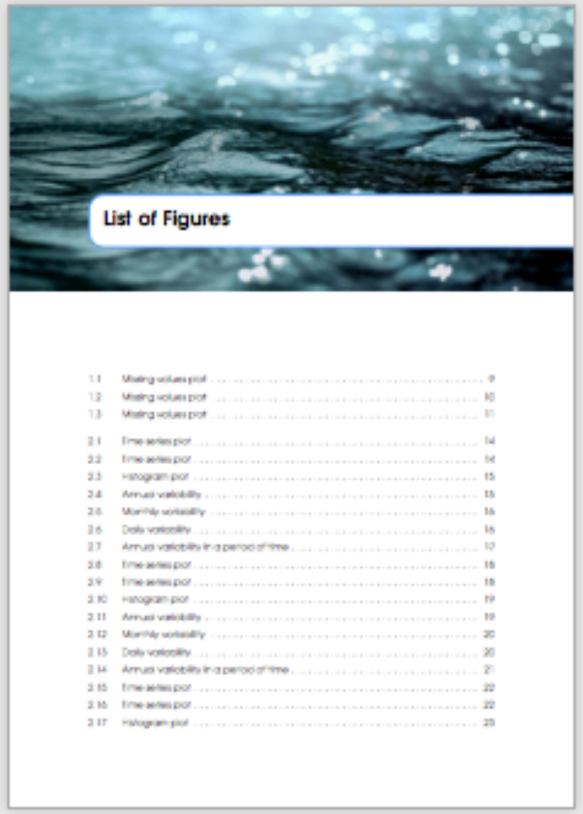
Universidad de Granada

1



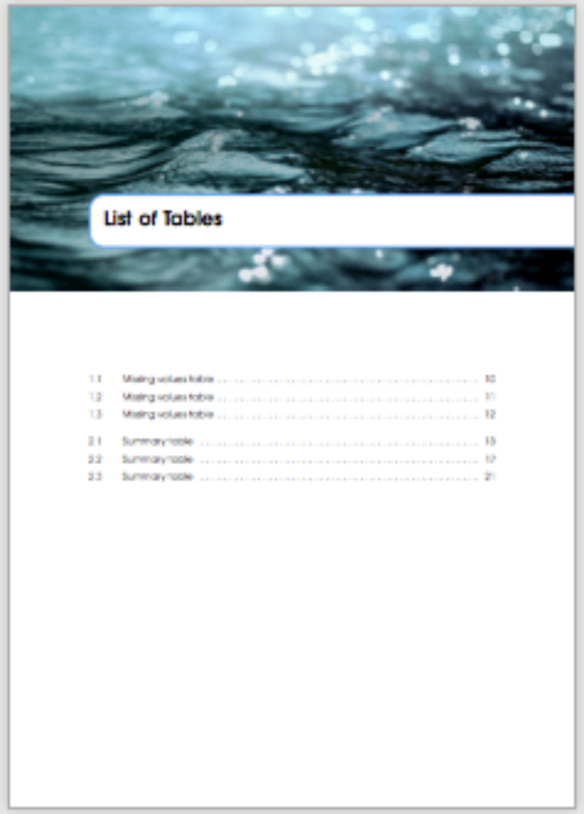
Contents

2



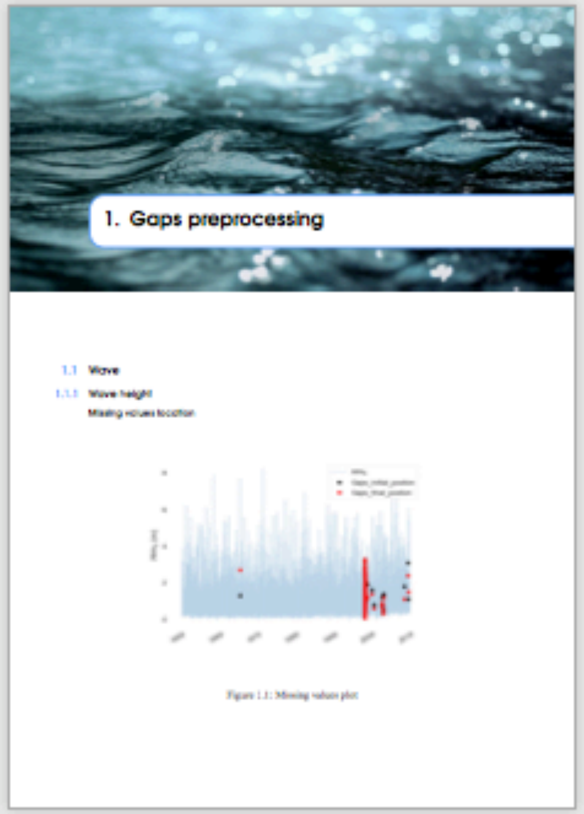
List of Figures

3



List of Tables

4



1. Gaps preprocessing

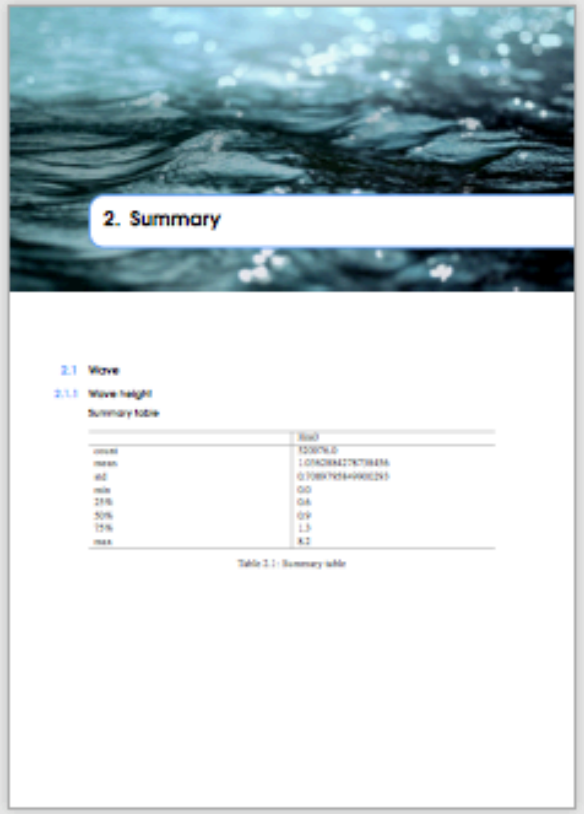
1.1 Wave

1.1.1 Wave height

Missing values location

Figure 1.1: Missing values plot

5



2. Summary

2.1 Wave

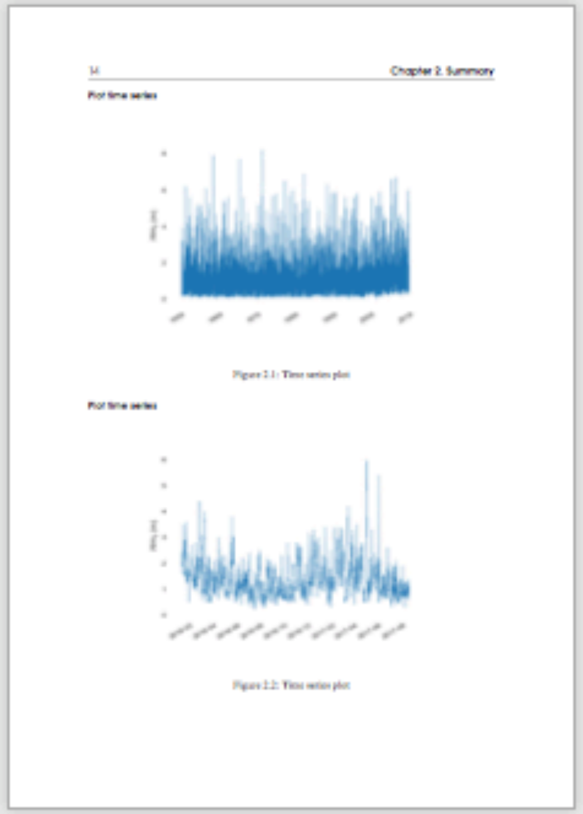
2.1.1 Wave height

Summary table

	Hm0
count	520076.0
mean	1.0762884278738456
std	0.7089795849900293
min	0.0
25%	0.6
50%	0.9
75%	1.3
max	8.2

Table 2.1: Summary table

6



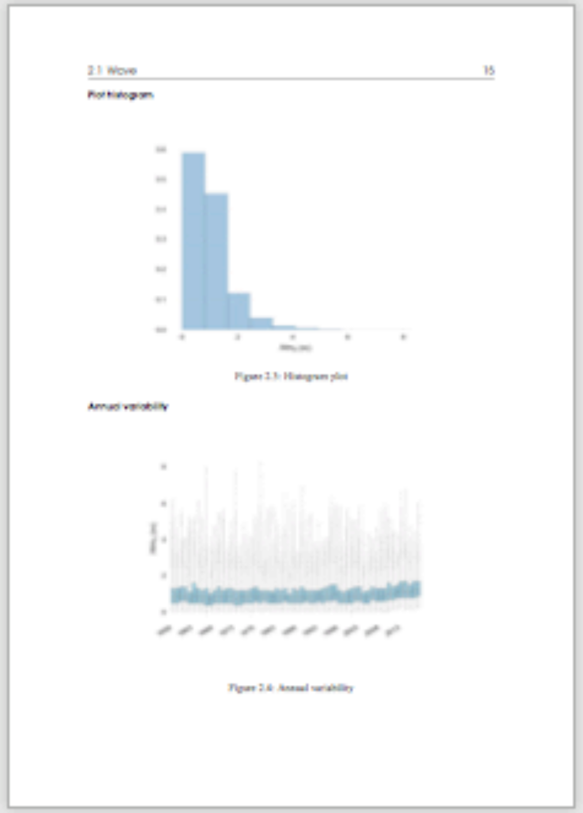
Plot time series

Figure 2.1: Time series plot

Plot time series

Figure 2.2: Time series plot

7



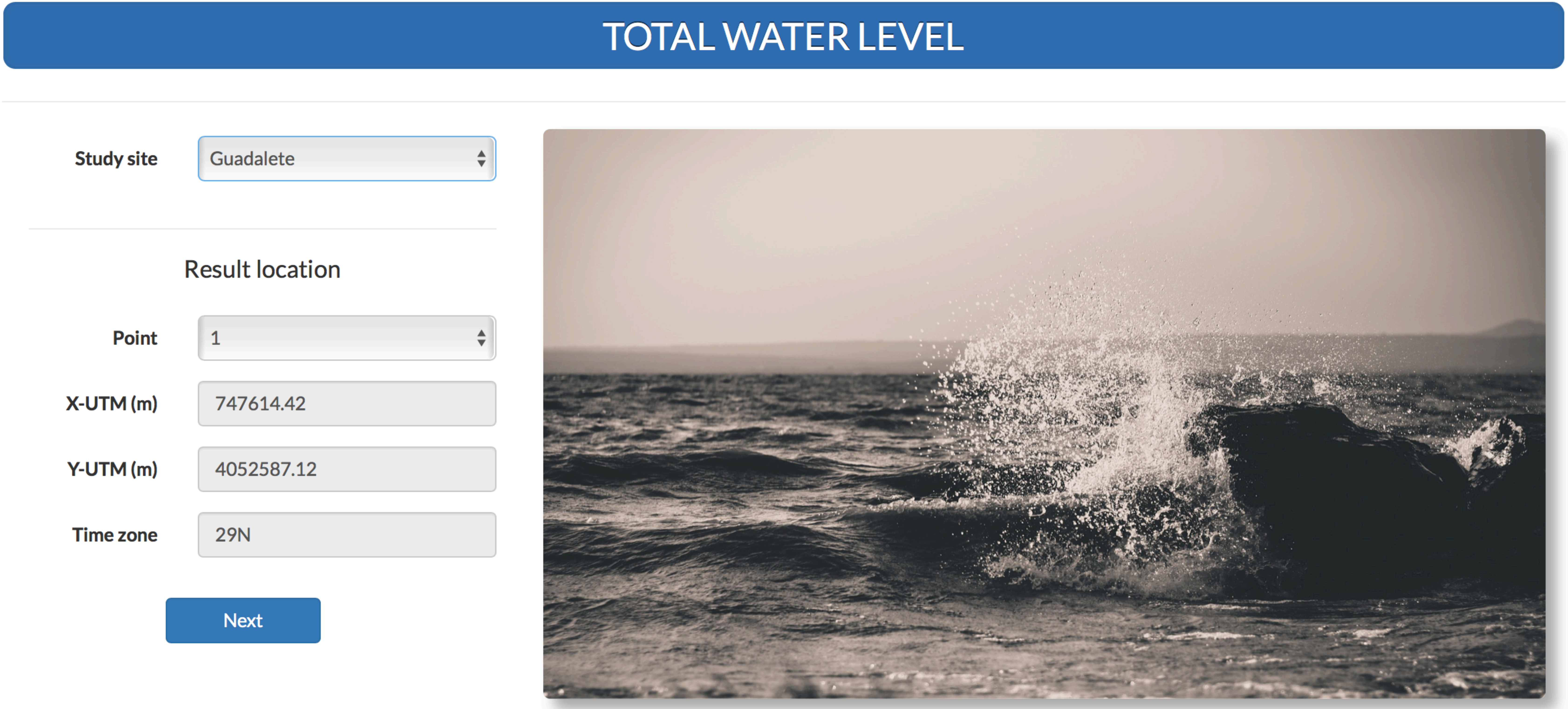
Plot histogram

Figure 2.3: Histogram plot

Annual variability

Figure 2.4: Annual variability

8

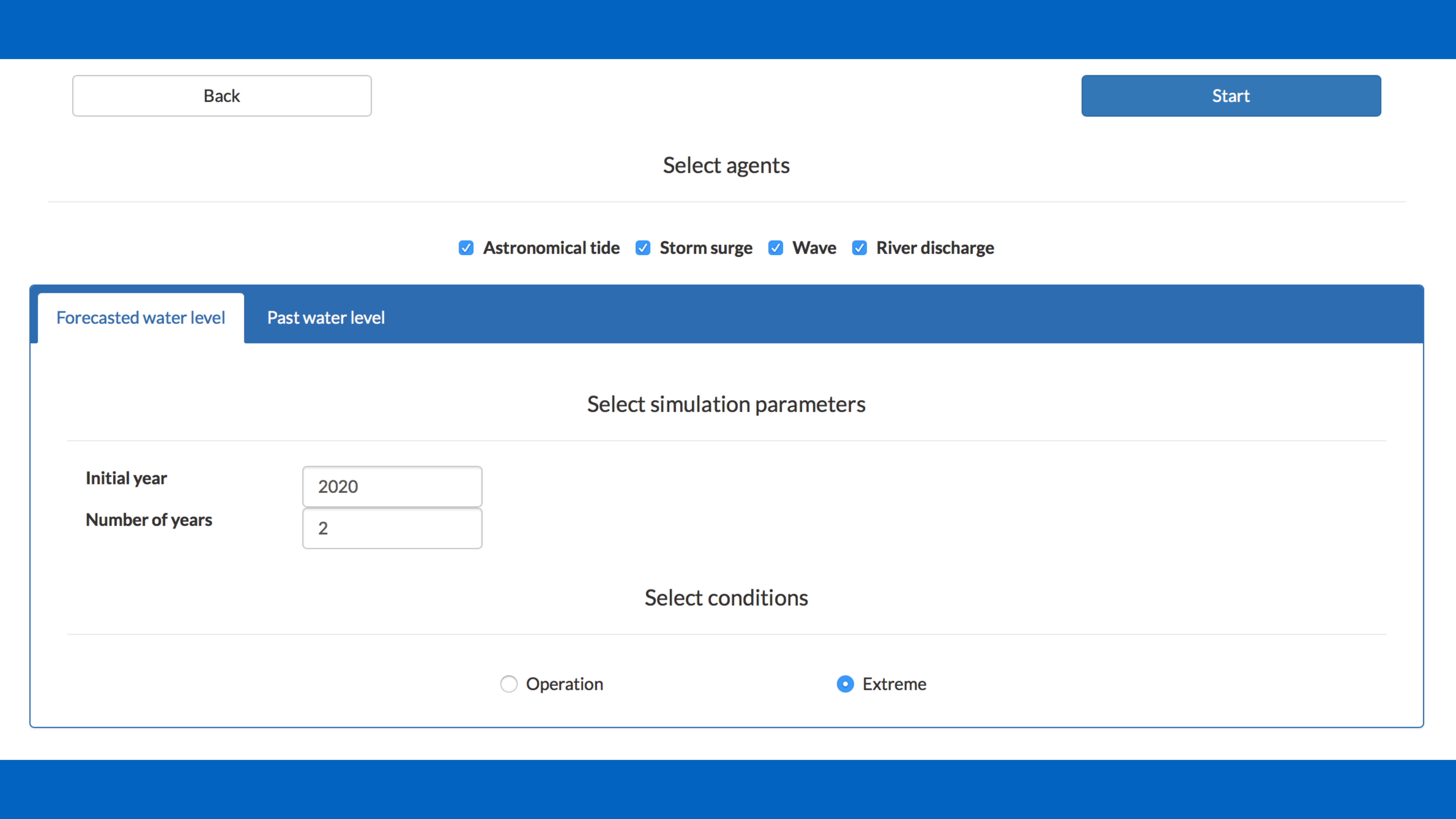


TOTAL WATER LEVEL

Study site	Guadalete

Result location

Point	1
X-UTM (m)	747614.42
Y-UTM (m)	4052587.12
Time zone	29N



Select agents

☑ Astronomical tide ☑ Storm surge ☑ Wave ☑ River discharge

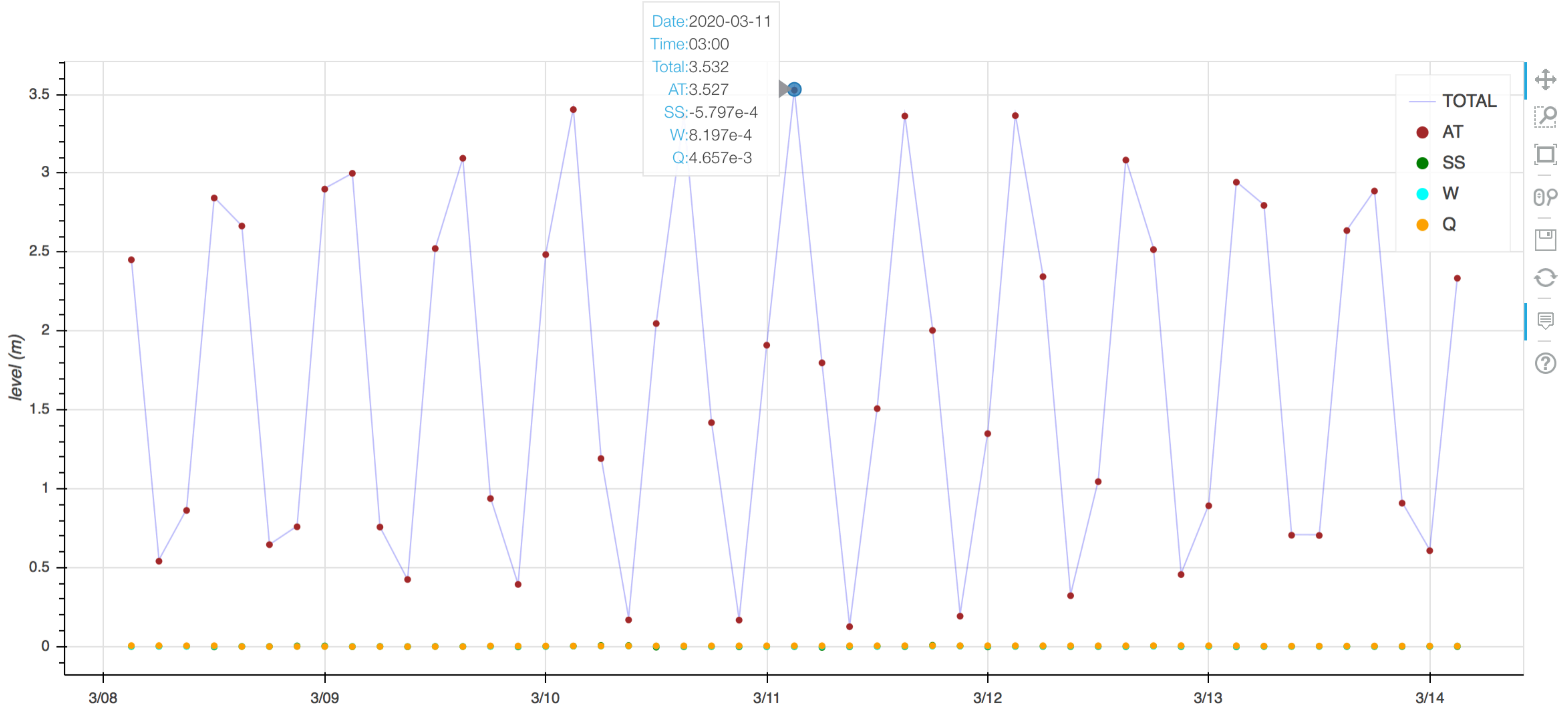
Select simulation parameters

Initial year: 2020

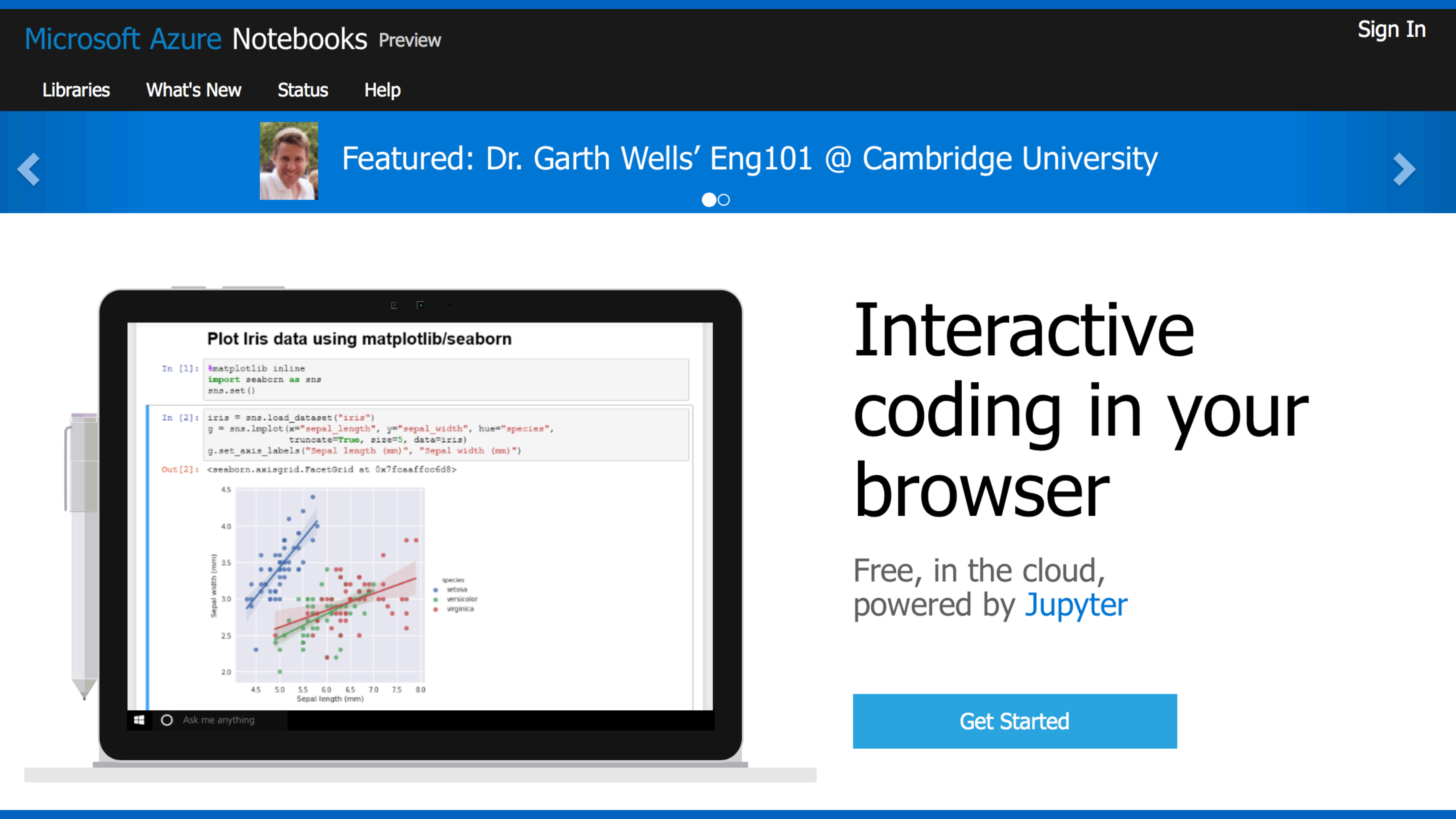
Number of years: 2

Select conditions

○ Operation ● Extreme



Date:2020-03-11
Time:03:00
Total:3.532
AT:3.527
SS:-5.797e-4
W:8.197e-4
Q:4.657e-3



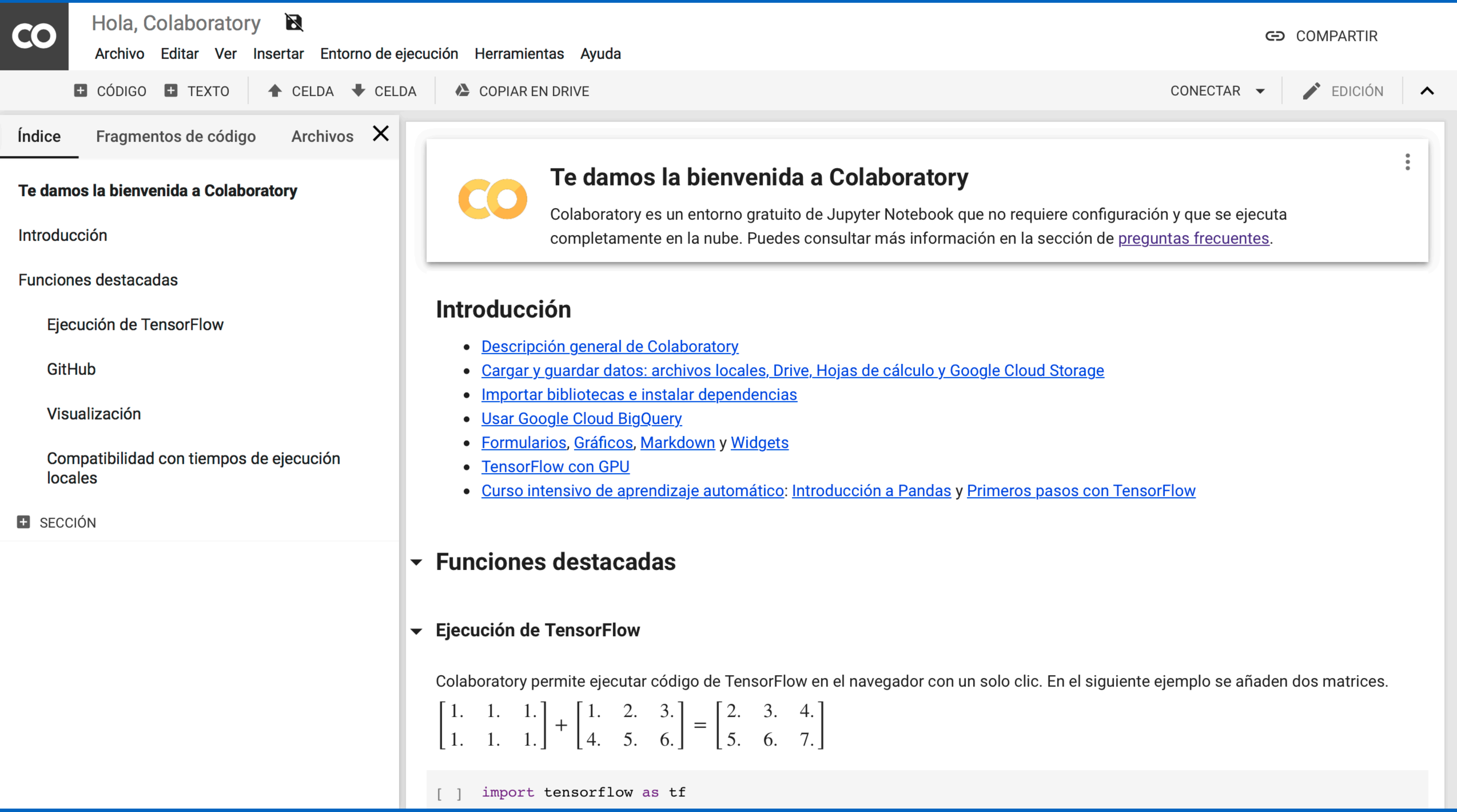
Featured: Dr. Garth Wells' Eng101 @ Cambridge University

Plot Iris data using matplotlib/seaborn

```
In [1]: %matplotlib inline
        import seaborn as sns
        sns.set()

In [2]: iris = sns.load_dataset("iris")
        g = sns.lmplot(x="sepal_length", y="sepal_width", hue="species",
                       truncate=True, size=5, data=iris)
        g.set_axis_labels("Sepal length (mm)", "Sepal width (mm)")

Out[2]: <seaborn.axisgrid.FacetGrid at 0x7fcaaffcc6d8>
```

Interactive coding in your browser

Free, in the cloud, powered by Jupyter



Te damos la bienvenida a Colaboratory

Colaboratory es un entorno gratuito de Jupyter Notebook que no requiere configuración y que se ejecuta completamente en la nube. Puedes consultar más información en la sección de preguntas frecuentes.

Introducción

- Descripción general de Colaboratory
- Cargar y guardar datos: archivos locales, Drive, Hojas de cálculo y Google Cloud Storage
- Importar bibliotecas e instalar dependencias
- Usar Google Cloud BigQuery
- Formularios, Gráficos, Markdown y Widgets
- TensorFlow con GPU
- Curso intensivo de aprendizaje automático: Introducción a Pandas y Primeros pasos con TensorFlow

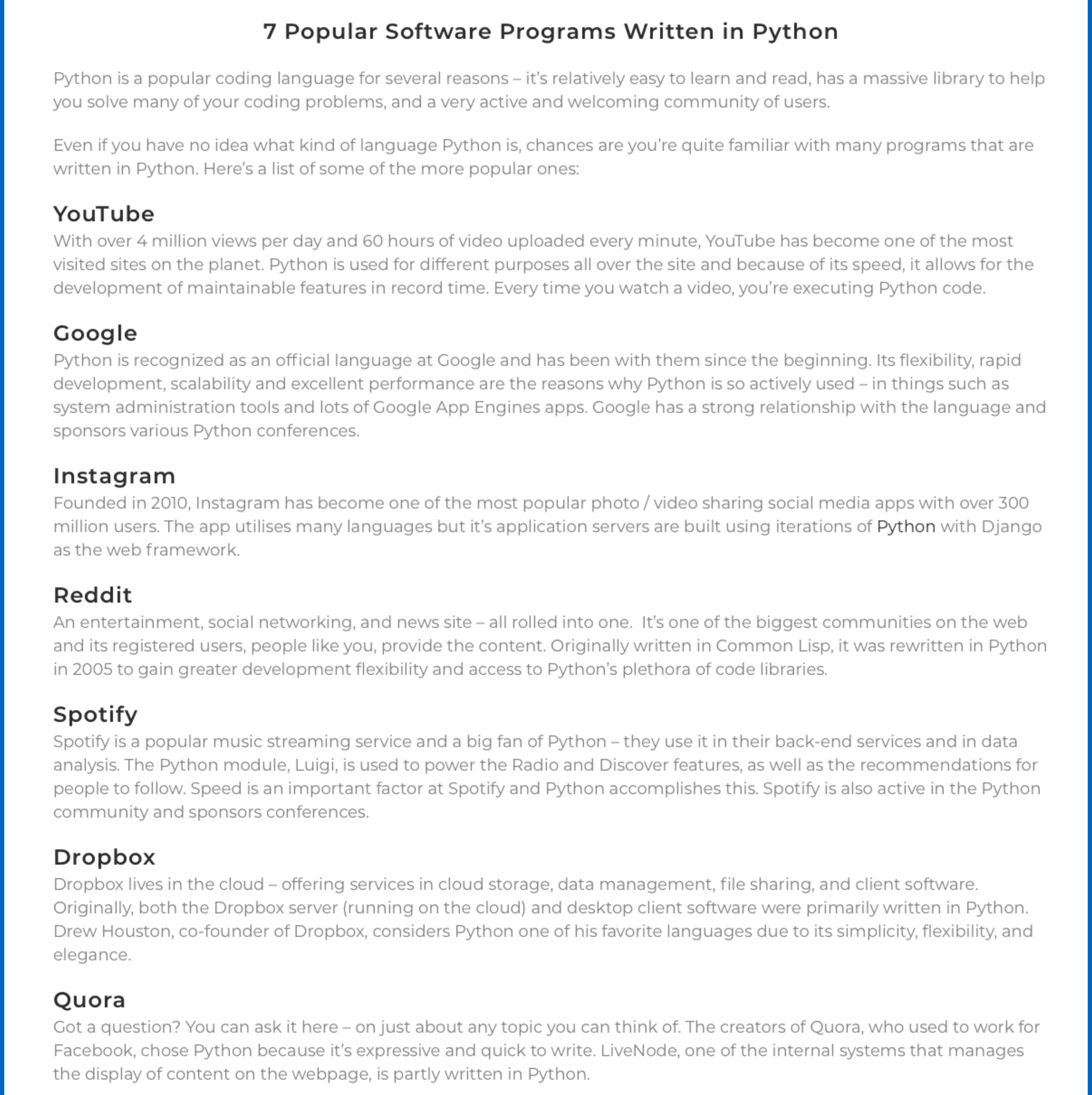
Funciones destacadas

Ejecución de TensorFlow

Colaboratory permite ejecutar código de TensorFlow en el navegador con un solo clic. En el siguiente ejemplo se añaden dos matrices.

$$\begin{bmatrix} 1. & 1. & 1. \\ 1. & 1. & 1. \end{bmatrix} + \begin{bmatrix} 1. & 2. & 3. \\ 4. & 5. & 6. \end{bmatrix} = \begin{bmatrix} 2. & 3. & 4. \\ 5. & 6. & 7. \end{bmatrix}$$

```
[ ]  import tensorflow as tf
```



7 Popular Software Programs Written in Python

Python is a popular coding language for several reasons – it's relatively easy to learn and read, has a massive library to help you solve many of your coding problems, and a very active and welcoming community of users.

Even if you have no idea what kind of language Python is, chances are you're quite familiar with many programs that are written in Python. Here's a list of some of the more popular ones:

YouTube

With over 4 million views per day and 60 hours of video uploaded every minute, YouTube has become one of the most visited sites on the planet. Python is used for different purposes all over the site and because of its speed, it allows for the development of maintainable features in record time. Every time you watch a video, you're executing Python code.

Google

Python is recognized as an official language at Google and has been with them since the beginning. Its flexibility, rapid development, scalability and excellent performance are the reasons why Python is so actively used – in things such as system administration tools and lots of Google App Engines apps. Google has a strong relationship with the language and sponsors various Python conferences.

Instagram

Founded in 2010, Instagram has become one of the most popular photo / video sharing social media apps with over 300 million users. The app utilises many languages but it's application servers are built using iterations of Python with Django as the web framework.

Reddit

An entertainment, social networking, and news site – all rolled into one. It's one of the biggest communities on the web and its registered users, people like you, provide the content. Originally written in Common Lisp, it was rewritten in Python in 2005 to gain greater development flexibility and access to Python's plethora of code libraries.

Spotify

Spotify is a popular music streaming service and a big fan of Python – they use it in their back-end services and in data analysis. The Python module, Luigi, is used to power the Radio and Discover features, as well as the recommendations for people to follow. Speed is an important factor at Spotify and Python accomplishes this. Spotify is also active in the Python community and sponsors conferences.

Dropbox

Dropbox lives in the cloud – offering services in cloud storage, data management, file sharing, and client software. Originally, both the Dropbox server (running on the cloud) and desktop client software were primarily written in Python. Drew Houston, co-founder of Dropbox, considers Python one of his favorite languages due to its simplicity, flexibility, and elegance.

Quora

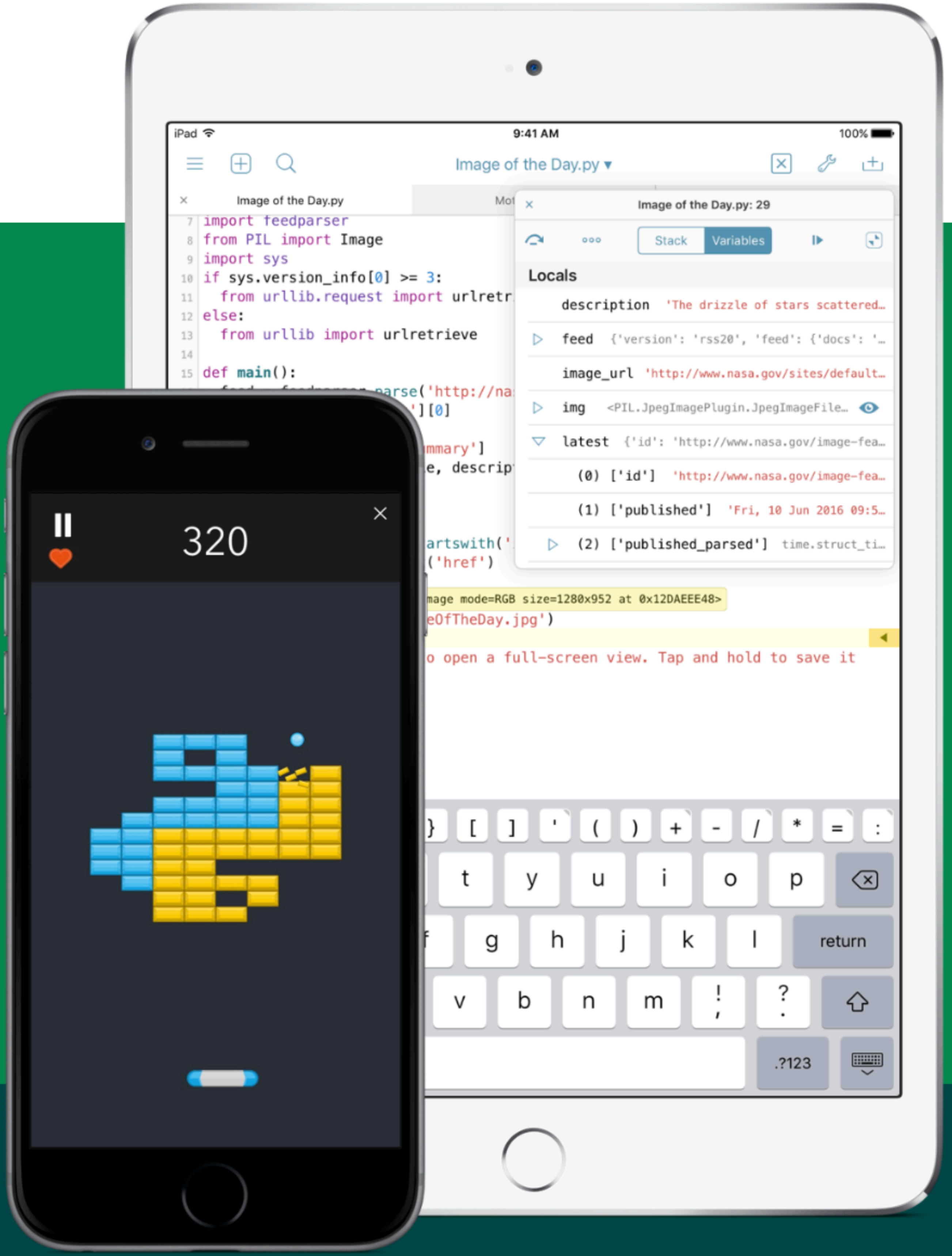
Got a question? You can ask it here – on just about any topic you can think of. The creators of Quora, who used to work for Facebook, chose Python because it's expressive and quick to write. LiveNode, one of the internal systems that manages the display of content on the webpage, is partly written in Python.

Pythonista 3

A Full Python IDE for iOS

Pythonista is a complete development environment for writing Python™ scripts on your iPad or iPhone. Lots of examples are included — from games and animations to plotting, image manipulation, custom user interfaces, and automation scripts.

In addition to the powerful standard library, Pythonista provides extensive support for interacting with native iOS features, like contacts, reminders, photos, location data, and more.



```
import feedparser
from PIL import Image
import sys
if sys.version_info[0] >= 3:
  from urllib.request import urlretr
else:
  from urllib import urlretrieve

def main():
```

```
description   'The drizzle of stars scattered…
feed   {'version': 'rss20', 'feed': {'docs': '…
image_url   'http://www.nasa.gov/sites/default…
img   <PIL.JpegImagePlugin.JpegImageFile…
latest   {'id': 'http://www.nasa.gov/image-fea…
  (0) ['id']   'http://www.nasa.gov/image-fea…
  (1) ['published']   'Fri, 10 Jun 2016 09:5…
  (2) ['published_parsed']   time.struct_ti…
```

```
mage mode=RGB size=1280x952 at 0x12DAEEE48>
eOfTheDay.jpg')
o open a full-screen view. Tap and hold to save it
```

Download on the App Store

Universal App for iPhone + iPad

Introduction to Python

What is Python?

Python is a modern, general-purpose, object-oriented, high-level programming language.

General characteristics of Python:

- **clean and simple language:** Easy-to-read and intuitive code, easy-to-learn minimalistic syntax, maintainability scales well with size of projects.
- **expressive language:** Fewer lines of code, fewer bugs, easier to maintain.

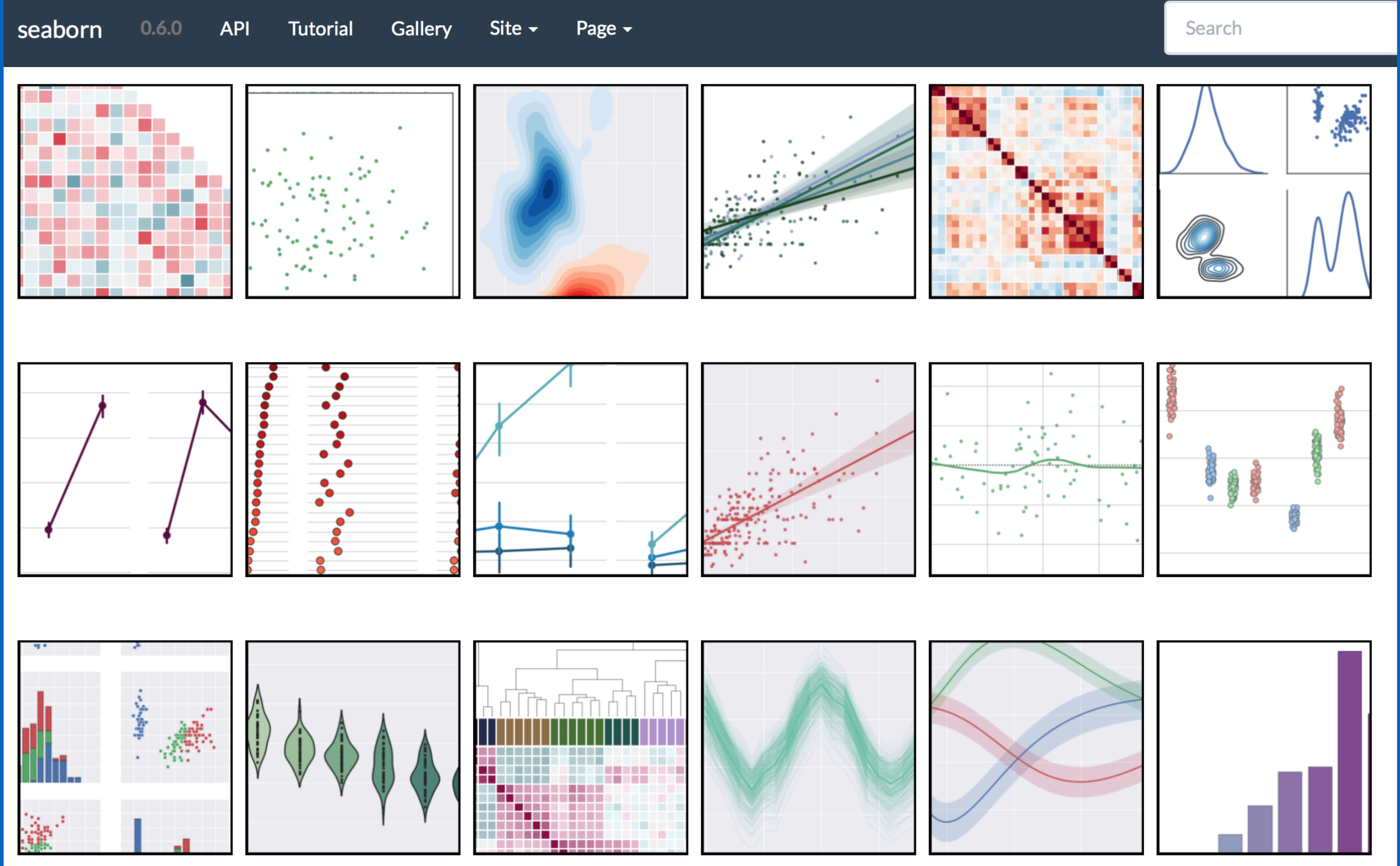
Advantages:

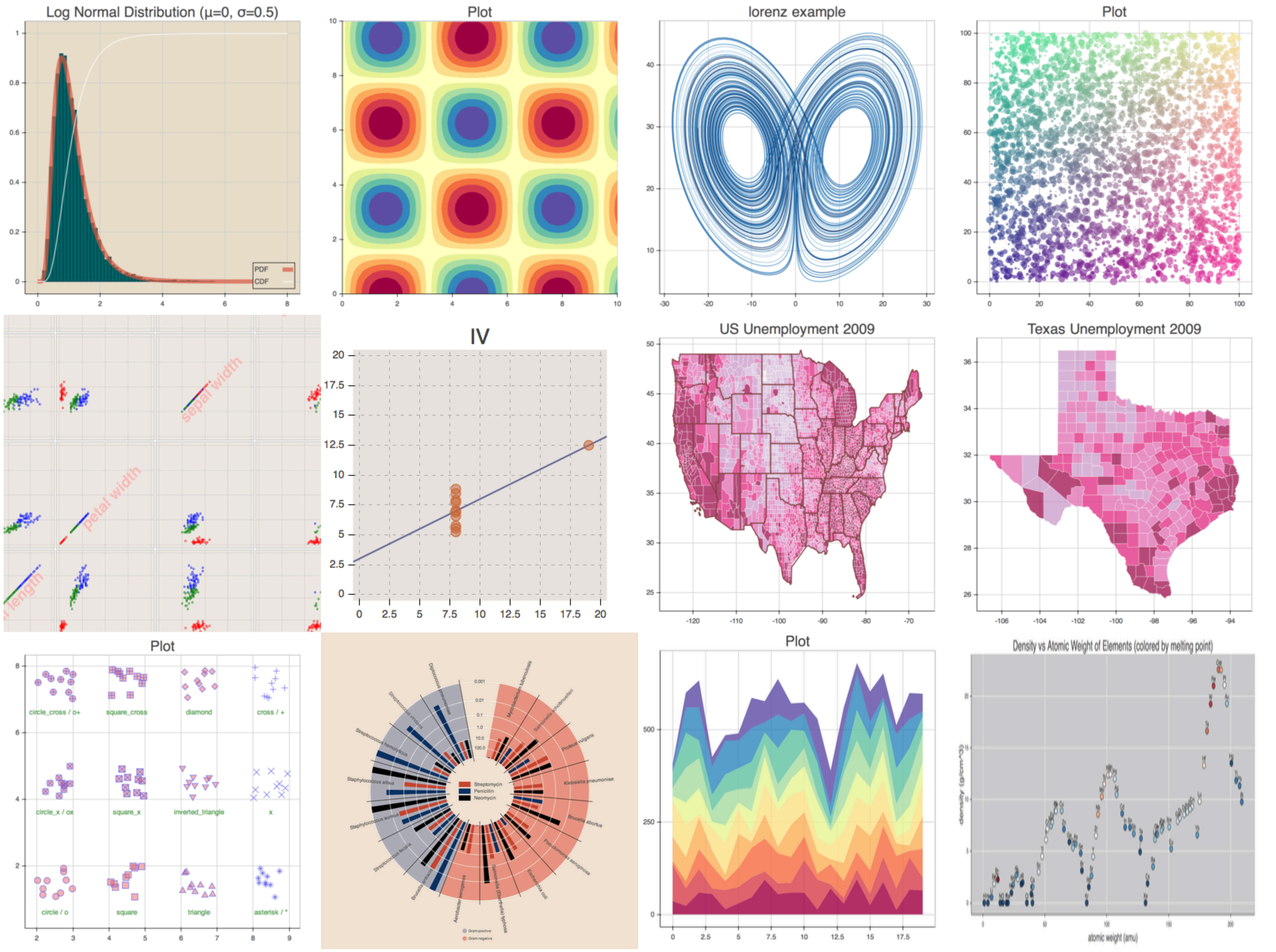
- The main advantage is **ease of programming**, minimizing the time required to develop, debug and maintain the code.
- Well designed language that **encourage many good programming practices**:
  - **Modular** and object-oriented programming, good system for packaging and re-use of code. This often results in more transparent, maintainable and bug-free code.
  - **Documentation tightly integrated with the code**.
- A large standard library, and a **large collection of add-on packages**.
- Packaging of programs into **standard executables**, that **work on computers without Python** installed.

Disadvantages:

- Since Python is an interpreted and dynamically typed programming language, **the execution of python code can be slow** compared to compiled statically typed programming languages, such as C/C++ and Fortran.
- Somewhat decentralized, with **different environment, packages and documentation spread out at different places**. Can make it harder to get started.

- Python has a strong position in scientific computing
  - **Large community of users**, easy to find help and documentation.
- Extensive ecosystem of **scientific libraries**
  - NumPy: numerical Python ≈ MATLAB matrices and arrays
  - SciPy: scientific Python ≈ MATLAB toolboxes
  - pandas: extends NumPy
  - Matplotlib: graphics library
  - Sympy: symbolic mathematics library


seaborn
0.6.0
API
Tutorial
Gallery
Site
Page
Search


Log Normal Distribution (μ=0, σ=0.5)
PDF
CDF
Plot
lorenz example
Plot
sepal width
petal width
length
IV
US Unemployment 2009
Texas Unemployment 2009
Plot
circle_cross / o+
square_cross
diamond
cross / +
circle_x / ox
square_x
inverted_triangle
x
circle / o
square
triangle
asterisk / *
Streptomycin
Penicillin
Neomycin
Gram-positive
Gram-negative
Plot
Density vs Atomic Weight of Elements (colored by melting point)
density (g/cm^3)
atomic weight (amu)

- **Scientific (and non-scientific) development environments** available
  - spyder: MATLAB-like environment
  - Jupyter/IPython notebooks: environment for interactive and exploratory Python
  - Visual Studio Code: new Python lightweight environment
  - PyCharm: Python environment for developers
- **Great performance due to** close integration with time-tested and highly **optimized codes written in C/C++ and Fortran**
- Readily available and **suitable for use** on high-performance **computing clusters**
- **No license costs**, no unnecessary use of research budget

Python for science, where to begin?

Why are there two versions of Python?

- At one time, there were a lot of modules not compatibles with Python 3
- Python 2 is still **actively supported**. For example, many Linux distributions and Macs are still using internally 2.x as default

It's 2018. Why to choose Python 3?

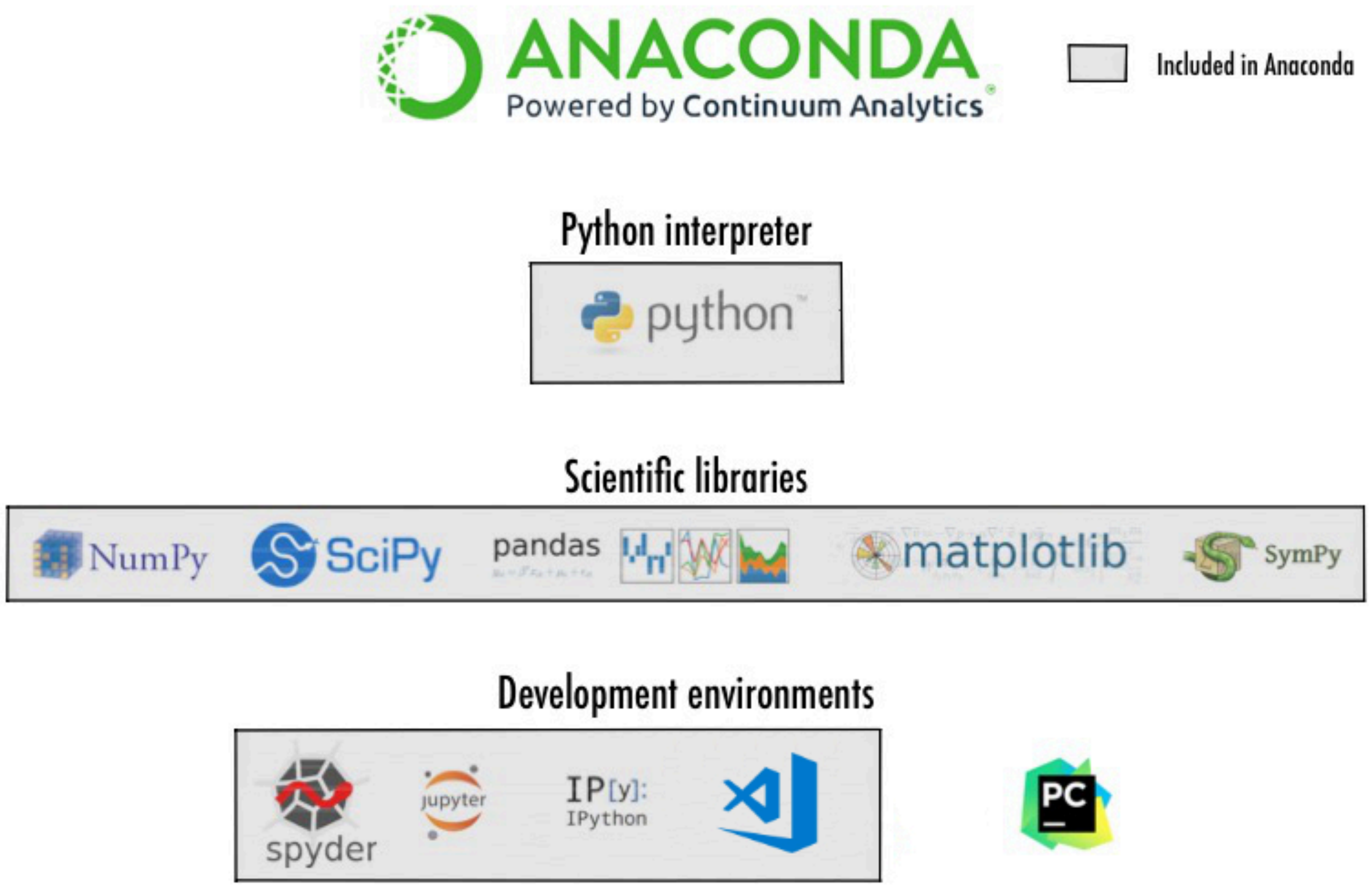
- **Differences** between Python 2 and 3 **are relatively minor** for *beginner programmers*
- Python 3 brings **many improvements over Python 2**
- Python 2 end-of-life will be on **January 1st, 2020**

Scientific-oriented Python Distributions

Provide a **Python interpreter** with commonly used **scientific libraries** in science like NumPy, SciPy, Pandas, matplotlib, etc. already installed. In the past, it was usually painful to build some of these packages.

Also, include **development environments** with advanced editing, debugging and introspection features.

- **Anaconda**
  - Cross-platform
  - Supports Python 2 and 3
  - **Most widely adopted**
- Canopy
  - Cross-platform
  - Supports Python 2 and 3
  - Includes a built-in IDE
- WinPython
  - Windows-only platform
  - Only supports Python 3
- Python(x,y)
  - Windows-only platform
  - Only supports Python 2
  - Not actively developed



ANACONDA
Powered by Continuum Analytics

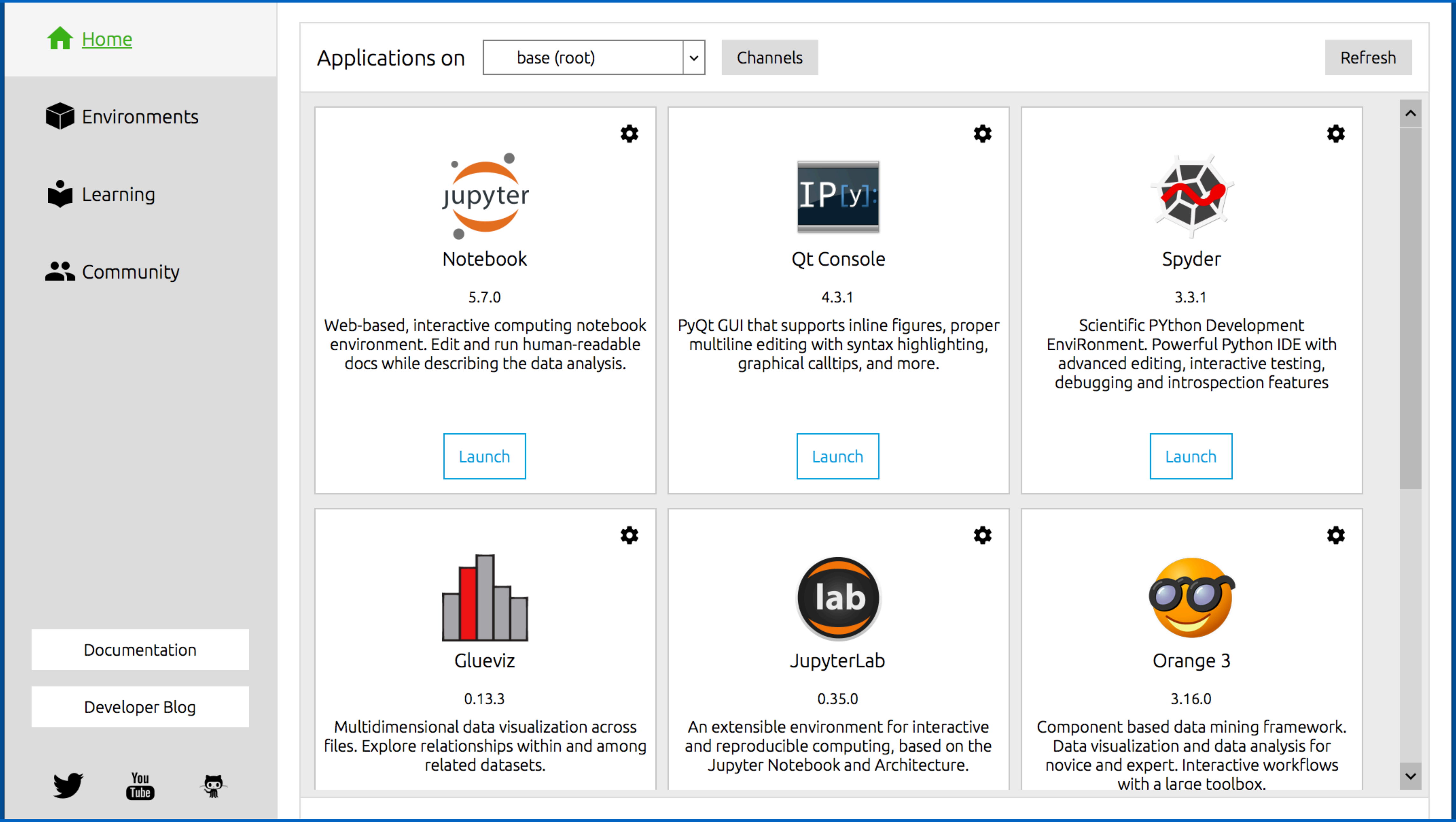
□ Included in Anaconda

Python interpreter

Scientific libraries

Development environments

Anaconda Navigator



Notebook

5.7.0

Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

Qt Console

4.3.1

PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.

Spyder

3.3.1

Scientific PYthon Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features

Glueviz

0.13.3

Multidimensional data visualization across files. Explore relationships within and among related datasets.

JupyterLab

0.35.0

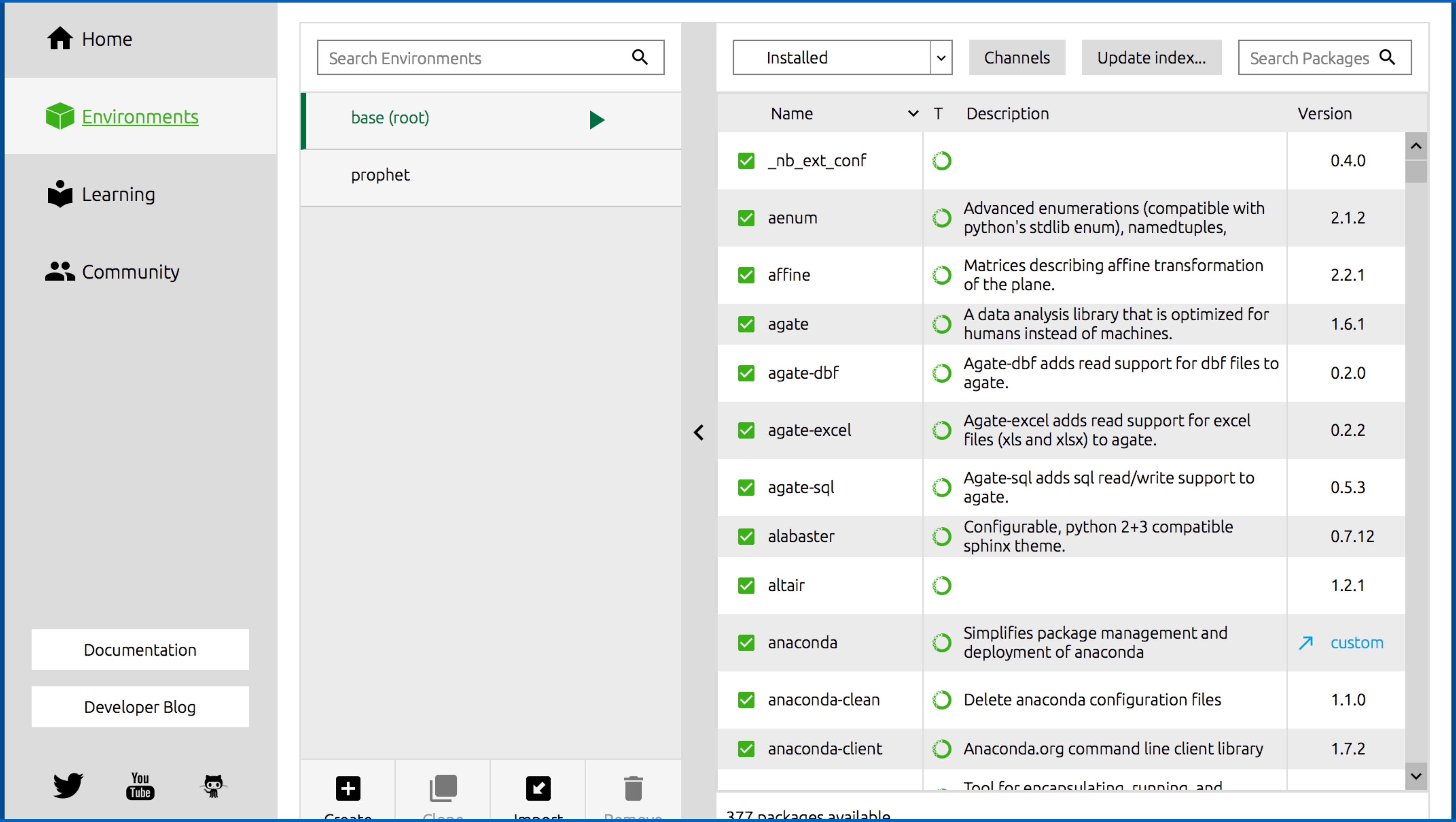
An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.

Orange 3

3.16.0

Component based data mining framework. Data visualization and data analysis for novice and expert. Interactive workflows with a large toolbox.

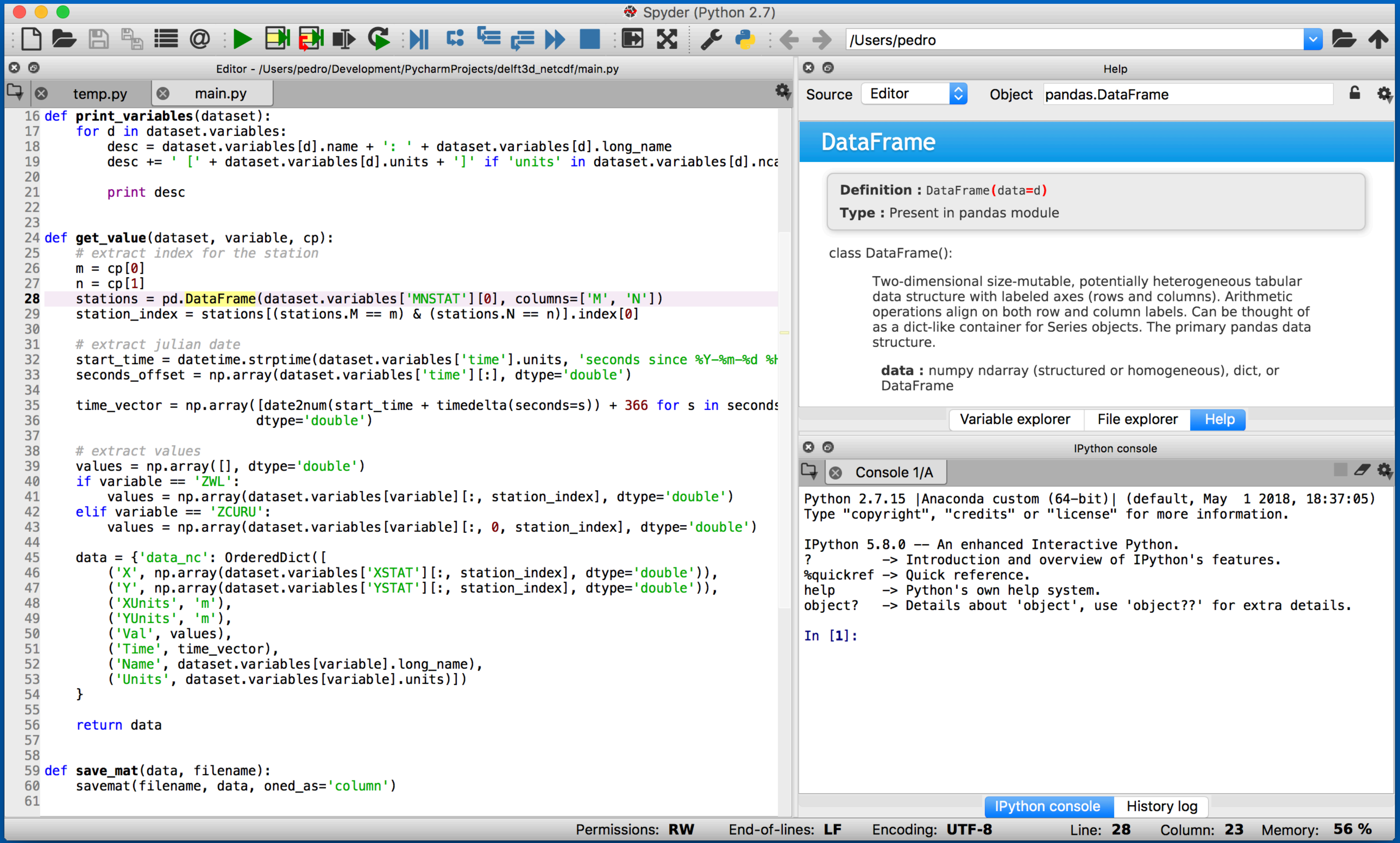
Anaconda Navigator: installing new packages



	Name	T	Description	Version
☑	_nb_ext_conf	○		0.4.0
☑	aenum	○	Advanced enumerations (compatible with python's stdlib enum), namedtuples,	2.1.2
☑	affine	○	Matrices describing affine transformation of the plane.	2.2.1
☑	agate	○	A data analysis library that is optimized for humans instead of machines.	1.6.1
☑	agate-dbf	○	Agate-dbf adds read support for dbf files to agate.	0.2.0
☑	agate-excel	○	Agate-excel adds read support for excel files (xls and xlsx) to agate.	0.2.2
☑	agate-sql	○	Agate-sql adds sql read/write support to agate.	0.5.3
☑	alabaster	○	Configurable, python 2+3 compatible sphinx theme.	0.7.12
☑	altair	○		1.2.1
☑	anaconda	○	Simplifies package management and deployment of anaconda	↗ custom
☑	anaconda-clean	○	Delete anaconda configuration files	1.1.0
☑	anaconda-client	○	Anaconda.org command line client library	1.7.2
			Tool for encapsulating, running, and	

377 packages available

Spyder



```python
def print_variables(dataset):
    for d in dataset.variables:
        desc = dataset.variables[d].name + ': ' + dataset.variables[d].long_name
        desc += ' [' + dataset.variables[d].units + ']' if 'units' in dataset.variables[d].nca

        print desc


def get_value(dataset, variable, cp):
    # extract index for the station
    m = cp[0]
    n = cp[1]
    stations = pd.DataFrame(dataset.variables['MNSTAT'][0], columns=['M', 'N'])
    station_index = stations[(stations.M == m) & (stations.N == n)].index[0]

    # extract julian date
    start_time = datetime.strptime(dataset.variables['time'].units, 'seconds since %Y-%m-%d %H
    seconds_offset = np.array(dataset.variables['time'][:], dtype='double')

    time_vector = np.array([date2num(start_time + timedelta(seconds=s)) + 366 for s in seconds
                           dtype='double')

    # extract values
    values = np.array([], dtype='double')
    if variable == 'ZWL':
        values = np.array(dataset.variables[variable][:, station_index], dtype='double')
    elif variable == 'ZCURU':
        values = np.array(dataset.variables[variable][:, 0, station_index], dtype='double')

    data = {'data_nc': OrderedDict([
        ('X', np.array(dataset.variables['XSTAT'][:, station_index], dtype='double')),
        ('Y', np.array(dataset.variables['YSTAT'][:, station_index], dtype='double')),
        ('XUnits', 'm'),
        ('YUnits', 'm'),
        ('Val', values),
        ('Time', time_vector),
        ('Name', dataset.variables[variable].long_name),
        ('Units', dataset.variables[variable].units)])
    }

    return data


def save_mat(data, filename):
    savemat(filename, data, oned_as='column')

```

DataFrame

Definition : `DataFrame(data=d)`

Type : Present in pandas module

class DataFrame():

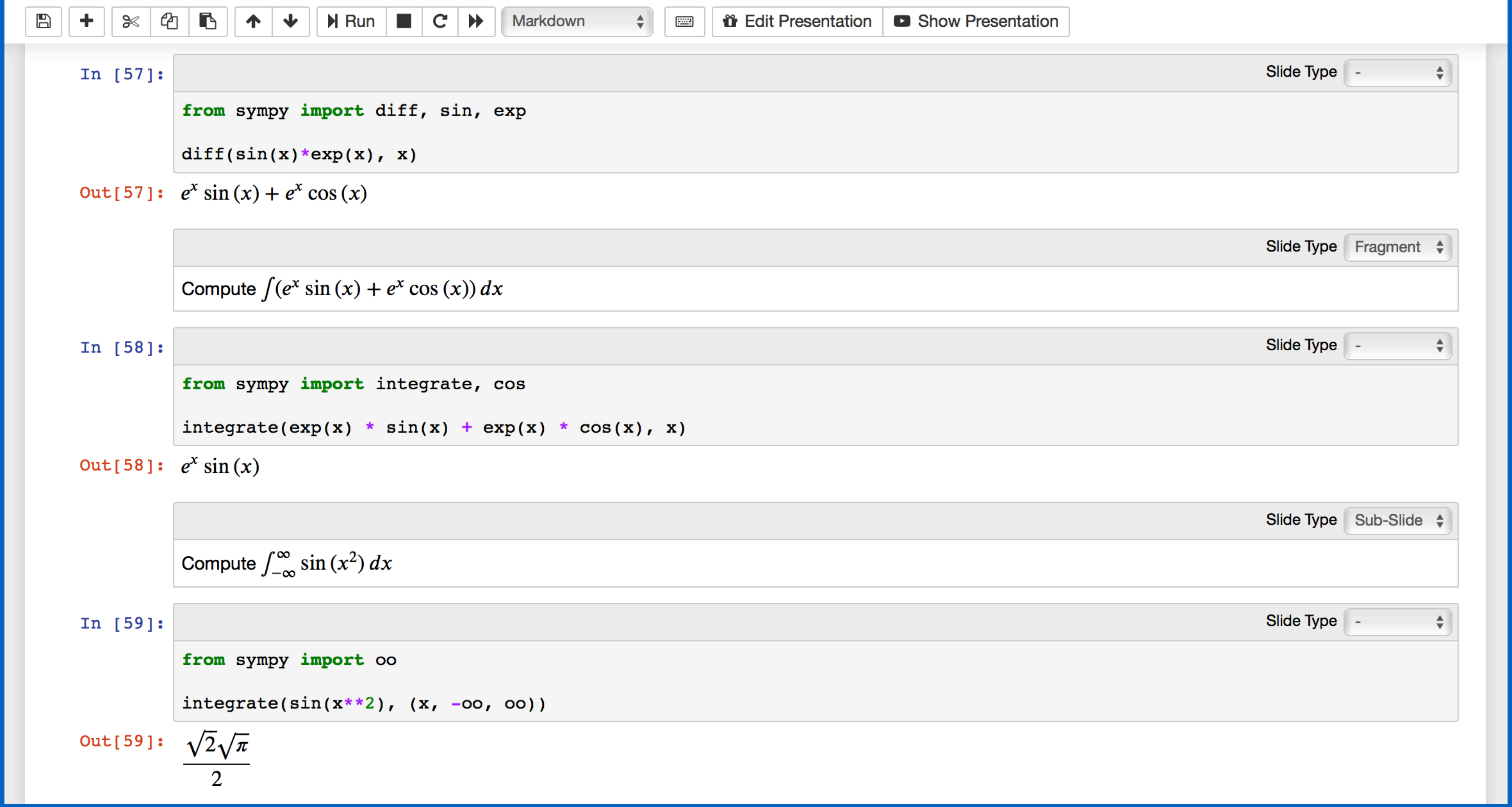
Two-dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns). Arithmetic operations align on both row and column labels. Can be thought of as a dict-like container for Series objects. The primary pandas data structure.

data : numpy ndarray (structured or homogeneous), dict, or DataFrame

```
Python 2.7.15 |Anaconda custom (64-bit)| (default, May  1 2018, 18:37:05)
Type "copyright", "credits" or "license" for more information.

IPython 5.8.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]:
```

Permissions: **RW**   End-of-lines: **LF**   Encoding: **UTF-8**   Line: **28**   Column: **23**   Memory: **56 %**

Jupyter notebooks



Slide Type: -

```
In [57]: from sympy import diff, sin, exp

         diff(sin(x)*exp(x), x)
```

Out[57]: $e^{x} \sin{\left(x \right)} + e^{x} \cos{\left(x \right)}$

Slide Type: Fragment

Compute $\int (e^{x} \sin{\left(x \right)} + e^{x} \cos{\left(x \right)})\, dx$

Slide Type: -

```
In [58]: from sympy import integrate, cos

         integrate(exp(x) * sin(x) + exp(x) * cos(x), x)
```

Out[58]: $e^{x} \sin{\left(x \right)}$

Slide Type: Sub-Slide

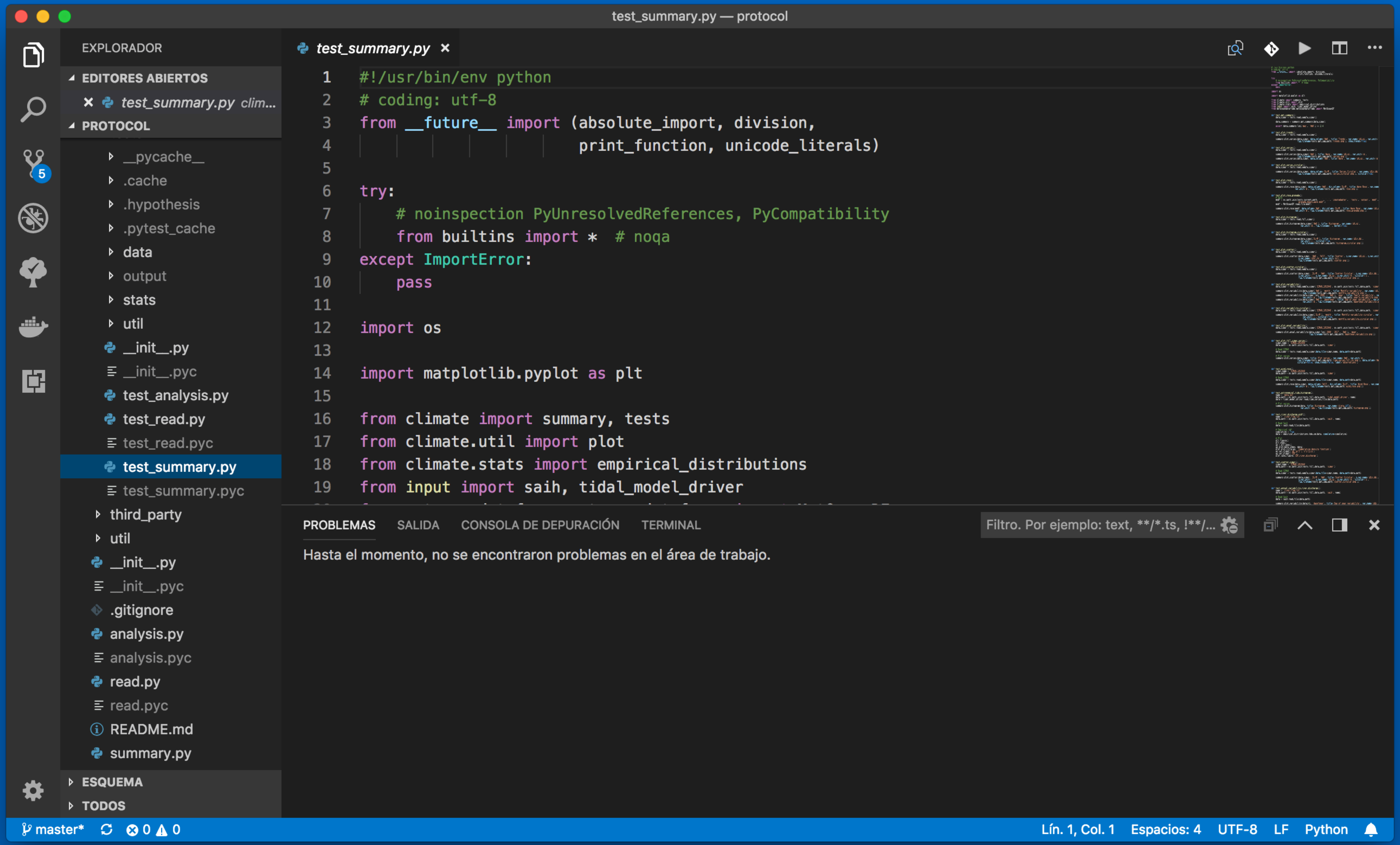
Compute $\int_{-\infty}^{\infty} \sin{\left(x^{2} \right)}\, dx$

Slide Type: -

```
In [59]: from sympy import oo

         integrate(sin(x**2), (x, -oo, oo))
```

Out[59]: $\frac{\sqrt{2} \sqrt{\pi}}{2}$

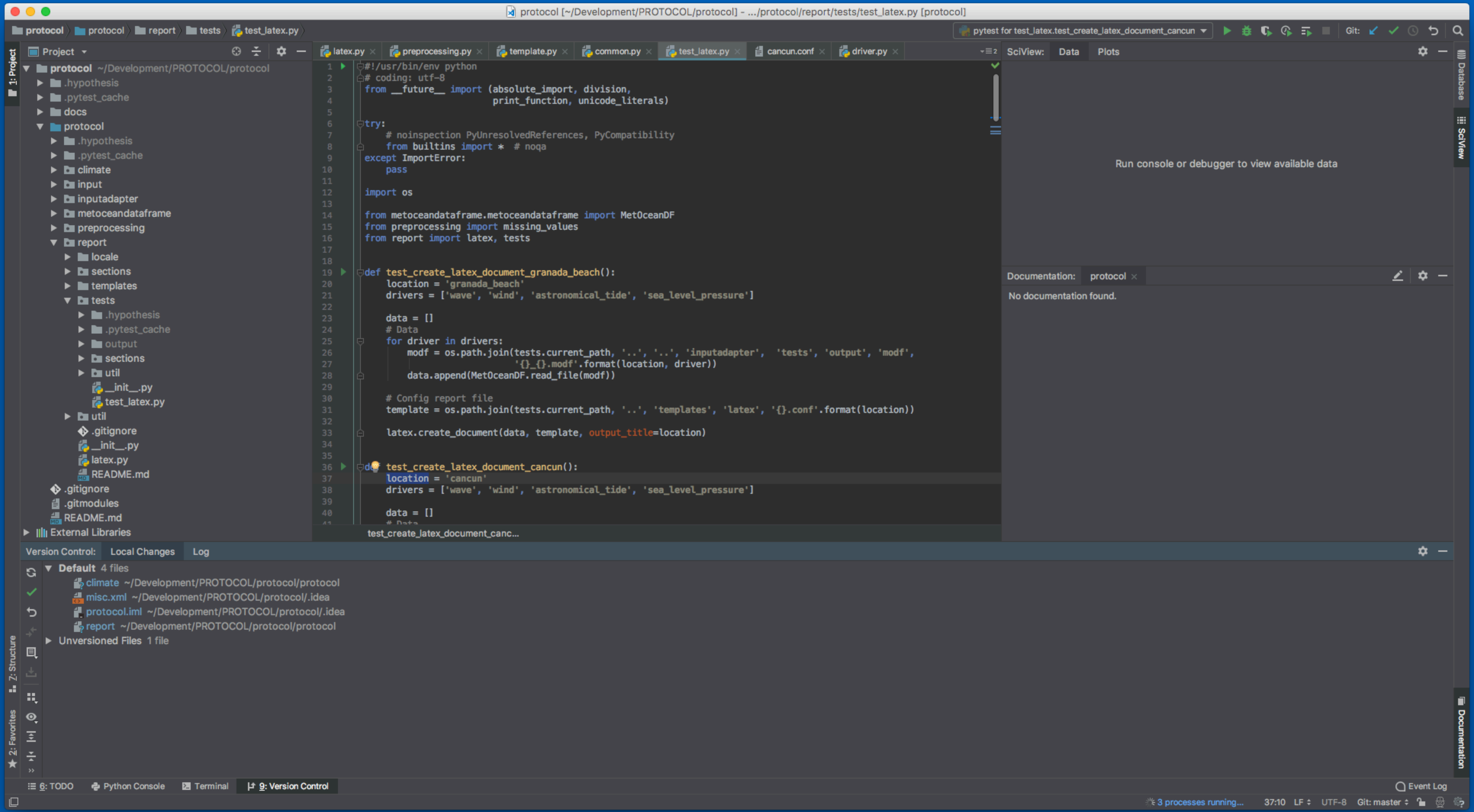
Visual Studio Code



```python
#!/usr/bin/env python
# coding: utf-8
from __future__ import (absolute_import, division,
                        print_function, unicode_literals)

try:
    # noinspection PyUnresolvedReferences, PyCompatibility
    from builtins import *  # noqa
except ImportError:
    pass

import os

import matplotlib.pyplot as plt

from climate import summary, tests
from climate.util import plot
from climate.stats import empirical_distributions
from input import saih, tidal_model_driver
```

Hasta el momento, no se encontraron problemas en el área de trabajo.

PyCharm (need to be installed separately from Anaconda)



```
#!/usr/bin/env python
# coding: utf-8
from __future__ import (absolute_import, division,
                        print_function, unicode_literals)

try:
    # noinspection PyUnresolvedReferences, PyCompatibility
    from builtins import *  # noqa
except ImportError:
    pass

import os

from metoceandataframe.metoceandataframe import MetOceanDF
from preprocessing import missing_values
from report import latex, tests


def test_create_latex_document_granada_beach():
    location = 'granada_beach'
    drivers = ['wave', 'wind', 'astronomical_tide', 'sea_level_pressure']

    data = []
    # Data
    for driver in drivers:
        modf = os.path.join(tests.current_path, '..', '..', 'inputadapter',  'tests', 'output', 'modf',
                            '{}_{}.modf'.format(location, driver))
        data.append(MetOceanDF.read_file(modf))

    # Config report file
    template = os.path.join(tests.current_path, '..', 'templates', 'latex', '{}.conf'.format(location))

    latex.create_document(data, template, output_title=location)


def test_create_latex_document_cancun():
    location = 'cancun'
    drivers = ['wave', 'wind', 'astronomical_tide', 'sea_level_pressure']

    data = []
```

Run console or debugger to view available data

No documentation found.

Default 4 files
climate ~/Development/PROTOCOL/protocol/protocol
misc.xml ~/Development/PROTOCOL/protocol/.idea
protocol.iml ~/Development/PROTOCOL/protocol/.idea
report ~/Development/PROTOCOL/protocol/protocol
Unversioned Files 1 file

Editor	Learning curve	Users	Benefits
Spyder	pretty short	Matlab and R background	mature, many features
Jupyter	smooth	teachers	interactive
Visual Studio Code	moderate	scientifics / developers	code quality
PyCharm	steep	developers	professional code

Where to look for help?

- **Official documentation**: http://www.scipy.org/docs.html
- Usually included in development environments as **contextual help**:
    - *Spyder*: Ctrl+I (Windows) or Cmd+I (Mac)
    - *Visual Studio Code*: Ctrl+Space (Windows/Mac)
    - *PyCharm*: F1 (Windows/Mac)
- **Be careful about code you get on the internet!**

Python language

Using Python as a Calculator

```
2 + 2
> 4

50 - 5*6
> 20

(50 - 5*6) / 4
> 5.0

# division always returns a floating point number
8 / 5
> 1.6
```

Strings

```
prefix = 'Py'
word = prefix + 'thon'

# character in position 0
print(word[0])
> P

# characters from position 0 (included) to 4 (excluded)
print(word[0:4])
> Pyth
```

- **0-based indexing**
- **half-open range indexing**: [a, b)
- **print** statement to get outputs
- **line comments**

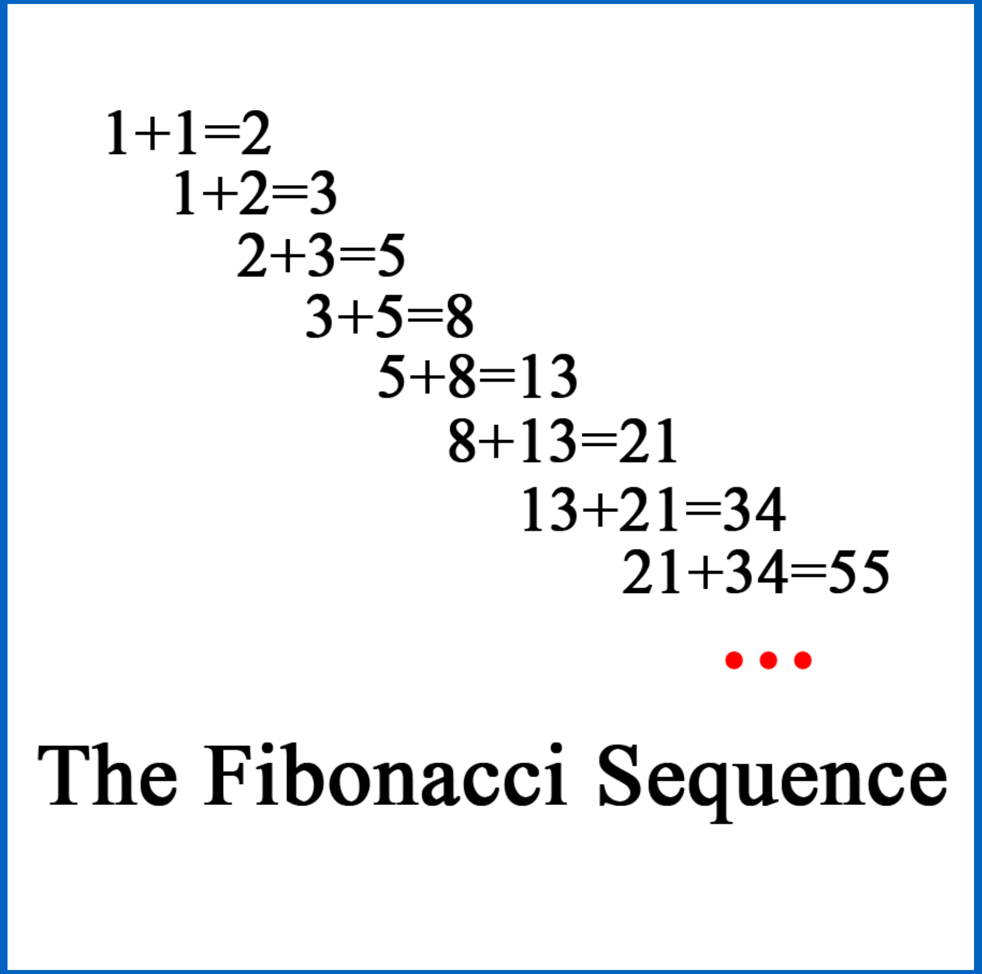
Lists

```
# empty list
squares = []

# lists might contain items of different types
squares = ['cat', 4, 3.2]

# negative indices mean count backwards from end of sequence
print(squares[-1])
> 3.2

# list concatenation
squares = squares + [81, 'dog']

# list functions
squares.remove(3.2)  # remove the first ocurrence
squares.append('horse')  # concatenation: same as +

print(squares)
> ['cat', 4, 81, 'dog', 'horse']
```

```
a = ['a', 'b', 'c']
n = [1, 2, 3]

# it is possible to nest lists
# (create lists containing other lists)
x = [a, n]

print(x)
> [['a', 'b', 'c'], [1, 2, 3]]

print(x[0])
> ['a', 'b', 'c']

print(x[0][1])
> b
```

Simple code: Fibonacci series

```
a, b = 0, 1
while a < 10:
    print(a),
    # the sum of two elements defines the next
    c = a + b
    a = b
    b = c

> 0 1 1 2 3 5 8
```



1+1=2
1+2=3
2+3=5
3+5=8
5+8=13
8+13=21
13+21=34
21+34=55
...

The Fibonacci Sequence

- **indentation level** of statements is significant
- **multiple assignment**

if Statements

```
x = -4

if x < 0:
    x = 0
    print('Negative changed to zero')
elif x == 0:
    print('Zero')
elif x == 1:
    print('Single')
else:
    print('More')
```

```
> Negative changed to zero
```

for Statements

```
words = ['cat', 'window', 'defenestrate']

for w in words:
    # len returns the number of items of an object.
    print(w, len(w))
```

```
> cat 3
> window 6
> defenestrate 12
```

range(`stop`): Built-in function to create lists containing arithmetic progressions.

```
print range(10)
> [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

print range(0, 10, 3)
> [0, 3, 6, 9]

print range(0, -10, -1)
> [0, -1, -2, -3, -4, -5, -6, -7, -8, -9]
```

```
total = 0
for i in range(4):  # range(4) = [0, 1, 2, 3]
    total = total + 1  # i is not used
print total
```

```
> 4
```

- Please **avoid Matlab-like for** statements with **range**:

```
for w in range(len(words)):
    print words[w], len(words[w])
```

Functions

```
def fibonacci(n):
    """Build a Fibonacci series up to n.

    Args:
        n: upper limit.

    Returns:
        A list with a Fibonacci series up to n.
    """
    f = []  # always initialize the returned value!

    a, b = 0, 1
    while a < n:
        f.append(a)
        # the sum of two elements defines the next
        c = a + b
        a = b
        b = c

    return f

# now call the function we just defined:
print fibonacci(1000)

> [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987]
```

Functions: documentation strings (docstrings)

- Python documentation strings (docstrings) provide a convenient **way of associating documentation with Python functions** and modules.
- Docstrings can be written following **several styles**. We use Google Python Style Guide.
- An object's docsting is defined by including a **string constant as the first statement in the function's definition**.
- Unlike conventional source code comments **the docstring should describe what the function does, not how**.
- **All functions should have a docstring**.
- This allows to inspect these comments at run time, for instance as an **interactive help system**, or **export them as HTML, LaTeX, PDF** or other formats.

Functions: default argument values

```python
def fibonacci(n, s=0):
    """Build a Fibonacci series up to n.

    Args:
        n: upper limit.
        s: lower limit. Default 0.

    Returns:
        A list with a Fibonacci series up to n.
    """
    f = []  # always initialize the returned value!

    a, b = 0, 1
    while a < n:
        if a >= s:  # lower limit
            f.append(a)
        # the sum of two elements defines the next
        c = a + b
        a = b
        b = c

    return f
```

```
print fibonacci(1000, 15)
> [21, 34, 55, 89, 144, 233, 377, 610, 987]

print fibonacci(1000, 0)
> [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987]

print fibonacci(1000)
> [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987]
```

Functions: keyword arguments

```
print fibonacci(1000, 15)  # positional arguments
> [21, 34, 55, 89, 144, 233, 377, 610, 987]

print fibonacci(s=15, n=1000)  # keyword arguments
> [21, 34, 55, 89, 144, 233, 377, 610, 987]
```

Functions: importing external functions

```
import functions  # without .py extension

print functions.fibonacci(3)

> [0, 1, 1, 2]
```

```
from functions import fibonacci

print fibonacci(3)

> [0, 1, 1, 2]
```

```
import functions as f  # alias

print f.fibonacci(3)

> [0, 1, 1, 2]
```

Recommendation

The best way to import libraries is included in their official help

Some examples:

```
import math
import numpy as np
from scipy import linalg, optimize
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
import sympy
```

Code Style

- Style Guide for Python Code: **PEP8**.
- Use **only English (ASCII) characters for variables, functions and files**.
- Name your **variables, functions and files** consistently: the convention is to use **lower*case*with_underscores**.
- We all use **single-quoted strings** to be consistent. Nevertheless, single-quoted strings and double-quoted strings are the same. PEP does not make a recommendation for this, **except for function documentation** where tripe-quote strings should be used.

PEP8 exceptions

- Long lines

It is very conservative and requires limiting lines to 79 characters. We use **all lines to a maximum of 119 characters**. This is the default behaviour in *PyCharm*.

- Disable checks in one line

Skip validation in one lines by adding following comment:
```
#  nopep8
```

`datetime` data type

The `datetime` module supplies classes for **manipulating dates and times**. **Avoid converting dates or times** to int (`datenum` or similar).

```
from datetime import datetime, date, time

# Using datetime.combine()
d = date(2005, 7, 14)
t = time(12, 30)
dt1 = datetime.combine(d, t)

print dt1
> 2005-07-14 12:30:00

print dt1.year
> 2005
```

timedelta([days[, seconds[, microseconds[, milliseconds[, minutes[, hours[, weeks]]]]]]])

All arguments are optional and default to 0. Arguments may be ints, longs, or floats, and may be positive or negative.

```
from datetime import timedelta

dt2 = dt1 + timedelta(hours=5)

print dt2
> 2005-07-14 17:30:00
```

`boolean` data type

`boolean` values are the **two constant objects `False` and `True`**. In numeric contexts (for example when used as the argument to an arithmetic operator), they behave like the integers 0 and 1, respectively.

Nevertheless, other values can also be considered false or true:

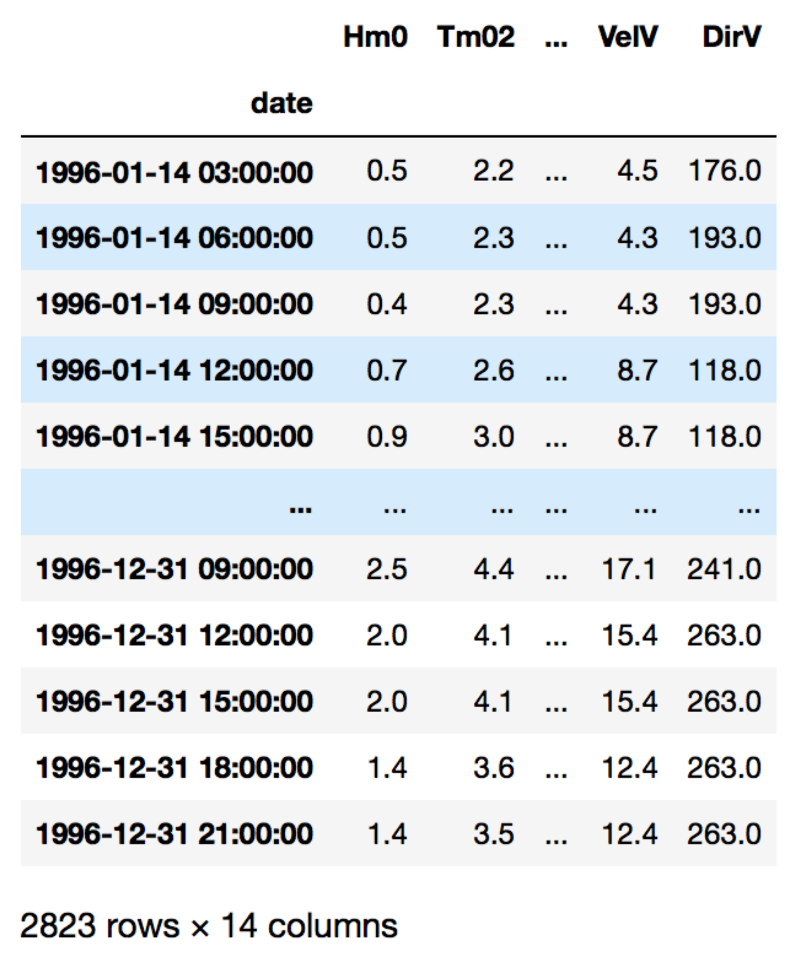
- the following values are considered false: `0`, `''`, `[]`, `()`, `{}`, None
- all other values are considered true, so objects of many types are always true

Scientific libraries

Pandas

- fast and efficient **Series (1-dimensional) and DataFrame (2-dimensional) heterogeneous objects** for data manipulation with integrated indexing
- tools for **reading and writing data from different formats**: CSV and text files, Microsoft Excel, SQL databases, HDF5...
- intelligent **label-based slicing**
- **time series-functionality**
- integrated **handling of missing data**

```python
import pandas as pd

simar = pd.read_table('WANA_2006008_Algeciras.txt',
                      delim_whitespace=True,
                      parse_dates= {'date' : [0,1,2,3]},
                      index_col='date', skiprows=70)

print(simar)
```



	Hm0	Tm02	...	VelV	DirV
date					
1996-01-14 03:00:00	0.5	2.2	...	4.5	176.0
1996-01-14 06:00:00	0.5	2.3	...	4.3	193.0
1996-01-14 09:00:00	0.4	2.3	...	4.3	193.0
1996-01-14 12:00:00	0.7	2.6	...	8.7	118.0
1996-01-14 15:00:00	0.9	3.0	...	8.7	118.0
...	...	...	...	...	...
1996-12-31 09:00:00	2.5	4.4	...	17.1	241.0
1996-12-31 12:00:00	2.0	4.1	...	15.4	263.0
1996-12-31 15:00:00	2.0	4.1	...	15.4	263.0
1996-12-31 18:00:00	1.4	3.6	...	12.4	263.0
1996-12-31 21:00:00	1.4	3.5	...	12.4	263.0

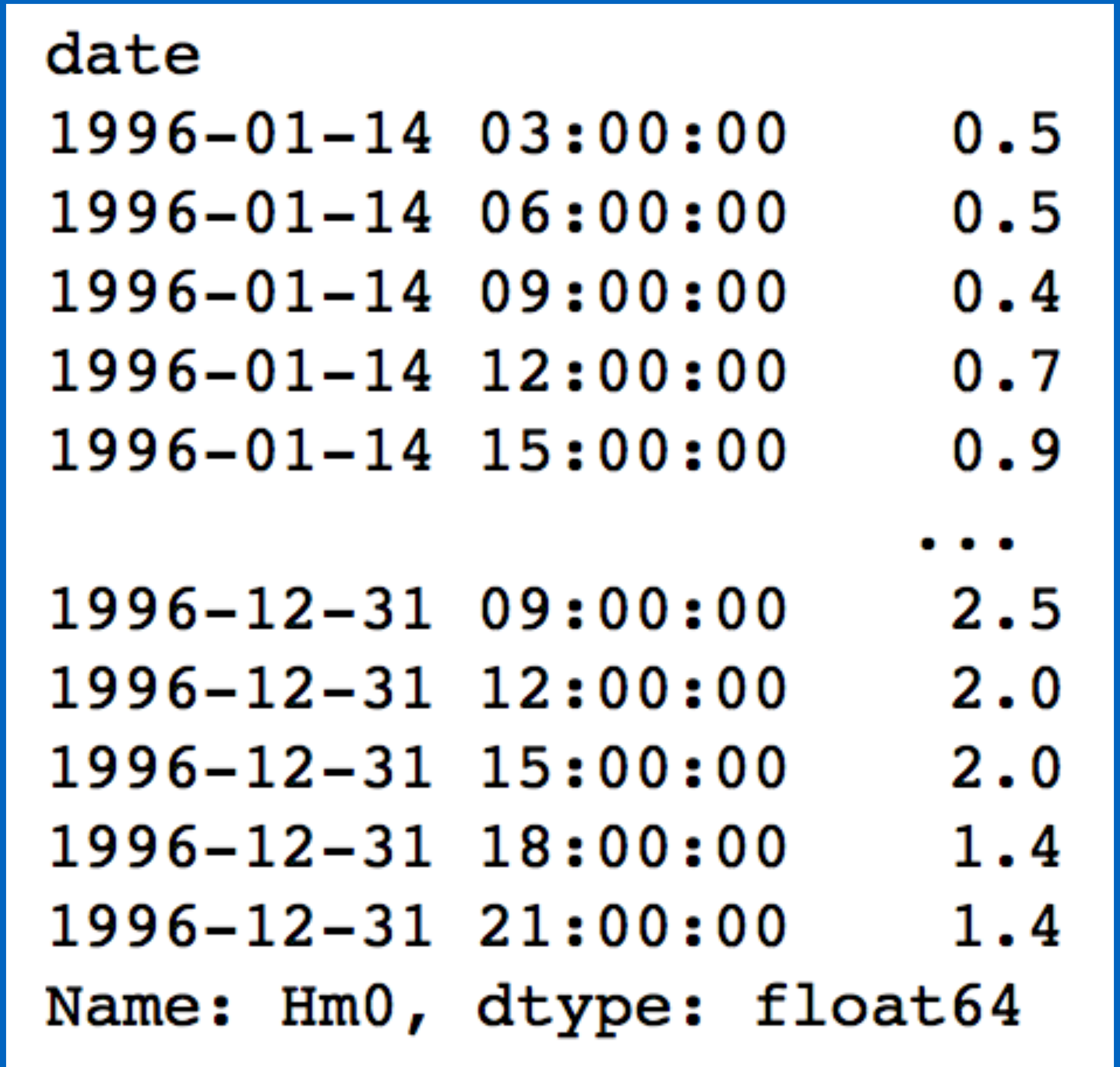
2823 rows × 14 columns

read_table(. . .)

Read general delimited file into DataFrame.

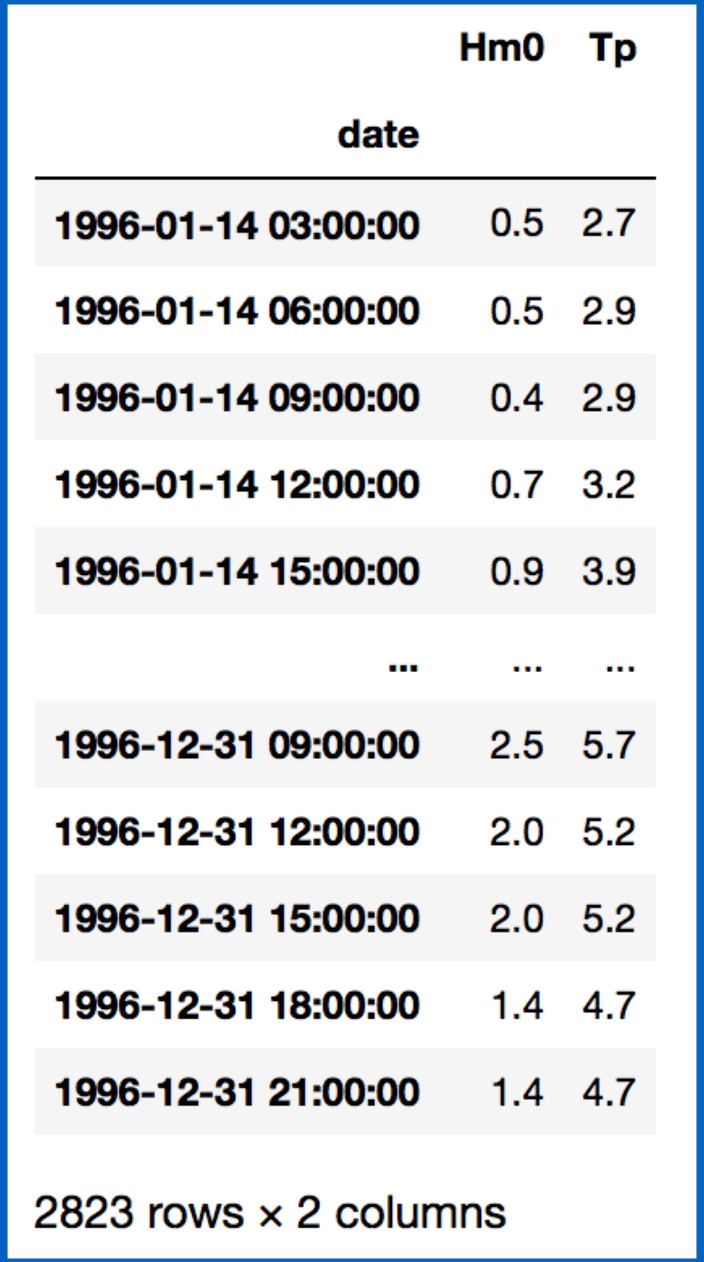
- `delim_whitespace`: boolean, default False. Specifies whether or not whitespace (e.g. ' ' or ' ') will be used as the sep.
- `parse_dates`: boolean or list of ints or names or list of lists or dict, default False boolean. dict, e.g. {'foo' : [1, 3]} -> parse columns 1, 3 as date and call result 'foo'
- `index_col`: int or sequence or False, default None. Column to use as the row labels of the DataFrame.
- `skiprows`: list-like or integer, default None. Line numbers to skip (0-indexed) or number of lines to skip (int) at the start of the file
- `header`: int or list of ints, default 'infer'. Row number(s) to use as the column names, and the start of the data. Default behavior is as if set to 0 if no names passed, otherwise None.

```
out = simar['Hm0']  # selecting a single column

print(out)
```



```
date
1996-01-14 03:00:00     0.5
1996-01-14 06:00:00     0.5
1996-01-14 09:00:00     0.4
1996-01-14 12:00:00     0.7
1996-01-14 15:00:00     0.9
                       ...
1996-12-31 09:00:00     2.5
1996-12-31 12:00:00     2.0
1996-12-31 15:00:00     2.0
1996-12-31 18:00:00     1.4
1996-12-31 21:00:00     1.4
Name: Hm0, dtype: float64
```

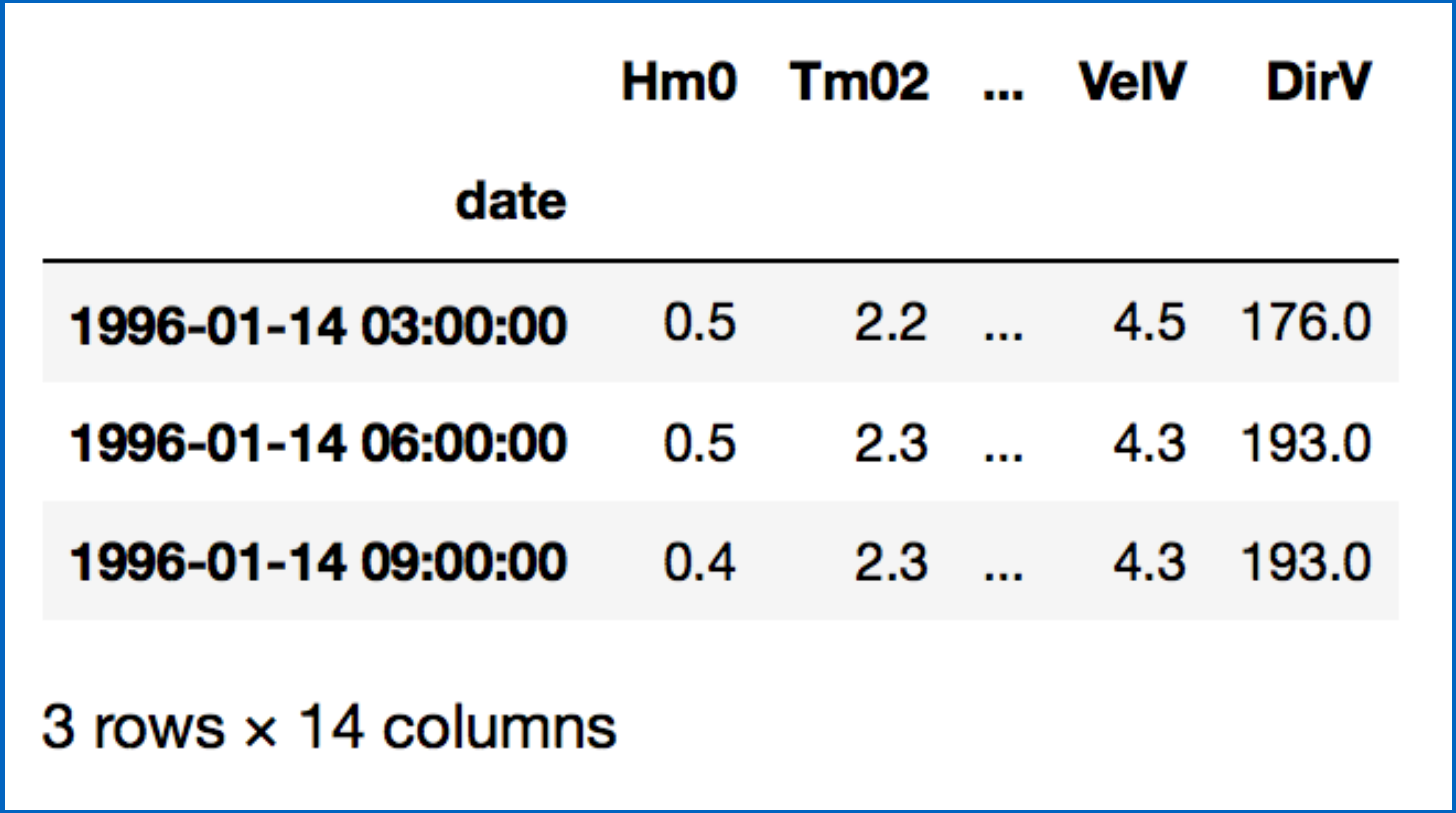
```python
out = simar[['Hm0', 'Tp']]  # selecting several columns using a list

print(out)
```



	Hm0	Tp
date		
1996-01-14 03:00:00	0.5	2.7
1996-01-14 06:00:00	0.5	2.9
1996-01-14 09:00:00	0.4	2.9
1996-01-14 12:00:00	0.7	3.2
1996-01-14 15:00:00	0.9	3.9
...	...	...
1996-12-31 09:00:00	2.5	5.7
1996-12-31 12:00:00	2.0	5.2
1996-12-31 15:00:00	2.0	5.2
1996-12-31 18:00:00	1.4	4.7
1996-12-31 21:00:00	1.4	4.7

2823 rows × 2 columns

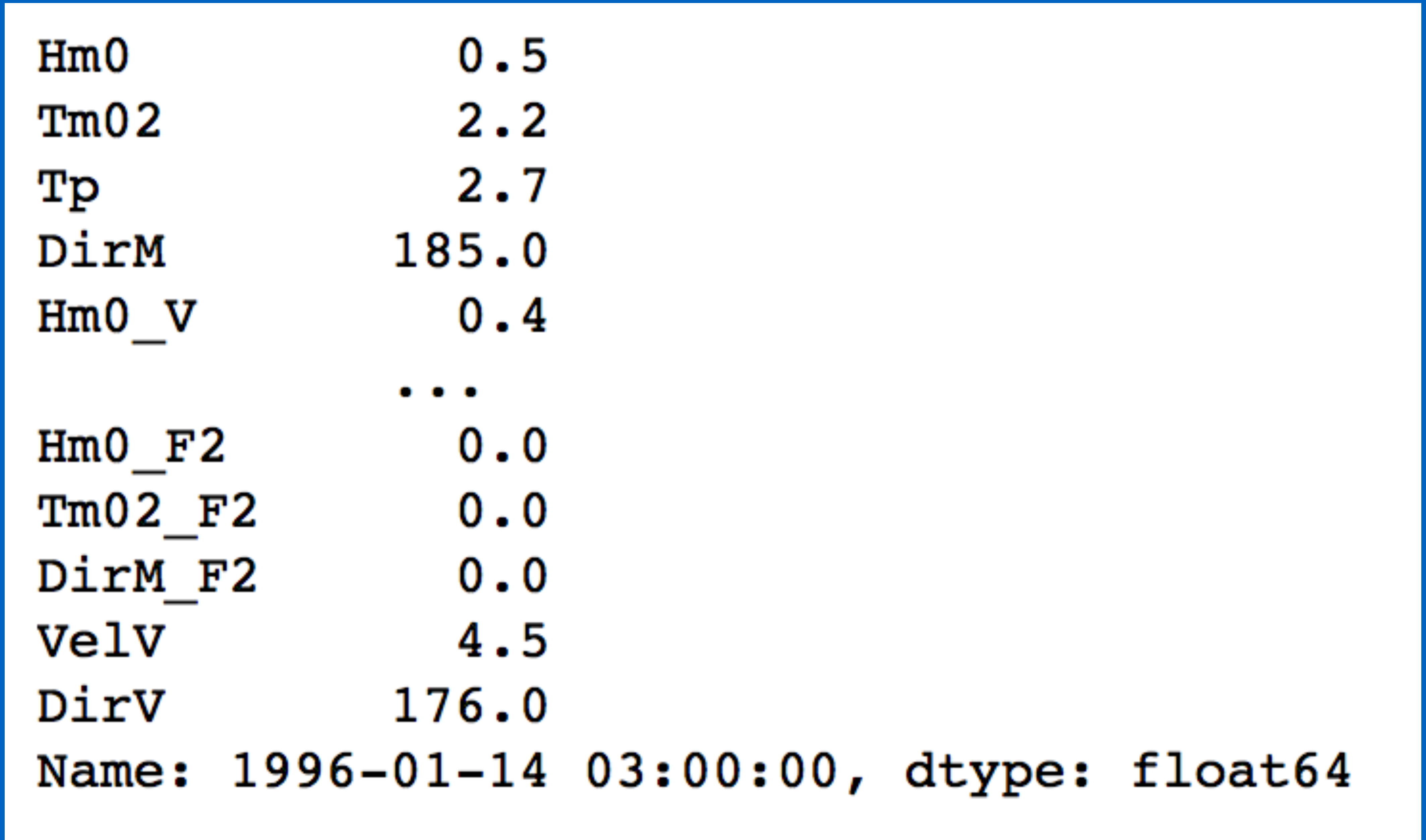
```
out = simar.iloc[0:3]  # selecting rows by position

print(out)
```



	Hm0	Tm02	...	VelV	DirV
date					
1996-01-14 03:00:00	0.5	2.2	...	4.5	176.0
1996-01-14 06:00:00	0.5	2.3	...	4.3	193.0
1996-01-14 09:00:00	0.4	2.3	...	4.3	193.0

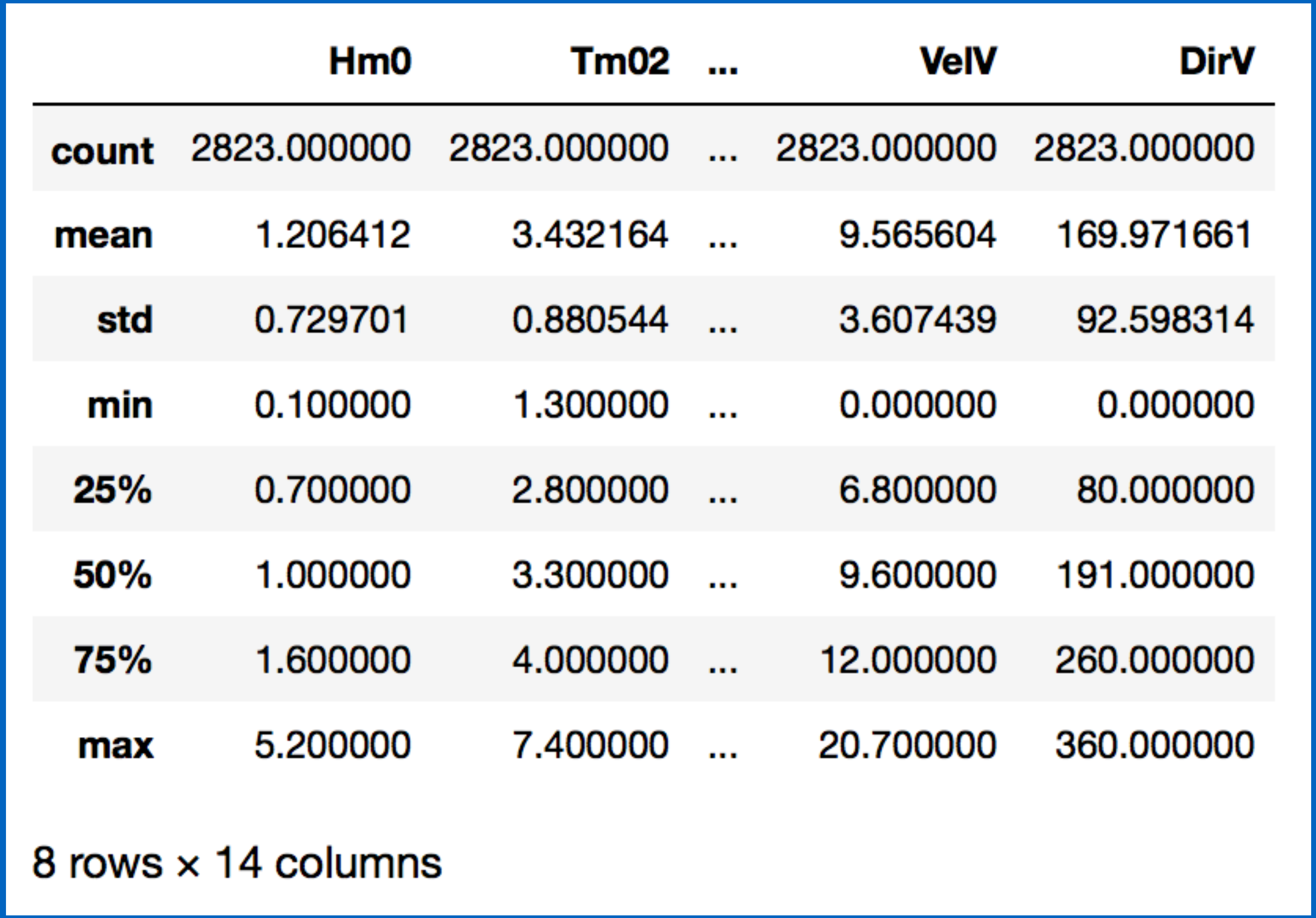
3 rows × 14 columns

```
out = simar.loc['1996-01-14 03:00:00']  # selecting rows by label

print(out)
```



```
Hm0             0.5
Tm02            2.2
Tp              2.7
DirM          185.0
Hm0_V           0.4
           ...
Hm0_F2          0.0
Tm02_F2         0.0
DirM_F2         0.0
VelV            4.5
DirV          176.0
Name: 1996-01-14 03:00:00, dtype: float64
```

```
out = simar.describe()

print(out)
```



	Hm0	Tm02	...	VelV	DirV
count	2823.000000	2823.000000	...	2823.000000	2823.000000
mean	1.206412	3.432164	...	9.565604	169.971661
std	0.729701	0.880544	...	3.607439	92.598314
min	0.100000	1.300000	...	0.000000	0.000000
25%	0.700000	2.800000	...	6.800000	80.000000
50%	1.000000	3.300000	...	9.600000	191.000000
75%	1.600000	4.000000	...	12.000000	260.000000
max	5.200000	7.400000	...	20.700000	360.000000

8 rows × 14 columns

Vectorization

Arrays enable you to express batch operations on data without writing any for loops. This is usually called **vectorization**:

- vectorized code is more concise and easier to read
- fewer lines of code generally means fewer bugs
- the code more closely resembles standard mathematical notation

But:

sometimes it's difficult to move away from the **for-loop** school of thought

NumPy

- Array manipulation routines
- Datetime Support Functions
- Discrete Fourier Transform (numpy.fft)
- Financial functions
- Indexing routines
- Linear algebra (numpy.linalg)
- Logic functions
- **Mathematical functions**
- Random sampling (numpy.random)
- Set routines
- Sorting, searching, and counting
- Statistics

Mathematical functions

- Trigonometric functions

  ```
  sin(x)
  cos(x)
  tan(x)
  ```

- Sums, products, differences

  ```
  prod(a)
  sum(a)
  nanprod(a)
  diff(a)
  ```

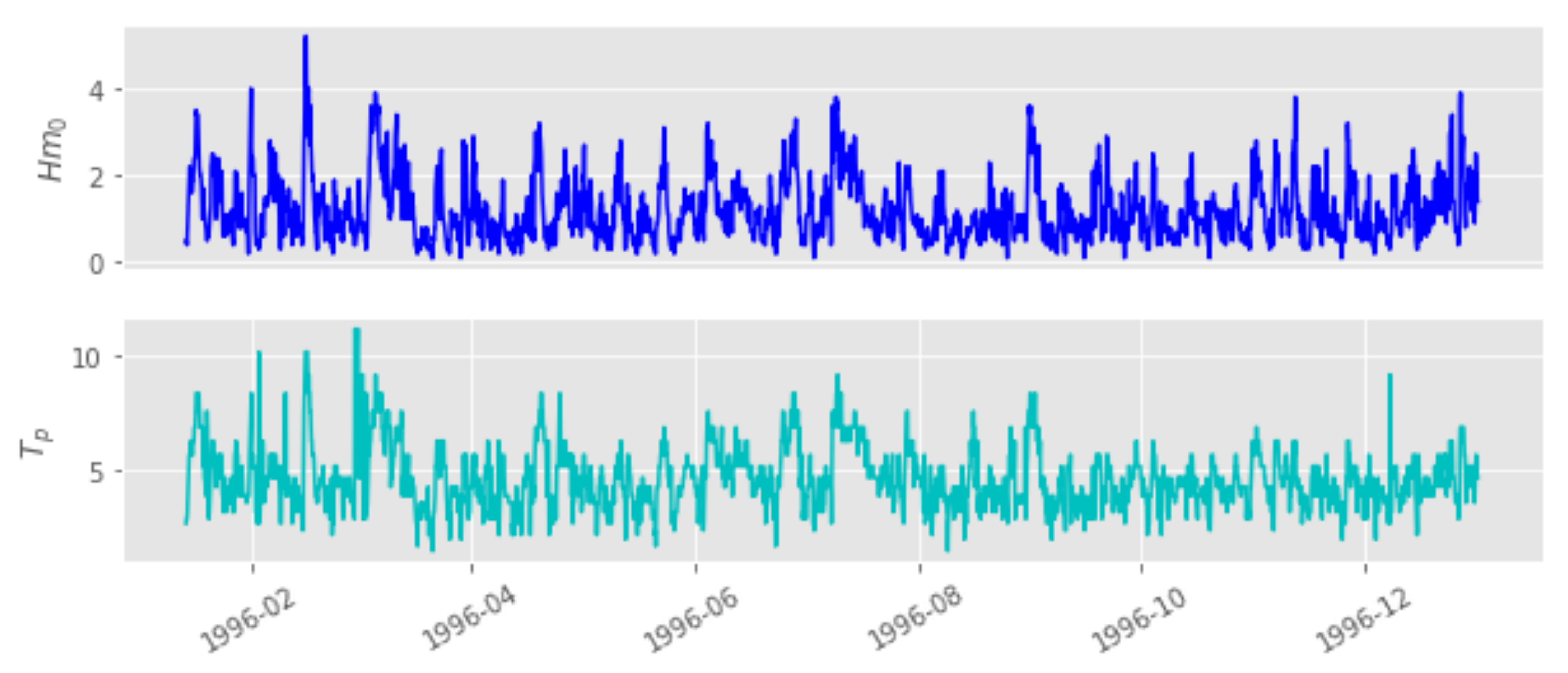
- Arithmetic operations
- Rounding
- Exponents and logarithms
- Hyperbolic functions

SciPy

- Clustering algorithms (`scipy.cluster`)
- Physical and mathematical constants (`scipy.constants`)
- Fast Fourier Transform routines (`scipy.fftpack`)
- Integration and ordinary differential equation solvers (`scipy.integrate`)
- Interpolation and smoothing splines (`scipy.interpolate`)
- Input and Output (`scipy.io`)
- Linear algebra (`scipy.linalg`)
- N-dimensional image processing (`scipy.ndimage`)
- Orthogonal distance regression (`scipy.odr`)
- Optimization and root-finding routines (`scipy.optimize`)
- Signal processing (`scipy.signal`)
- Sparse matrices and associated routines (`scipy.sparse`)
- Spatial data structures and algorithms (`scipy.spatial`)
- Special functions (`scipy.special`)
- Statistical distributions and functions (`scipy.stats`)
- C/C++ integration (`scipy.weave`)

matplotlib

`matplotlib` is a library for making plots in Python. The main component of `matplotlib` is `pylab` which allow the user to create plots with code quite similar to MATLAB figure generating code. `matplotlib` has its origins in emulating the MATLAB® graphics commands.



Sympy

SymPy is a Python library for symbolic mathematics.

```python
from sympy import symbols, init_printing

init_printing()  # pretty printing

x, y = symbols('x y')
expr = x + 2*y

print(expr)
```

> $x + 2y$

Derivative of $sin(x)e^x$

```
from sympy import diff, sin, exp

out = diff(sin(x)*exp(x), x)

print(out)
```

> $e^x sin(x) + e^x cos(x)$

Compute $\int \left(e^{x} \sin{\left(x \right)} + e^{x} \cos{\left(x \right)}\right)\, dx$

```
from sympy import integrate, cos

out = integrate(exp(x) * sin(x) + exp(x) * cos(x), x)

print(out)
```

> $e^x sin(x)$

Bibliography

- Elegant SciPy: The Art of Scientific Python por Juan Nunez-Iglesias, Stéfan van der Walt y otros (2017). ISBN: 9781491922873.
- Python for Data Analysis (2nd Edition) por Wes McKinney (2017). ISBN: 1491957662.
- Pandas Cookbook: Recipes for Scientific Computing, Time Series Analysis and Data Visualization using Python por Theodore Petrou (2017). ISBN: 9781784393878.

MOOC (Online Courses)

- Python for Data Science (University of California)
- Introduction to Python for Data Science (Microsoft)
- Intro to Python for Data Science (Datacamp)
- MOOC aggregator