Data Science in Python

Zime, Songbian PhD

September 23, 2023

Introduction

Data Science is the study of the generalizable extraction of knowledge from data; knowledge used for answering (scientific) questions through gathering, storing, processing and analyzing large amounts of data. (Dhar, 2013)

What is Data Science?

Data Science is neither about a particular set of tools like NoSQL Databases, Hadoop etc., nor about storing Google-scale amount of data, it is simply about the scientific questions that can be answered with (big) data. (Leek, 2013)

Disciplines in Data Science

Data Science is also an umbrella term covering a number of different disciplines:

- Statistics and statistical inference
- Stochastics as a mathematical modeling approach
- Databases as methods/tools for data storage
- Parallel Computing as methods for scaling up using multiple machines
- Machine Learning/Data Mining for providing algorithms
- Social Network Analysis as an application area
- Human Computing Interfaces and Visual Analytics
- Engineering
- ...

Let Other People Talk...

Click here to watch the video

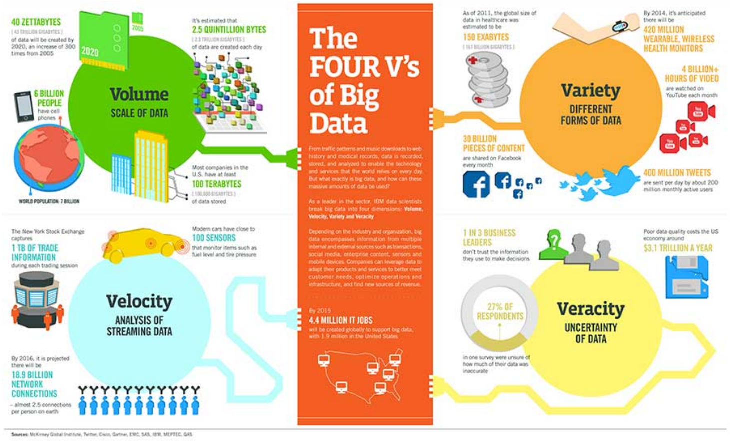
Big Data

Big Data

Big Data is yet another umbrella term bringing together all methods and disciplines involved in analyzing, capturing, curating, searching, sharing, storing, transferring, and visualizing large data sets. The term has been created out of the enormous increase in available data due to the Internet and subsequent phenomena.

Big Data is "defined" (in a non-mathematical sense) via 4 V's, namely Volume, Velocity, Variety, Veracity. It stems out of characterization by Gartner (Gartner, 2011) who used 3V's, refined from IBM.

Big Data



The FOUR V's of Big Data

From traffic patterns and music downloads to web history and medical records, data is recorded, stored, and analyzed to enable the technology and services that the world relies on every day. But what exactly is big data, and how can these massive amounts of data be used?

As a leader in the sector, IBM data scientists break big data into four dimensions: **Volume, Velocity, Variety and Veracity**.

Depending on the industry and organization, big data encompasses information from multiple internal and external sources such as transactions, social media, enterprise content, sensors and mobile devices. Companies can leverage data to adapt their products and services to better meet customer needs, optimize operations and infrastructure, and find new sources of revenue.

By 2015 4.4 MILLION IT JOBS will be created globally to support big data, with 1.9 million in the United States

40 ZETTABYTES [43 TRILLION GIGABYTES] of data will be created by 2020, an increase of 300 times from 2005

It's estimated that 2.5 QUINTILLION BYTES [2.3 TRILLION GIGABYTES] of data are created each day

6 BILLION PEOPLE have cell phones

Most companies in the U.S. have at least 100 TERABYTES [100,000 GIGABYTES] of data stored

The New York Stock Exchange captures 1 TB OF TRADE INFORMATION during each trading session

Modern cars have close to 100 SENSORS that monitor items such as fuel level and tire pressure

By 2016, it is projected there will be 18.9 BILLION NETWORK CONNECTIONS – almost 2.5 connections per person on earth

As of 2011, the global size of data in healthcare was estimated to be 150 EXABYTES [161 BILLION GIGABYTES]

By 2014, it's anticipated there will be 420 MILLION WEARABLE, WIRELESS HEALTH MONITORS

4 BILLION+ HOURS OF VIDEO are watched on YouTube each month

30 BILLION PIECES OF CONTENT are shared on Facebook every month

400 MILLION TWEETS are sent per day by about 200 million monthly active users

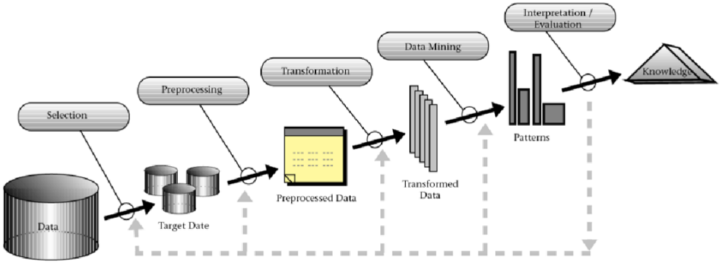
1 IN 3 BUSINESS LEADERS don't trust the information they use to make decisions

27% OF RESPONDENTS in one survey were unsure of how much of their data was inaccurate

Poor data quality costs the US economy around $3.1 TRILLION A YEAR

Sources: McKinsey Global Institute, Twitter, Cisco, Gartner, EMC, SAS, IBM, MEPTEC, QAS

The KDD Process



What is Python?

- Python is a modern, general-purpose programming language.
- It is object-oriented and high-level.
- Python is known for its simplicity and readability.
- It has a vast ecosystem of libraries and frameworks.

General Characteristics of Python

- Clean and simple language:
  - Easy-to-read and intuitive code
  - Easy-to-learn minimalistic syntax
  - Maintainability scales well with the size of projects
- Expressive language:
  - Fewer lines of code
  - Fewer bugs
  - Easier to maintain

Technical Details of Python

- Dynamically typed:
  - No need to define the type of variables
  - Function arguments or return types are not explicitly declared
- Automatic memory management:
  - No need to explicitly allocate and deallocate memory for variables and data arrays
  - No memory leak bugs
- Interpreted:
  - No need to compile the code
  - The Python interpreter reads and executes the Python code directly

Python: Pros and Cons

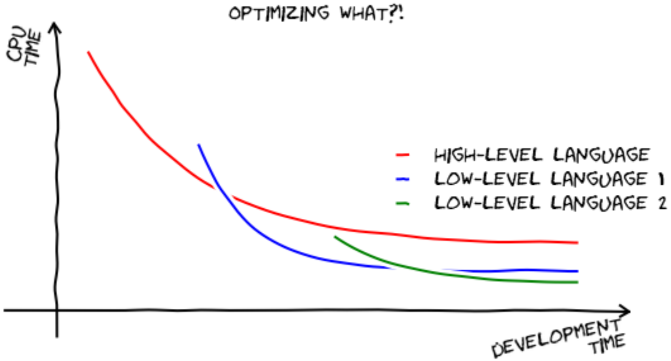
Advantages:

- Ease of programming and minimizing development time
- Modular and object-oriented programming
- Good system for packaging and code re-use
- Documentation tightly integrated with the code
- A large standard library, and a huge number of libraries

Disadvantages:

- Python execution can be slow compared to compiled statically typed languages (e.g., C and Fortran)
- Decentralized ecosystem with different environments, packages, and documentation
- Integration of fast C code (e.g., for inner loops) requires extensions like Cython

What Makes Python Suitable for Data Science?

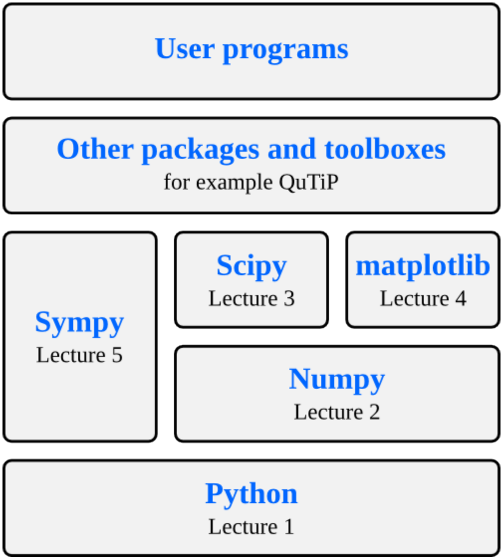


What Makes Python Suitable for Data Science?

Python has a strong position in scientific computing:

- Large community of users, easy to find help and documentation.
- Extensive ecosystem of important libraries and environments
  - numpy - Numerical Python
  - scipy - Scientific Python
  - matplotlib - graphics library
- Great performance due to close integration with time-tested and highly optimized codes written in C and Fortran:
  - blas, altas blas, lapack, arpack, Intel MKL, ...
- Good support for
  - Parallel processing with processes and threads
  - Interprocess communication (MPI)
  - GPU computing (OpenCL and CUDA)
- Readily available and suitable for use on high-performance computing clusters.
  - No license costs

The Scientific Python Software Stack



Python Environments

Python also refers to the standard implementation of the interpreter, technically referred to as:

- ➔ CPython - The interpreter runs the Python code on a computer.

The interpreter runs the Python code on a computer. Different environments through which Python can be used allow different workflows. One strength of Python is that it's versatile and can be used in complementary ways. However, it can be confusing for beginners, so we will start with a brief survey of Python environments that are useful for scientific computing.

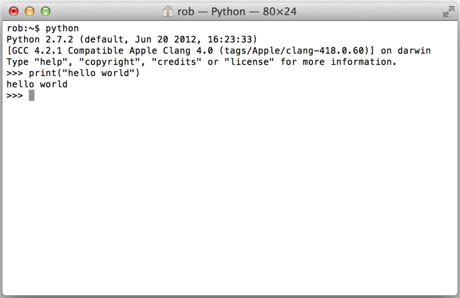
Python Interpreter

The standard way to use the Python programming language is to use the Python interpreter to run Python code. The Python interpreter is a program that reads and executes Python code in files passed to it as arguments. At the command prompt, the command `python` is used to invoke the Python interpreter.
For example, to run a file `my-program.py` that contains Python code from the command prompt, use:

```
$ python my-program.py
```

We can also start the interpreter by simply typing `python` at the command line and interactively type Python code into the interpreter.

Python Interpreter

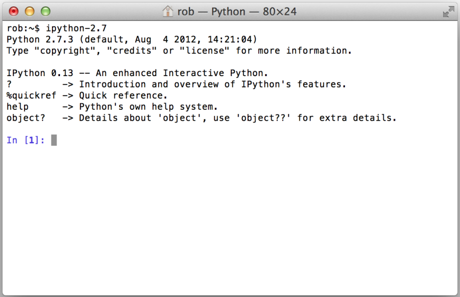


```
rob:~$ python
Python 2.7.2 (default, Jun 20 2012, 16:23:33)
[GCC 4.2.1 Compatible Apple Clang 4.0 (tags/Apple/clang-418.0.60)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> print("hello world")
hello world
>>> 
```

This is often how we want to work when developing scientific applications or when doing small calculations. However, the standard Python interpreter is not very convenient for this kind of work, due to a number of limitations.

IPython: Interactive Python

IPython is an interactive shell that significantly improves upon the standard Python interpreter. It's a powerful tool for scientific Python programming, offering an enhanced and user-friendly interactive environment for coding and experimentation.



```
rob:~$ ipython-2.7
Python 2.7.3 (default, Aug  4 2012, 14:21:04)
Type "copyright", "credits" or "license" for more information.

IPython 0.13 -- 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]: 
```

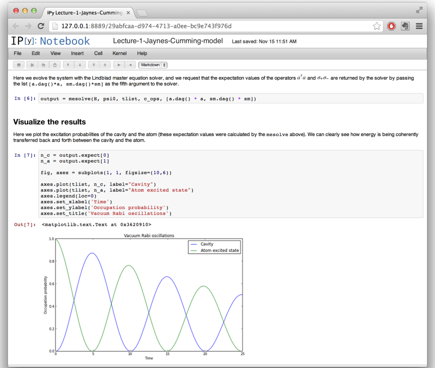
IPython: Interactive Python

IPython offers a wide range of useful features, including:

- Command history that can be browsed with the up and down arrows on the keyboard.
- Tab auto-completion for faster coding.
- In-line code editing for quick modifications.
- Object introspection and automatic extraction of documentation strings from Python objects like classes and functions.
- Seamless interaction with the operating system shell.
- Support for multiple parallel back-end processes, allowing for distributed computing on clusters or cloud services like Amazon EC2.

IPython Notebook

IPython Notebook is an HTML-based notebook environment for Python, similar to Mathematica or Maple. It is built on the IPython shell but provides a cell-based environment with great interactivity, allowing you to organize and document calculations in a structured way.


IPy Lecture-1-Jaynes-Cumming
127.0.0.1:8889/29abfcaa-d974-4713-a0ee-bc9e743f976d
IP[y]: Notebook
Lecture-1-Jaynes-Cumming-model
Last saved: Nov 15 11:51 AM
File Edit View Insert Cell Kernel Help
Markdown

Here we evolve the system with the Lindblad master equation solver, and we request that the expectation values of the operators $a^\dagger a$ and $\sigma_+\sigma_-$ are returned by the solver by passing the list `[a.dag()*a, sm.dag()*sm]` as the fifth argument to the solver.

```
In [6]: output = mesolve(H, psi0, tlist, c_ops, [a.dag() * a, sm.dag() * sm])
```

Visualize the results

Here we plot the excitation probabilities of the cavity and the atom (these expectation values were calculated by the `mesolve` above). We can clearly see how energy is being coherently transferred back and forth between the cavity and the atom.

```
In [7]: n_c = output.expect[0]
        n_a = output.expect[1]

        fig, axes = subplots(1, 1, figsize=(10,6))

        axes.plot(tlist, n_c, label="Cavity")
        axes.plot(tlist, n_a, label="Atom excited state")
        axes.legend(loc=0)
        axes.set_xlabel('Time')
        axes.set_ylabel('Occupation probability')
        axes.set_title('Vacuum Rabi oscillations')

Out[7]: <matplotlib.text.Text at 0x3620910>
```

Starting IPython Notebook

Although IPython notebooks use a web browser as a graphical interface, they are typically run locally on the same computer as the browser. To start a new IPython notebook session, run the following command from a directory where you want the notebooks to be stored:

```
$ ipython notebook
```

This will open a new browser window (or a new tab in an existing window) with an index page where existing notebooks are shown and from which new notebooks can be created.

Why and When to Use IPython Notebook?

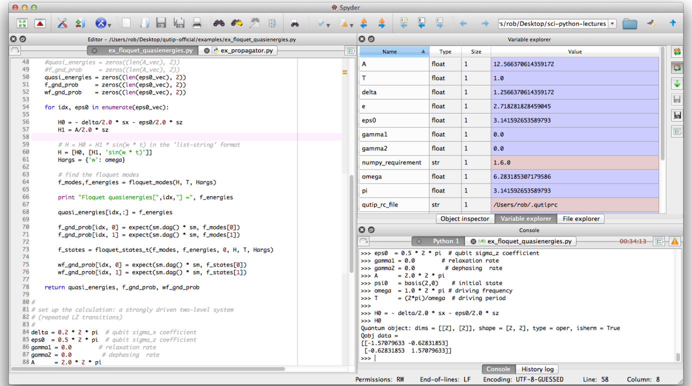
- All exercises can be done using IPython Notebook.
- There is a special introduction to IPython Notebook.
- Very useful for exploring unknown solutions.
- Follows the REPL Cycle - Read, Evaluate, Print, Loop.
- Allows you to document your work while working.
- A great resource for learning.
- Not very useful for large-scale industry projects.
  - No project management.
  - Missing refactoring.
  - Use IDEs in this case (Eclipse-based PyDev, Intellij's PyCharm).

Spyder

Spyder is a MATLAB-like IDE for scientific computing with Python. It offers many advantages of a traditional IDE environment, including:

- Code editing
- Execution
- Debugging
- Single environment for all tasks
- Project organization

Spyder



Spyder
s/rob/Desktop/sci-python-lectures
Editor - /Users/rob/Desktop/qutip-official/examples/ex_floquet_quasienergies.py
ex_floquet_quasienergies.py
ex_propagator.py
Variable explorer
Object inspector
Variable explorer
File explorer
Console
Python 1
ex_floquet_quasienergies.py
00:34:13
Console
History log

```
    #quasi_energies = zeros((len(A_vec), 2))
    #f_gnd_prob     = zeros((len(A_vec), 2))
    quasi_energies = zeros((len(eps0_vec), 2))
    f_gnd_prob     = zeros((len(eps0_vec), 2))
    wf_gnd_prob    = zeros((len(eps0_vec), 2))

    for idx, eps0 in enumerate(eps0_vec):

        H0 = - delta/2.0 * sx - eps0/2.0 * sz
        H1 = A/2.0 * sz

        # H = H0 + H1 * sin(w * t) in the 'list-string' format
        H = [H0, [H1, 'sin(w * t)']]
        Hargs = {'w': omega}

        # find the floquet modes
        f_modes,f_energies = floquet_modes(H, T, Hargs)

        print "Floquet quasienergies[",idx,"] =", f_energies

        quasi_energies[idx,:] = f_energies

        f_gnd_prob[idx, 0] = expect(sm.dag() * sm, f_modes[0])
        f_gnd_prob[idx, 1] = expect(sm.dag() * sm, f_modes[1])

        f_states = floquet_states_t(f_modes, f_energies, 0, H, T, Hargs)

        wf_gnd_prob[idx, 0] = expect(sm.dag() * sm, f_states[0])
        wf_gnd_prob[idx, 1] = expect(sm.dag() * sm, f_states[1])

    return quasi_energies, f_gnd_prob, wf_gnd_prob

#
# set up the calculation: a strongly driven two-level system
# (repeated LZ transitions)
#
delta = 0.2 * 2 * pi  # qubit sigma_x coefficient
eps0  = 0.5 * 2 * pi  # qubit sigma_z coefficient
gamma1 = 0.0         # relaxation rate
gamma2 = 0.0          # dephasing  rate
A      = 2.0 * 2 * pi
```

Name	Type	Size	Value
A	float	1	12.566370614359172
T	float	1	1.0
delta	float	1	1.2566370614359172
e	float	1	2.718281828459045
eps0	float	1	3.141592653589793
gamma1	float	1	0.0
gamma2	float	1	0.0
numpy_requirement	str	1	1.6.0
omega	float	1	6.283185307179586
pi	float	1	3.141592653589793
qutip_rc_file	str	1	/Users/rob/.qutiprc

```
>>> eps0  = 0.5 * 2 * pi  # qubit sigma_z coefficient
>>> gamma1 = 0.0        # relaxation rate
>>> gamma2 = 0.0         # dephasing  rate
>>> A      = 2.0 * 2 * pi
>>> psi0   = basis(2,0)    # initial state
>>> omega  = 1.0 * 2 * pi # driving frequency
>>> T      = (2*pi)/omega  # driving period
>>>
>>> H0 = - delta/2.0 * sx - eps0/2.0 * sz
>>> H0
Quantum object: dims = [[2], [2]], shape = [2, 2], type = oper, isherm = True
Qobj data =
[[-1.57079633 -0.62831853]
 [-0.62831853  1.57079633]]
>>> |
```

Permissions: RW   End-of-lines: LF   Encoding: UTF-8-GUESSED   Line: 58   Column: 8

Advantages of Spyder

Some advantages of Spyder include:

- Powerful code editor with syntax highlighting
- Dynamic code introspection
- Integration with the Python debugger
- Variable explorer
- IPython command prompt
- Integrated documentation and help

Versions of Python

There are two main versions of Python:

1. Python 2 (2.7 still contains the majority of libraries)
2. Python 3 (not backward compatible)

For this lecture, we will use Python 2.7.

To check which version of Python you have, run the following commands:

```
$ python --version
Python 2.7.3
$ python3.2 --version
Python 3.2.3
```

Note that several versions of Python can be installed in parallel.

Installation on Linux

On Ubuntu Linux, you can install Python and its requirements using the following commands:

```
$ sudo apt-get install python ipython ipython-notebook
$ sudo apt-get install python-numpy python-scipy
python-matplotlib python-sympy
$ sudo apt-get install spyder
```

These commands will help you set up Python and essential libraries.

Installation on macOS (Macports)

On macOS, you can install Python and its requirements using Macports. First, install a new Python environment with the necessary packages:

```
$ sudo port install py27-ipython
+pyside+notebook+parallel+scientific
$ sudo port install py27-scipy py27-matplotlib py27-sympy
$ sudo port install py27-spyder
```

These commands will associate the commands `python` and `ipython` with the versions installed via Macports.

To select the Macports Python and IPython versions over the default macOS versions, run the following commands:

```
$ sudo port select python python27
$ sudo port select ipython ipython27
```

Installation on Windows

Windows lacks a good packaging system, so the easiest way to set up a Python environment is to install a pre-packaged distribution. Here are some good alternatives:

- **Enthought Python Distribution (EPD)**: EPD is a commercial product but is available for free for academic use.
- **Anaconda CE**: Anaconda Community Edition is free, while Anaconda Pro is a commercial product.
- **Python(x,y)**: Python(x,y) is a fully open-source distribution.

Note

EPD and Anaconda CE are also available for Linux and macOS.

Further Reading

- Python: https://www.python.org (Official Python website)
- Python Tutorials: https://docs.python.org/2/tutorial/ (Official Python tutorials)
- Think Python: https://greenteapress.com/wp/think-python (A free book on Python)
- Learn Python the Hard Way: https://learnpythonthehardway.org/book (An exercise book for learning programming)
- Vasant Dhar. 2013. Data science and prediction. Commun. ACM 56, 12 (December 2013), 64-73. DOI=10.1145/2500499 http://doi.acm.org/10.1145/2500499
- Jeff Leek (2013-12-12). "The key word in "Data Science" is not Data, it is Science". http://simplystatistics.org/2013/12/12/the-key-word-in-data-science-is-not-data-it-is-science/
- Beyer, Mark (2011). "Gartner Says Solving 'Big Data' Challenge Involves More Than Just Managing Volumes of Data". Gartner. Archived, http://www.gartner.com/newsroom/id/1731916, last