A Data Science Learning and Project Book

by Patrick Prunty

March 1, 2022

Declaration

Declare some things here ...

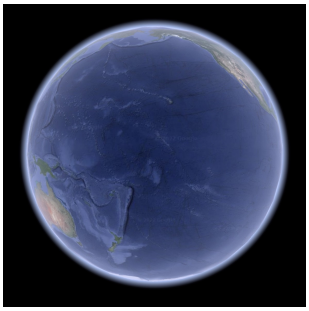
Contents

Chapter 1

Introduction

Just an example of 4 images side by side, move to ch. 2 to see some stuff!

(a) label 1



(b) label 2

Figure 1.1: 2 Figures side by side

(a) label 1

(b) label 2

Figure 1.2: 2 Figures side by side

Chapter 2

Python for Data Science

2.1 Jupyter Notebooks

Project Jupyter is a project and community whose goal is to "develop open-source software, open-standards, and services for interactive computing across dozens of programming languages". It was spun off from IPython in 2014 by Fernando Pérez. Project Jupyter's name is a reference to the three core programming languages supported by Jupyter, which are Julia, Python and R, and also a homage to Galileo's notebooks recording the discovery of the moons of Jupiter. Project Jupyter has developed and supported the interactive computing products Jupyter Notebook, JupyterHub, and JupyterLab. Put simply, it is a web interface for python programs that allows for easy distribution and documentation handling.

2.1.1 Installation

Jupyter Notebook can easily be installed using pip install:

```
pip install notebook
```

2.1.2 Usage

The following section will detail some usage and caveats of Jupyter Notebooks. This will be done using a Q&A format that will be useful for future reference.

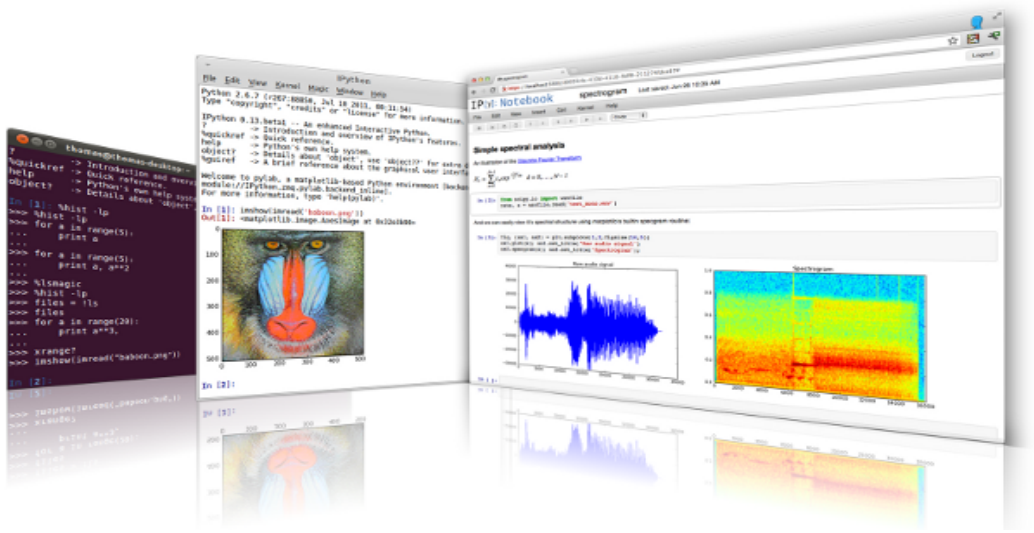


Figure 2.1: Example of iPython Notebook Presentation

How do I start a notebook session?

Once Jupyter Notebook is installed, you can carry out the following steps:

1. Open a terminal sessions and navigate to a directory in which you would like to open a notebook.

2. Start the notebook server from the command line:

```
jupyter notebook
```

3. You should see the notebook open in your browser.

This will print some information about the notebook server in your terminal, including the URL of the web application (by default, `http://localhost:8888`):
When the notebook opens in your browser, you will see the Notebook Dashboard (2.3), which will show a list of the notebooks, files, and subdirectories in the directory where the notebook server was started. Most of the time, you will wish to start a notebook server in the highest level directory containing notebooks. Often this will be your home directory. It might be a good idea to create this notebook directory to store all future notebook projects.

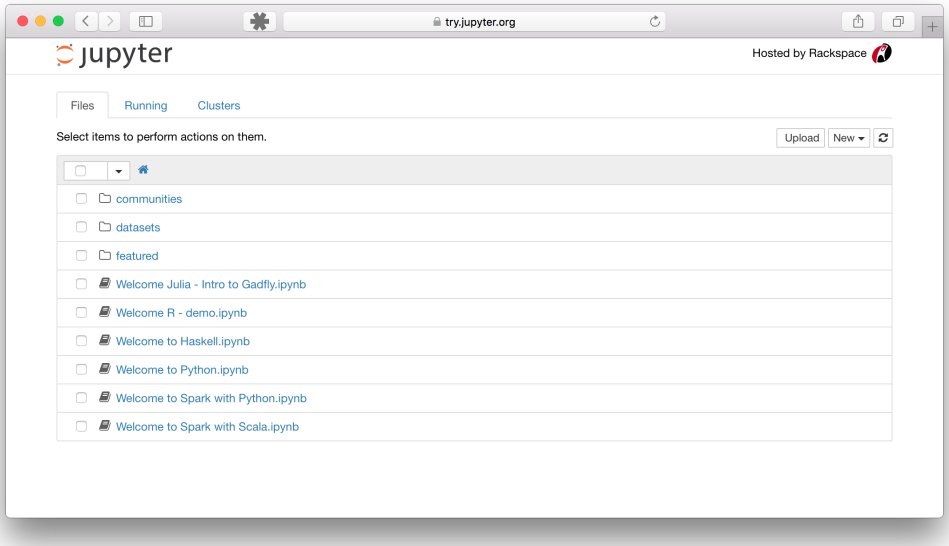


Figure 2.2: Jupyter Notebook Dashboard

How do I open a specific Notebook?

The following code should open the given notebook in the currently running notebook server, starting one if necessary.

```
jupyter notebook <notebook_name>.ipynb
```

How do I start the Notebook using a custom IP or port?

2.1.3 DataSpell

An alternative to running these commands inside a terminal is Jetbrain's DataSpell. DataSpell is an IDE for data science with intelligent Jupyter notebooks, interactive Python scripts, and lots of other built-in tools. It is commonly advertised as the IDE for Data Scientists and can be downloaded using the following here - `https://www.jetbrains.com/dataspell/`

2.2 Notebooks Project Structure

Notebooks should primarily be used to represent a project. Scratch work is well and good, but the function of notebooks is to have a well-presented piece of work with step-by-step story-telling between code.
Below is a sample of what the structure of a notebook should look like:

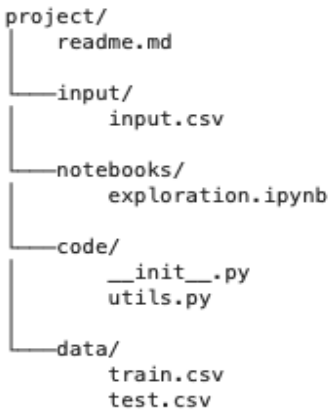


Figure 2.3: Jupyter Notebook Project Structure

Using the following structure, the first few cells of the notebook should look like so:

In [1]:
```
\%load_ext autoreload
\%autoreload 2
import os
import sys
module_path = os.path.abspath(os.path.join('..', '
    ↪ code'))
if module_path not in sys.path:
    sys.path.append(module_path)
```

In [2]:
```
import utils # from project/code
```

In [3]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

The final cell imports a number of Python packages. These are some of the core Python libraries used in Data Science and will be covered in the next section.

2.3 Data Science Libraries

The following section covers some of the core Python packages used in data science. They are as follows:

- Pandas - Data Analysis
- NumPy- Mathematical Operations
- SciPy - Scientific and High-Performance Computing
- BeautifulSoup - Web Scraping
- Matpolotlib - Data Visualisation
- Seaborn - Statistical Data Visualisation

Each of these libraries will be explored in detail, however, I recommend the reader does not to use these exercises as a main function of their learning. Think of this section as more of a troubleshooting space and one which we can refer back to in the next chapter when we put some of these packages to use. So yes, feel free to move to the next chapter and we will return to the gritty stuff on the proceeding pages when we need to!

2.3.1 Pandas

`pandas` is a software library written for the Python programming language for data manipulation and analysis. The name is derived from the term "panel data", an econometrics term for data sets that include observations over multiple time periods for the same individuals. Its name is a play on the phrase "Python data analysis". Wes McKinney started building what would become pandas at AQR Capital while he was a researcher there from 2007 to 2010.

Getting Started

To use pandas, you'll typically start with the following line of code:

```
In [4]:  # this is a comment
         import pandas as pd
```

Creating Data

There are two core objects in pandas: the **DataFrame** and the **Series**.

DataFrame A DataFrame is a table. It contains an array of individual entries, each of which has a certain value. Each entry corresponds to a row (or record) and a column.
For example, consider the following simple DataFrame:

```
In [5]:  pd.DataFrame({'Yes': [50, 21], 'No': [131, 2]})
```

```
Out[5]:
        Yes   No
    0    50  131
    1    21    2
```

DataFrame entries are not limited to integers. For instance, here's a DataFrame whose values are strings:

```
In [6]:  pd.DataFrame({'Bob': ['I liked it.', 'It was awful.
            ↪ '], 'Sue': ['Pretty good.', 'Bland.']})
```

```
Out[6]:
            Bob            Sue
    0  I liked it.    Pretty good.
    1  It was awful.  Bland.
```

We are using the `pd.DataFrame()` constructor to generate these DataFrame objects. The syntax for declaring a new one is a dictionary whose keys are

the column names (Bob and Sue in this example), and whose values are a list of entries. This is the standard way of constructing a new DataFrame, and the one you are most likely to encounter.
The dictionary-list constructor assigns values to the column labels, but just uses an ascending count from 0 (0, 1, 2, 3, ...) for the row labels. Sometimes this is OK, but oftentimes we will want to assign these labels ourselves.
The list of row labels used in a DataFrame is known as an Index. We can assign values to it by using an index parameter in our constructor:

```
In [7]: pd.DataFrame({'Bob': ['I liked it.', 'It was awful.
        ↪ '],
                      'Sue': ['Pretty good.', 'Bland.']},
                     index=['Product A', 'Product B'])
```

```
Out[7]:                Bob            Sue
           Product A   I liked it.    Pretty good.
           Product B   It was awful.  Bland.
```

Series

A Series, by contrast, is a sequence of data values. If a DataFrame is a table, a Series is a list. And in fact you can create one with nothing more than a list:

Reading Data

2.4 NumPy

2.5 Gnuplot with Python

2.6 SciPy

2.7 Data Science Project

Chapter 3

Web Scraping, Regular Expressions and viz

Chapter 4

Pandas, SQL and Grammar of Data

Chapter 5

Statistical Models

Chapter 6

Probability, Distributions and Frequentest Statistics

Chapter 7

Story Telling and Effective Communication

Chapter 8

Regression and Logistic Regression

Chapter 9

Machine Learning, SVM and Evaluation

Bibliography

[1] Shamos M I, Hoey D. *Closest-point problems*. In Proc. 16th IEEE Annu. Symp. Found. Comput. Sci., Berkeley, US, 1975, pp.151–162.

[2] Inderjit S. Dhillon1 and Dharmendra S. Modha. *A Data-Clustering Algorithm On Distributed Memory Multiprocessors*, 'In Large-Scale Parallel Data Mining, Lecture Notes in Artificial Intelligence', p245–260, (2000).

[3] Okabe, Atsuyuki and Boots, Barry and Sugihara, Kokichi. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*, John Wiley Sons, Inc. (1992).

[4] S. Fortune. *A Sweep-Line algorithm for Voronoi diagrams*, Algorithmica, Vol. 2 Issue 1-4, p153-174, 22p, (Nov. 1987).

[5] R. Morrin *5614: Assignment 5. Concurrency with Convex Hulls*. Assignment for TCD High-Performance Computing Class of 2020. May 2, 2020.

[6] Pixar in a Box. Website. Khan Academy. *Voronoi Partition, Patterns, Computer Animation* `https://www.khanacademy.org/computing/pixar/pattern/dino/v/patterns2_new`

[7] L. D. Libersky, et al. *High strain Lagrangian hydrodynamics: a three-dimensional SPH code for dynamic material response*, Journal of computational physics **109** (1), p67-75, (1993).

[8] R. Descartes. *Principia Philosophiae*. Ludovicus Elzevirius, Amsterdam, 1644.

[9] G. Voronoy (1908a). *Nouvelles applications des paramètres continus à la théorie des formes quadratiques. Premier mémoire. Sur quelques propriétés des formes quadratiques positives parfaites* . Journal für die Reine und Angewandte Mathematik. 1908 (133): 97–178.

[10] Hartigan, J.A. *Clustering Algorithms*. Wiley (1975)

[11] Freedman, David; Diaconis, Persi. *On the histogram as a density estimator: L2 theory". Probability Theory and Related Fields*. 57 (4): 453–476. (December 1981).

[12] F. Aurenhammer,

[13] Sheldon M. Ross. *Simulation*. Knovel Library. Academic Press, 2012.