Lecture 1 : Jupyter Basics

Data Science, DST, UIC

Why Python?

Python has emerged over the last couple decades as a first-class tool for scientific computing tasks, including the analysis and visualization of large datasets. This may have come as a surprise to early proponents of the Python language: the language itself was not specifically designed with data analysis or scientific computing in mind. The usefulness of Python for data science stems primarily from the large and active ecosystem of third-party packages: NumPy for manipulation of homogeneous array-based data, Pandas for manipulation of heterogeneous and labeled data, SciPy for common scientific computing tasks, Matplotlib for publication-quality visualizations, IPython for interactive execution and sharing of code, Scikit-Learn for machine learning, and many more tools that will be mentioned in the following pages.

If you are looking for a guide to the Python language itself, I would suggest a book, "A Whirlwind Tour of the Python Language". This short report provides a tour of the essential features of the Python language, aimed at data scientists who already are familiar with one or more other programming languages.

Python 2 vs Python 3

In this course, we will uses the syntax of Python 3, which contains language enhancements that are not compatible with the 2.x series of Python. Though Python 3.0 was first released in 2008, adoption has been relatively slow, particularly in the scientific and web development communities. This is primarily because it took some time for many of the essential third-party packages and toolkits to be made compatible with the new language internals. Since early 2014, however, stable releases of the most important tools in the data science ecosystem have been fully compatible with both Python 2 and 3, and so this course will use the newer Python 3 syntax.

PEP8-Style Guide for Python Code

https://legacy.python.org/dev/peps/pep-0008/ (https://legacy.python.org/dev/peps/pep-0008/)

To Write Python codes, we need the following

1. **An editor**: Such as Windows NodePad, GEdit, Sublime, Spyder, PyCharm, etc, basically any text editor will do;
2. **An intepreter**: Python is an interpreted language, which is a type of programming language for which most of its implementations execute instructions directly and freely, without previously compiling a program into machine-language instructions. Python has different version of intepreters such as Cpython (default), Jpython, Ironpython, etc;
3. **Packages**: One of the most important advantage of Python is its packages system,it includes more than 10000 different packages up to now. Packages are like function libraries that are implemented by the others and can be used easily in your codes to do something specific. It is very easy to use packages in Python, just a simple "import xxx" will do the work. We will learn "Numpy", "Pandas", "Scikit_learn" and other packages that are closely related to data science in this course.
4. Package manager: software that manage the packages, install, upgrade or delete them. Such as Pip, conda, etc.

In practice, we can use different products above in combination. Or, much easier for beginners, we can use a integrated solution such as "jupyter notebook" where we can write, run, debug and learn python programming in one place.

1. How to use juypter notebook to write codes

Jupyter notebook is one of the front end of Anaconda python development environment. There are two fairly prominent terms that you should notice, which are probably new to you: **cells** and **kernels** are key both to understanding Jupyter and to what makes it more than just a word processor.

- A **kernel** is a “computational engine” that executes the code contained in a notebook document.
- A **cell** is a container for text to be displayed in the notebook or code to be executed by the notebook’s kernel.

1.1 Kernels

Behind every notebook runs a kernel. When you run a code cell, that code is executed within the kernel and any output is returned back to the cell to be displayed. The kernel’s state persists over time and between cells — it pertains to the document as a whole and not individual cells.

For example, if you import libraries or declare variables in one cell, they will be available in another. In this way, you can think of a notebook document as being somewhat comparable to a script file, except that it is multimedia.

If you ever wish to reset things, there are several incredibly useful options from the Kernel menu:

- **Restart**: restarts the kernel, thus clearing all the variables etc that were defined.
- **Restart & Clear Output**: same as above but will also wipe the output displayed below your code cells.
- **Restart & Run All**: same as above but will also run all your cells in order from first to last.

If your kernel is ever stuck on a computation and you wish to stop it, you can choose the Interupt option.

1.2 Cells

- **Cell**: The jupyter notebook are files ended with *.ipynb*. Inside the file, each notebook consists a bunch of Cells. Cells are just blocks of texts in Markdown formats or codes.
  - Code cells allow you to enter and run code. Run a code cell using `Shift-Enter`.
  - Markdown cells can contain text. You can change the cell type to Markdown by using the Cell menu, the toolbar, or the key shortcut *m* (in command mode).
    - *Markdown is a popular markup language that is a superset of HTML. For markdown tips, refer to [Markdown Cheatsheet]*(http://nestacms.com/docs/creating-content/markdown-cheat-sheet (http://nestacms.com/docs/creating-content/markdown-cheat-sheet)).
- **Mode of cell**: a cell has two possible status, **Command** mode and **Edit** mode
  - When a cell is in a **blue rectangle frame**, it is in **"Command"** mode, it means you can edit the cell as a whole, for example, you can copy, paste, delete, merge, insert cells in this status, using the menu in jupyter notebook interface, or with keyboard shortcuts. To enter the **Command** mode press **ESC** or click anywhere outside the cell. Here are some frequently used shortcuts for editing cells as a whole.

> c(copy), x(cut), v(paste), dd(delete), a(insert before), b(insert after), o(dislay/hide output), l(line number), shift+m(merge)

- When a cell is in a **green rectangle frame**, it is in the **"Edit"** mode, it means you can edit the content of the cell.You can enter this mode by pressing `enter` or clicking the mouse on a cell editor area. Here are some frequently used shortcuts for running and editing code cells.

> ctrl+enter(run the cell), shift+enter(run the current cell and jump to next cell), alt+enter(run the current cell and insert a new cell below), ctrl+/(comment the current line)

There are a lot of other shortcuts and manipulations in jupyter notebook, you can always press 'h' in you keyboard when the cell is in **'Command'** mode to see the help, we also recommend you to go to jupyter docs (https://jupyter-notebook.readthedocs.io/en/stable/) to check it out.

1.3 Inputs

- Input cells are code cells that the user can write code in. The code will be passed into the **kernel** of the jupyter notebook to interpret and run. And results will be displayed in the following output cells, which will be discussed later.
- You can write python syntax statements in the input cells as you are writing python in any editor or IDE.
- In python, line starts with # is comment.
- Input cells start with a **In [ ]** in the left side of the cell, the number in the **[ ]** means the execution order of the cell. For example, a cell with **In [6]** at the left side means the cell in the 6th cell to be executed in the current notebook. **In [*]** means the current cell in still being executed. **In [ ]** means the current cell has not been run yet.

- to run the cell, press `ctrl+enter`

In [1]:
```
# define variable x
x=1
print(x)
```
```
1
```

- the variables in one notebook belong to the same python program file

In [2]:
```
# define variable y
y=x+1
print(y)
```
```
2
```

- In data science projects, tracing and displaying the current value of varibles is very important
- you may run one cell multiple times

In [3]:
```
y=y+1
print(y)
```
```
3
```

- In jupyter notebook, the order of cell execution is not decided by the order of the cells written in the notebook, but rather depends on the actual run order of the cell which can be decided by the user.
- Most of the time, we run cells in their natural order.

In [4]:
```
z=x+y
print(z)
```
```
4
```

1.4 Outputs

- Outputs display the output messages in jupyter notebook. The outputs have **Out [ ]** at the left side
- The number in the **Out [ ]** is identical to the input cell that produce this output
- Jupyter notebook will display the evaluation result of the last line. If you do not want the result to be displayed, you can add ; after last line of code.

```
In [5]: test = 'this is an output'
        test
```

```
Out[5]: 'this is an output'
```

```
In [6]: x + y
```

```
Out[6]: 4
```

```
In [7]: x+y;
```

- In Python, we use **print()** function to print contents to the terminal, which is different in the output in jupyter notebook.
- When writing a complete python program in a .py file, please use **print()** function instead of the syntax in jupyter notebook.

```
In [8]: test = 'difference between print function and output in jupyter notebook'
        print(test)
        test
```

```
difference between print function and output in jupyter notebook
```

```
Out[8]: 'difference between print function and output in jupyter notebook'
```

1.5 Errors and exceptions

- It is more than often we encounter **error** messages when programming, we should see them as **friends** that are really helpful instead of **devils**.
- **Error** messages usually give you a lot of clues of what go wrong and how to correct them

```
In [9]: defined_variable = 0
        undefined_variable
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-9-d0457fdc6164> in <module>
      1 defined_variable = 0
----> 2 undefined_variable

NameError: name 'undefined_variable' is not defined
```

- Read the **error** messages carefully each time you encounter them, as you gain more and more experience, you will be able to correct them more and more easily
- Some of the common errors that can be made by beginners can be find here (https://inventwithpython.com/blog/2012/07/09/16-common-python-runtime-errors-beginners-find/)
- You can always use google or Python docs (https://docs.python.org/3/tutorial/errors.html) for more comprehensive explanations to the **errors** you encounter

1.6 Read in data files from outside the program

- The same as working with any programming language, if you want to read a file in the program, you need to specify the path the file actually lies in, or you can copy the file to your curret **working folder**
- The way to check your current working folder is this

```
In [ ]: import os # import the package "os" that contains functions related to the operation system
        print(os.getcwd())
```

- Now you can download the file **bc_data.csv** from iSpace and
  - upload the file to your working directory of the jupyter hub server you are using, or
  - put the file in your working directory locally if you are using your own computer and jupyter notebook environment

```
In [11]: from pandas import read_csv  # 'pandas' is a powerful data processing package in python which
         we will introduce later
         data = read_csv('bc_data.csv')
         data.head(2)
```

Out[11]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	comp
0	842302	M	17.99	10.38	122.8	1001.0	0.11840	
1	842517	M	20.57	17.77	132.9	1326.0	0.08474	

2 rows × 32 columns