经济大数据与 Python 应用

Jupyter: Beyond Normal Python

陈洲 湘潭大学商学院 2025

Press Space for next page \rightarrow

Contents

Tools for interactive analysis

- Help and Documentation
- Magic Commands
- IO History
- Shell Command

Not covered in this lecture

- Errors and Debugging
- Timing and Profiling

This part: tools for interactive analysis

- Jupyter and IPython features
- Useful magic commands
 - speed up common tasks in creating and using data science code.
- Features of the notebook
 - useful for understanding data and sharing results

Three modes of working

- IPython shell for trying out short sequences of commands
- Jupyter Notebook for longer interactive analysis and for sharing content with others
- interactive development environments (IDEs) like Emacs or
 VSCode for creating reusable Python packages

This course: IPython and Jupyter

unching the IPython Shell

```
on the command line

on 3.11.4 | packaged by conda-forge | (main, Jun 10 2023, 17:59:51) [MSC v.1935 64 bit (AMD 'copyright', 'credits' or 'license' for more information non 8.14.0 -- An enhanced Interactive Python. Type '?' for help.
```

```
1]: exit
```

ipython

Launching the Jupyter Notebook

The Jupyter Notebook is a browser-based graphical interface to the IPython shell.

```
$ jupyter lab
```

- DO NOT enter the \$ sign in the terminal.
- DO NOT close the terminal window while the notebook is running.

Features:

- formatted text
- static and dynamic visualizations
- mathematical equations

Graphical interface to the IPython shell

? Difference between IPython and Jupyter notebook?

Jupyter Features

Help and Documentation in IPython Accessing Documentation with ?

Every Python object contains a reference to a string, known as a *docstring*

- Concise summary of the object and how to use it.
- Built-in help function

Demo

```
In [1]: help(len)
Help on built-in function len in module builtins:
len(obj, /)
    Return the number of items in a container.
```

A shorthand for accessing this documentation

```
In [2]: len?
Signature: len(obj, /)
Docstring: Return the number of items in a container.
Type: builtin_function_or_method
```

This notation works for just about **anything**, including object methods:

```
In [3]: L = [1, 2, 3]
In [4]: L.insert?
Signature: L.insert(index, object, /)
Docstring: Insert object before index.
Type: builtin_function_or_method
```

Or even objects themselves, with the documentation from their type:

```
In [5]: L?
Type: list
String form: [1, 2, 3]
Length: 3
Docstring:
Built-in mutable sequence.

If no argument is given, the constructor creates a new empty list.
The argument must be an iterable if specified.
```

? Difference between function help and question mark?

help is a python feature. ? is a IPython feature.

· I mere a construction of a c

Exploring Modules with **Tab Completion**

Tab completion of object contents

To narrow down the list

type the first character or several characters

Demo

If there is only a single option

• pressing the Tab key will complete the line for you.

Demo: L.count:

In [10]: L.cou<TAB>

Tab completion when importing

Tab completion is also useful when importing objects from packages.

Demo: imports in the itertools package that start with co:

To see which imports are available on your system

Demo

IPython Magic

Commands

Prefixed by the % character

Adds on top of the normal Python syntax: *magic commands*

line magics, which are denoted by a single % prefix

cell magics, which are denoted by a double \magica prefix

Magic commands come in two flavors:

Running External Code: %run

Demo: create a *myscript.py* file with the following contents:

```
# file: myscript.py

def square(x):
    """square a number"""
    return x ** 2

for N in range(1, 4):
    print(f"{N} squared is {square(N)}")
```

You can execute this from your IPython session as follows:

```
In [6]: %run myscript.py
1 squared is 1
2 squared is 4
3 squared is 9
```

Any functions defined within it are available for use in your IPython session:

```
In [7]: square(5)
Out[7]: 25
```

Timing Code Execution: %timeit

```
In [8]: %timeit L = [n ** 2 for n in range(1000)]
430 µs ± 3.21 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

Perform multiple runs in order to attain more robust results.

For multiline statements, adding a second % sign

For example, here's the equivalent construction with a for loop:

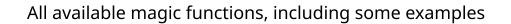
```
In [9]: %%timeit
...: L = []
...: for n in range(1000):
...: L.append(n ** 2)
...:
484 µs ± 5.67 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

 List comprehensions are about 10% faster than the equivalent for loop construction in this case

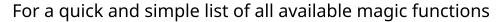
Help on Magic Functions: ?, %magic, and %lsmagic

To read the documentation of the %timeit magic function, simply type this:

```
In [10]: %timeit?
```



In [11]: %magic



In [12]: %lsmagic

Input and Output

History

IPython creates some Python variables called In and Out

Automatically updated to reflect this history:

```
import math
math.sin(2)
math.cos(2)
In [4]: In
Out[4]: ['', 'import math', 'math.sin(2)', 'math.cos(2)', 'In']
In [5]: Out
Out[5]:
{2: 0.9092974268256817,
3: -0.4161468365471424,
4: ['', 'import math', 'math.sin(2)', 'math.cos(2)', 'In', 'Out']}
```

In [1] can refer to the first command:

```
In [6]: print(In[1])
import math
```

The Out object is **not** a list

but a dictionary mapping input numbers to their outputs (if any):

```
In [7]: print(Out[2])
0.9092974268256817
```

Not all operations have outputs:

Any command that returns None is not added to Out.

output.

import statements and print statements don't affect the

Demo: let's check the sum of sin(2) ** 2 and cos(2) ** 2 using the previously computed results:

```
In [8]: Out[2] ** 2 + Out[3] ** 2
Out[8]: 1.0
```

Underscore Shortcuts and Previous Outputs

Shortcut for accessing previous output:

the variable _ (i.e., a single underscore)

```
In [9]: print(_)
1.0
```

IPython takes this a bit further

- Double underscore to access the second-to-last output
- Triple underscore to access the third-to-last output (skipping any commands with no output):

```
In [10]: print(__)
-0.4161468365471424

In [11]: print(__)
0.9092974268256817
```

```
    Shorthand for Out[X] is _X
```

• A single underscore followed by the **line number**:

```
In [12]: Out[2]
Out[12]: 0.9092974268256817

In [13]: _2
Out[13]: 0.9092974268256817
```

Suppressing Output

Add a semicolon to the end of the line:

```
In [13]: math.sin(2) + math.cos(2);
```

The result is computed silently

- Output is neither displayed
- Nor stored in the Out dictionary:

```
In [15]: 14 in Out
Out[15]: False
```

Coding exercises

Refresh your fingers.

Task 1: Fibonacci sequence

The Fibonacci numbers may be defined by

$$F_0 = 0, F_1 = 1$$

and

$$F_n = F_{n-1} + F_{n-2}$$

• For example, fib(0) = 0, fib(19) = 4181

For loop

```
def fib(n):
    if n = 0:
        return 0
    elif n = 1:
        return 1
    else:
        a, b = 0, 1
        for _ in range(2, n+1):
            a, b = b, a + b
        return b
```

Recursion

```
def fib2(n):
    if n = 0:
        return 0
    elif n = 1:
        return 1
    else:
        return fib2(n-2) + fib2(n-1)
```

Task 2: Expected income

The income of a graduate at graduation is determined by their effort e in college and the economic situation s.

- In times of economic prosperity, they receive an income bonus
- While in times of economic downturn, they receive an income penalty.

The equation for income can be expressed as

$$I^E(e) = lpha e + \mathbb{E}[s],$$

where lpha=1 represents the marginal payoff of effort, with probabilities $\Pr(s=100)=0.7$ and $\Pr(s=-100)=0.3$.

Find $I^E(5000)$

```
def expected_income(h, pl=0.3, ph=0.7, alpha=1):
    expected_shock = pl * -100 + ph * 100
    return alpha * h + expected_shock
```

What if

$$egin{aligned} \Pr(s = 200) &= 0.1 \ \Pr(s = 100) &= 0.7 \ \Pr(s = -100) &= 0.2 \ \Pr(s = -200) &= 0.1 \end{aligned}$$

```
def expected_income2(h, states, probabilities, alpha):
    expected_shock = sum(s*p for (s, p) in zip(states, probabilities))
    return alpha*h + expected_shock
```

More IPython

Resources

Web Resources

- The IPython website
- The nbviewer website
- A curated collection of Jupyter notebooks

Books

- Python for Data Analysis (O'Reilly)
- Learning IPython for Interactive Computing and Data Visualization (Packt)
- IPython Interactive Computing and Visualization Cookbook (Packt)

