Nicholas Bowman, Sonja Johnson-Yu, Kylie Jue
CS 106AP

Handout #15
August 5, 2019

Jupyter Reference Guide

This handout goes over the basics of Jupyter notebooks, including how to install Jupyter, launch a notebook, and run its cells. Jupyter notebooks are a common tool used across different disciplines for exploring and displaying data, and they also make great interactive explanatory tools.

Installing Jupyter

To install Jupyter, run the following command inside your Terminal (replace **`python3`** with **`py`** if you're using a Windows device):

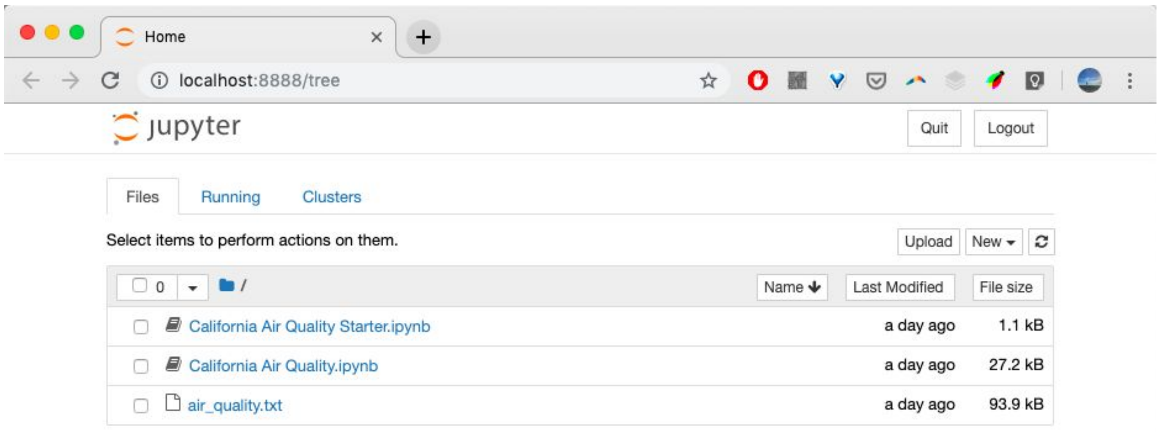
```
$ python3 -m pip install jupyter
```

Launching a notebook

A Jupyter notebook is a file that ends in the extension **`.ipynb`**, which stands for "**i**nteractive **Py**thon **n**ote**b**ook." To launch a Python notebook after you've installed Jupyter, you should first navigate to the directory where your notebook is located (you'll be there by default if you're using the PyCharm terminal inside a project). Then run the following command inside your Terminal (if this doesn't work, see the "Troubleshooting" section below):

```
$ jupyter notebook
```

This should open a window in your browser (Chrome, Firefox, Safari, Edge, etc.) that looks like Figure 1. Jupyter shows the files inside the current directory and allows you to click on any Python notebook files within that folder.



California Air Quality Starter.ipynb	a day ago	1.1 kB
California Air Quality.ipynb	a day ago	27.2 kB
air_quality.txt	a day ago	93.9 kB

Figure 1: After running the **`jupyter notebook`** command, you should see a window that lists the Python notebooks inside the current directory from which you ran the command. The picture above shows the Lecture 25 directory.

To launch a particular notebook, click on its file name. This should open a new tab with the notebook.

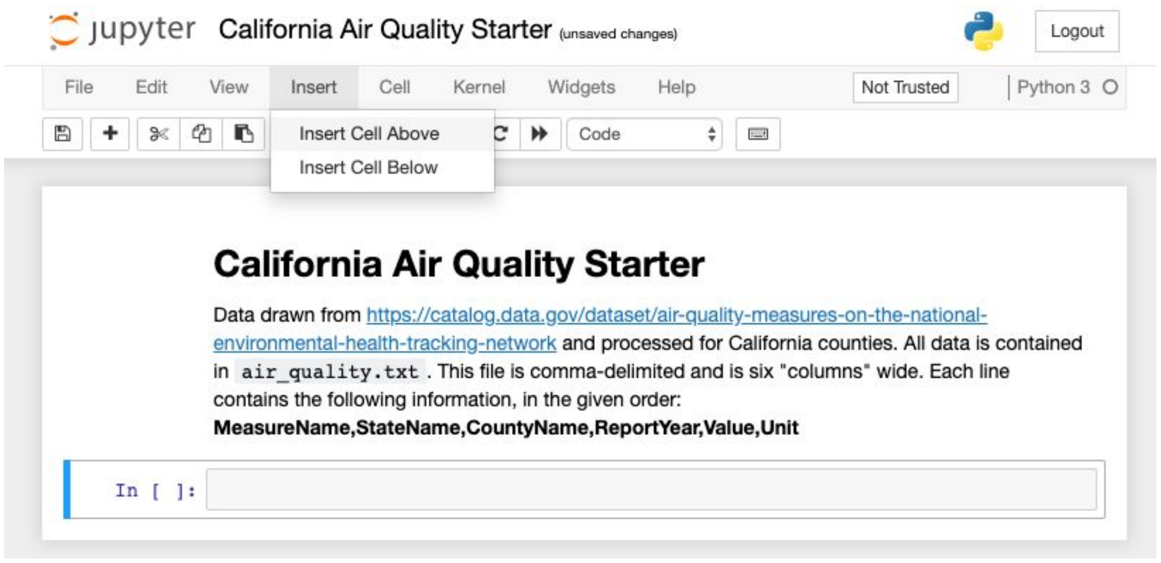
Troubleshooting

If you run into an issue where "**`jupyter`**" is not recognized as a command when trying to run **`jupyter notebook`** in your Terminal, try using the following command to launch Jupyter instead (replace **`python3`** with **`py`** if you're using a Windows machine):

```
$ python3 -m notebook
```

Editing and "running" a notebook

Jupyter notebooks consist of individual cells. Each cell can be either a text cell or a code cell. You can add a cell by first selecting a cell and then clicking on **Insert > Insert Cell Above/Below** in the top menu bar within a Python notebook (shown in Figure 2) or by clicking on the '**+**' button on the lower top menu bar (next to the save icon).


jupyter California Air Quality Starter (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3
Insert Cell Above
Insert Cell Below
Code

California Air Quality Starter

Data drawn from https://catalog.data.gov/dataset/air-quality-measures-on-the-national-environmental-health-tracking-network and processed for California counties. All data is contained in `air_quality.txt`. This file is comma-delimited and is six "columns" wide. Each line contains the following information, in the given order: **MeasureName,StateName,CountyName,ReportYear,Value,Unit**

```
In [ ]:
```

Figure 2: To add a cell, select the relevant option under the Insert tab in the top menu bar.

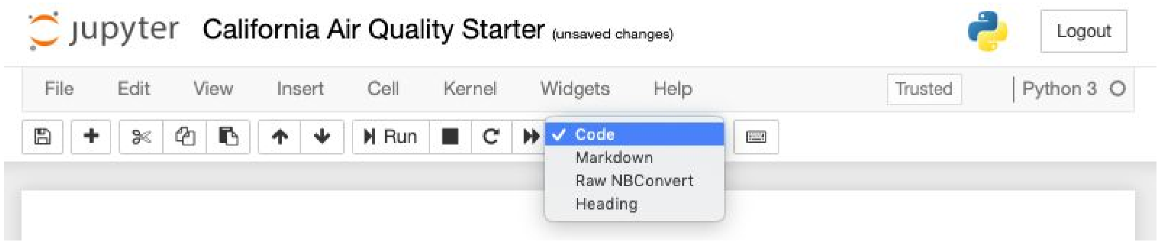
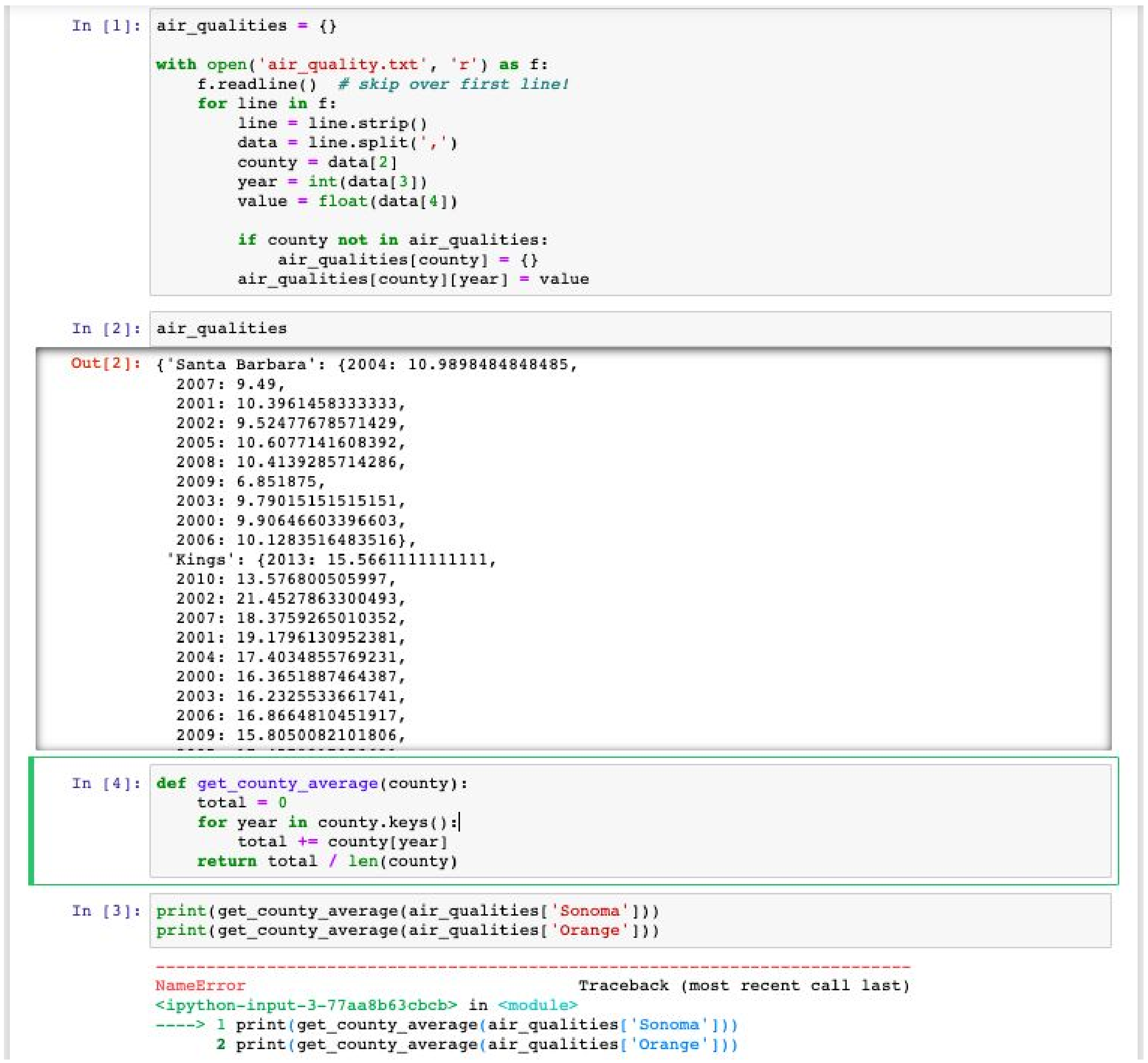

jupyter California Air Quality Starter (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
✓ Code
Markdown
Raw NBConvert
Heading

Figure 3: To change a cell's type, use the dropdown menu pictured above.

You can change a cell's type using the dropdown in the menu bar below the top menu, as pictured in Figure 3. For text cells select Markdown, and for code cells select code. Markdown is a language for literally "marking" up your text in order to format it. You can double-click on any text cells in the notebooks we've provided to you to get a sense for how the language works, and for a more extensive Markdown reference guide, we recommend checking out this page.

Text cells often provide instructions or descriptions relevant to the code cells around them. Adding informative text cells makes your notebook more readable and helps create a better experience for the person using your notebook. After inputting your text using Markdown (or after double-clicking on one of our provided text cells), you can "compile" it to show the formatting by hitting <**`Shift + Enter`**> while inside the cell.

Code cells are snippets of Python code that run line-by-line, just as normal Python code would. To execute (run) the code inside a cell, click on the cell and then hit <**`Shift + Enter`**>. While a cell is running, an asterisk (*) will appear in the square brackets to the left of the cell and after **`In`**. Once the cell is finished, a number will replace the asterisk and will indicate the order in which the cells have been run (as pictured in Figure 4).



```
In [1]: air_qualities = {}

        with open('air_quality.txt', 'r') as f:
            f.readline()  # skip over first line!
            for line in f:
                line = line.strip()
                data = line.split(',')
                county = data[2]
                year = int(data[3])
                value = float(data[4])

                if county not in air_qualities:
                    air_qualities[county] = {}
                air_qualities[county][year] = value

In [2]: air_qualities

Out[2]: {'Santa Barbara': {2004: 10.9898484848485,
          2007: 9.49,
          2001: 10.3961458333333,
          2002: 9.52477678571429,
          2005: 10.6077141608392,
          2008: 10.4139285714286,
          2009: 6.851875,
          2003: 9.79015151515151,
          2000: 9.90646603396603,
          2006: 10.1283516483516},
         'Kings': {2013: 15.5661111111111,
          2010: 13.576800505997,
          2002: 21.4527863300493,
          2007: 18.3759265010352,
          2001: 19.1796130952381,
          2004: 17.4034855769231,
          2000: 16.3651887464387,
          2003: 16.2325533661741,
          2006: 16.8664810451917,
          2009: 15.8050082101806,

In [4]: def get_county_average(county):
            total = 0
            for year in county.keys():
                total += county[year]
            return total / len(county)

In [3]: print(get_county_average(air_qualities['Sonoma']))
        print(get_county_average(air_qualities['Orange']))

        ---------------------------------------------------------------------------
        NameError                                 Traceback (most recent call last)
        <ipython-input-3-77aa8b63cbcb> in <module>
        ----> 1 print(get_county_average(air_qualities['Sonoma']))
              2 print(get_county_average(air_qualities['Orange']))
```

Figure 4: The numbers to the left of each cell indicate the order in which the user ran the cells. Note that the cells don't have to be run in top-to-bottom order (i.e. you can choose to

run any cell at any time). In the case pictured above, the bottom-most cell was run before the cell above it, which caused a **`NameError`** because **`get_county_average()`** was called before it was defined. To fix this issue, the user would have to run the cell that defines **`get_country_average()`** *before* running the last cell.

Note that when you run a code cell, Jupyter acts like the Python interpreter in how it displays the output of that cell. As a result, if you want to inspect the output of a single line or expression, you do not need to call **`print()`** on it.

Common pitfalls when using notebooks

Note that only the code inside the current cell will run when you hit <**`Shift + Enter`**>. This can be great for creating very modular notebooks, but it also means that if you edit code in an early cell and later cells also depend on that code, in order for the changes to take place you will need to rerun all of the cells in the correct order. In other words, if you define a function called **`foo()`** in an early cell and then edit that cell later, you'll need to rerun all of the cells that call **`foo()`** for the changes to have their full effect throughout your notebook and to correct any displayed output.

Furthermore, **you don't necessarily need to run code cells in linear order down the notebook**. This means that you'll need to be careful if you define variables or functions with the same names in multiple places because your code cells' outputs may change depending on the order in which you run the cells. And if you aren't careful about what order you run your cells, you may also end up with errors, like the one pictured in Figure 4. In particular, pay attention to the numbers in the square brackets next to each cell to ensure that you ran your code cells in the expected order!

Creating a notebook from scratch

To create a notebook from scratch, you should first run the command **`jupyter notebook`** inside the directory where you want to create the notebook. Then go to the "New" dropdown that is on the top right of the page and select "Python 3" (see Figure 5).

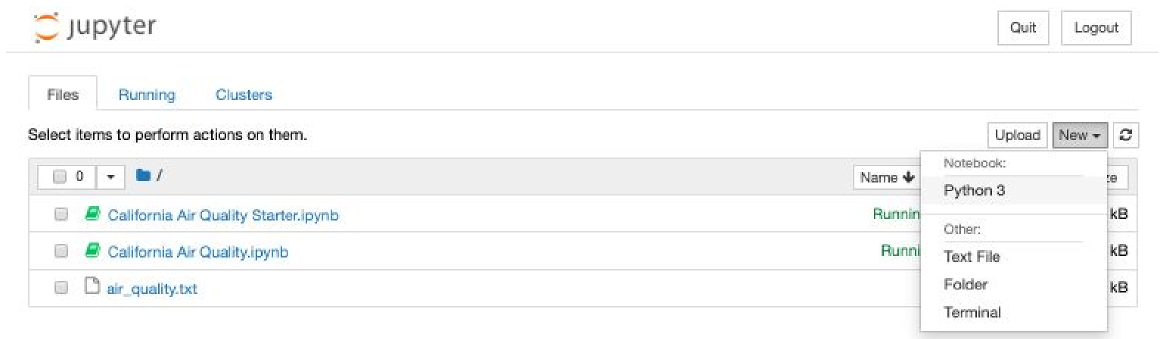


Figure 5: To create a new Python notebook, click on **New > Python 3** at the top right of the browser tab that opens after you run **`jupyter notebook`**.

Using matplotlib inside of a Jupyter notebook

matplotlib is a useful Python library for visualizing data and creating plots. To install matplotlib, run the following command inside your Terminal:

```
$ python3 -m pip install matplotlib
```

To use matplotlib (or any other library) inside a Jupyter notebook, you'll need to import the correct module, just like you would in a normal Python script. Put the following line at the top of any code cell, and make sure to execute the cell containing this line before running any other cells that use specific functions from the matplotlib library:

```
import matplotlib.pyplot as plt
```

For full documentation on the matplotlib library, visit the matplotlib website. The site includes tutorials, as well as sample plots with source code (example).