InfPALS

Data Science - Intro (30’)

Getting started (30 minutes)

Quick Overview

This session is based around a Python-based tool suite commonly referred to as scipy, consisting of the following popular packages: scipy (science!), numpy (numerical methods), pandas (data analysis), matplotlib (plotting, ikr), etc. All of these can be used in a normal python console, but we can use something slightly nicer: the Jupyter Notebook! Simply put, the Jupyter Notebook is a user interface which runs on your web browser and interacts with a local server (kernel) that does all the number crunching.

This is intended as a short, basic, linear intro to Jupyter Notebook, and since the next activities assume you’ve gone through this guide, make sure you’ve tried it before you go any further.

Installing (5 - 10 minutes)

We'll be setting up a virtual environment to hold all packages required for this session. To begin, open up a terminal window and:

1. Create a dedicated directory (something like `mkdir infpals-datasci` might work)
2. `$ python3 -m venv infpals-datasci` - creates the virtual environment

(make sure you have ~200 MB of free space for this, check with the command `freespace`)

quality of life note: when typing out directories, pressing Tab can autocomplete it for you

3. `$ source infpals-datasci/bin/activate`

(this line tells the terminal that we want to use our newly created python environment; *reasons why we'd want to do this are outlined in the footnote*[1])

Your terminal should now have a new extra to the left of your DICE username like so:

```
$ (infpals-datasci) [penguin]s1600000:
```

Good news! The terminal is now using our virtual environment: whatever we do modifies only this environment.

Note: if you close your terminal you'll have to go through this step again to use the virtual environment, but none of your installed packages will be lost.

Let's use pip (the default python package manager) to install jupyter, its dependencies, and few other bits and bobs:

```
$ pip install --upgrade pip
$ pip install jupyter numpy matplotlib pandas
```

You should see a bunch of progress bars loading: things are getting installed and momentarily you'll be creating a Jupyter Notebook. That's mostly it for scary command-line stuff!

While you wait, have a look at the next few sections with the session leaders.

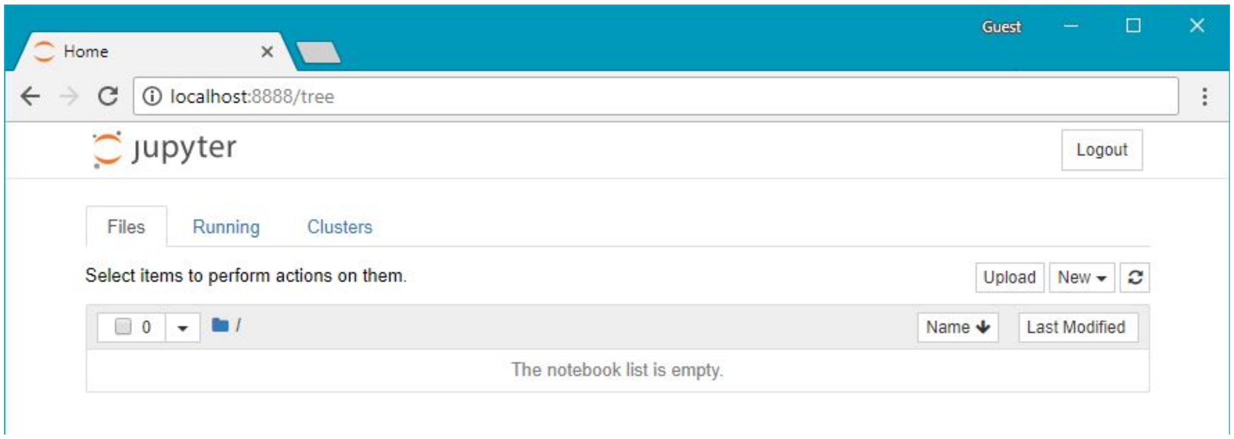
[1] *Small print: Why make our own virtual environment?*
The way DICE is set up means that programs such as python are installed in a central location accessible to every user, but which is not user modifiable (good luck cracking the sudo password). If we try to install packages using pip, it'll complain about us not having enough privileges to install to that central location. To get around this, we create a virtual environment: a way to manage our own installation of python with only the packages we need for our current project, separate from the rest of the system. This is also super useful if you want to use specific versions of packages and dependencies. Virtual environments are heavily used in production (including others like docker, nvm), where it is essential to keep track of exact versions of dependencies for a project.

Jupyter Notebook preamble (5 minutes)

Once you're done installing, you can start to tour jupyter notebook by creating a new directory to store the notebooks, and issuing the following command:

```
$ mkdir jupyter-stuff  # call it what you want
$ cd jupyter-stuff
$ jupyter notebook
```

This last command starts the notebook server in the current directory. It should also open up a browser window[2] (Chrome or Firefox depending on your DICE preferences). This is where we'll be working from now on.



Select items to perform actions on them.

The notebook list is empty.

Jupyter Notebook running

To start, click the [New] button on the right, and choose Notebook: [Python 3]. This should open a new tab that looks a little bit like Google Docs. A few short things to note:

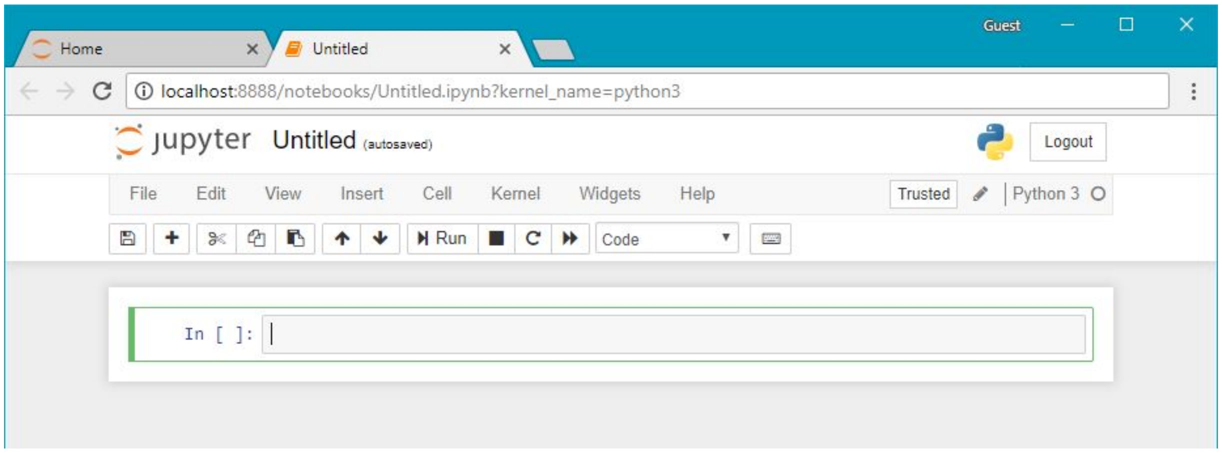
- Jupyter works with **cells**. Cells can hold code or markdown: the latter is intended for annotations. Each cell can be run independently, and as they run, they are "stacked" on top of each other.
- Each notebook has a **Kernel**. The Kernel's job is to run code that you input and return the output.

These two are the basics of the notebook; let's try them out!

[2] *If a browser window does not appear, it should have given you a message about not being able to find a browser, and a link with a large token (http://localhost:8888/?token=7a4...). Simply copy it into the address bar of your browser and you should be up and running*

Trying out Jupyter Notebook (5 - 10 minutes)

To start us off, let's throw some basic python at it and see what happens. The first cell in your notebook should currently be empty, and look something like this:



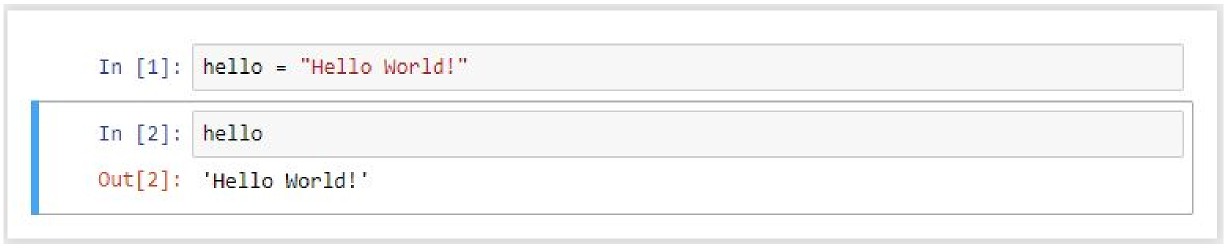
An empty notebook

Start by giving your notebook a name. Then, select the cell (the outline should turn green), and type:

```
print("Hello World!")
```

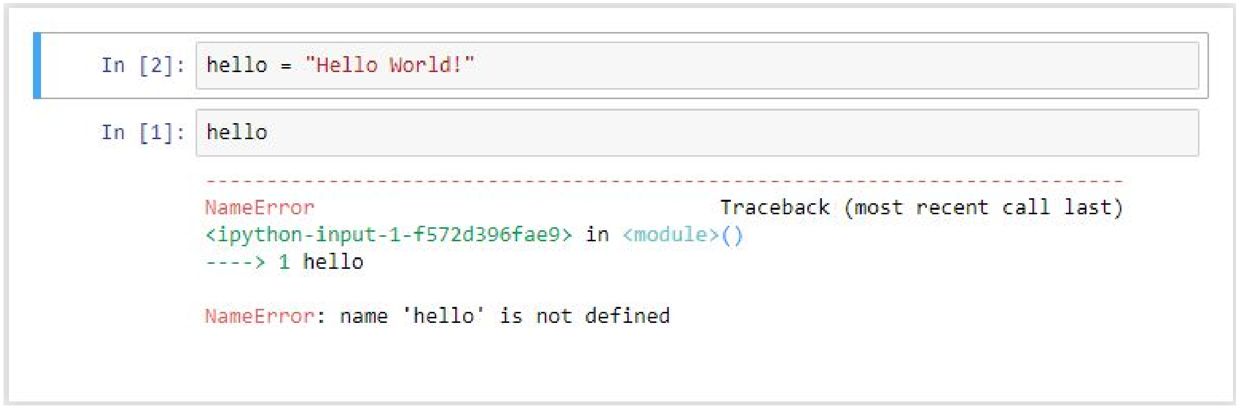
Notice that if you try to press `Enter`, the cell will just expand; this behaviour is intended. The way to run cells is to either press `Ctrl+Enter` (only runs the cell), or `Shift+Enter` (runs the cell and adds an empty cell underneath).

After running, the left part of the cell should now display `In [1]:` - the cell was executed, and the number represents the order in which it was executed. This is important because order matters here, since different parts of the notebook can be run independently. Take the following example:



```
In [1]: hello = "Hello World!"

In [2]: hello

Out[2]: 'Hello World!'
```

This order returns the correct result

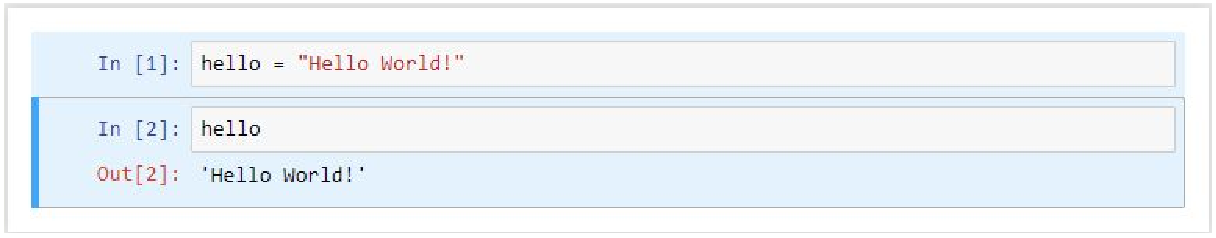


```
In [2]: hello = "Hello World!"

In [1]: hello

        ---------------------------------------------------------------------------
        NameError                                 Traceback (most recent call last)
        <ipython-input-1-f572d396fae9> in <module>()
        ----> 1 hello

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

This order fails because the variable had not yet been defined

Mostly, this happens when you go back and change an import, or modify a class or function further up in the notebook and forget to rerun the cell. It can be avoided by shifting cells up or down to ensure a good, linear flow through the notebook. Let's flesh our notebook out a little more to see what we can do.

Begin by clicking on the left of the top cell so that it has a blue outline like the figure above. Then, hold `Shift`, and select the last cell so that the background on both turns blue: they're selected!



```
In [1]: hello = "Hello World!"

In [2]: hello

Out[2]: 'Hello World!'
```

Both cells are now selected!

Now, do the unthinkable: click [Edit] ➔ [Delete Cells]. Poof! They're gone. Unless you click [Undo Delete Cells]… I'm sure this will come in handy one day. While you're at it, click [Kernel] ➔ [Restart] to reset cell numbers and "flush" all variables. This gives you a clean slate to start executing.

You should be back to staring at an empty cell. Let's import something:

```
import math
```

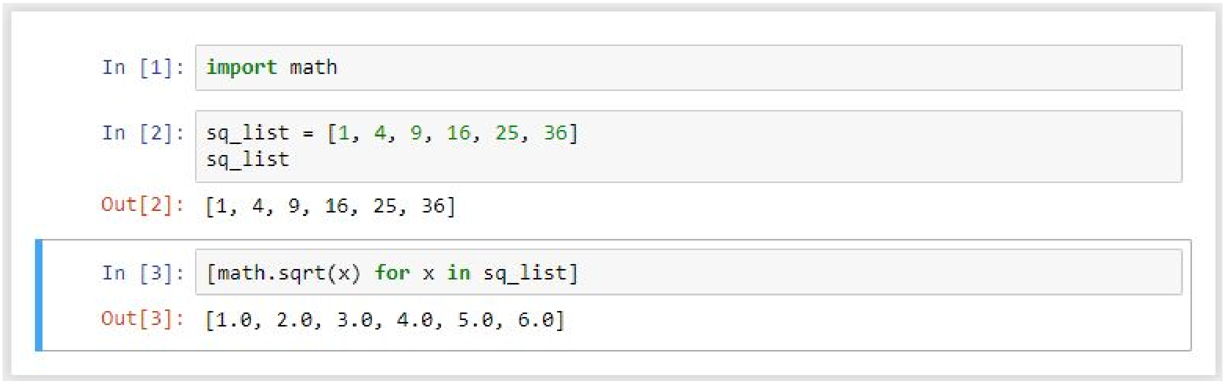
Keep this import on the first cell. Now on the cell below type:

```
sq_list = [1, 4, 9, 16, 25, 36]
sq_list
```

Make sure you remember to run it. You may also have noticed that sometimes there's an "`Out[]`" associated with a cell - this happens if the last line contains something which returns and output - in this case, it will print `sq_list`. Next, type:

```
[math.sqrt(x) for x in sq_list]
```

The above is a list comprehension for square rooting each element of `sq_list`. Note here how we make use of the property from just before, and obtain an output without needing to assign it to a variable. Your notebook should look like this:



```
In [1]: import math

In [2]: sq_list = [1, 4, 9, 16, 25, 36]
        sq_list

Out[2]: [1, 4, 9, 16, 25, 36]

In [3]: [math.sqrt(x) for x in sq_list]

Out[3]: [1.0, 2.0, 3.0, 4.0, 5.0, 6.0]
```

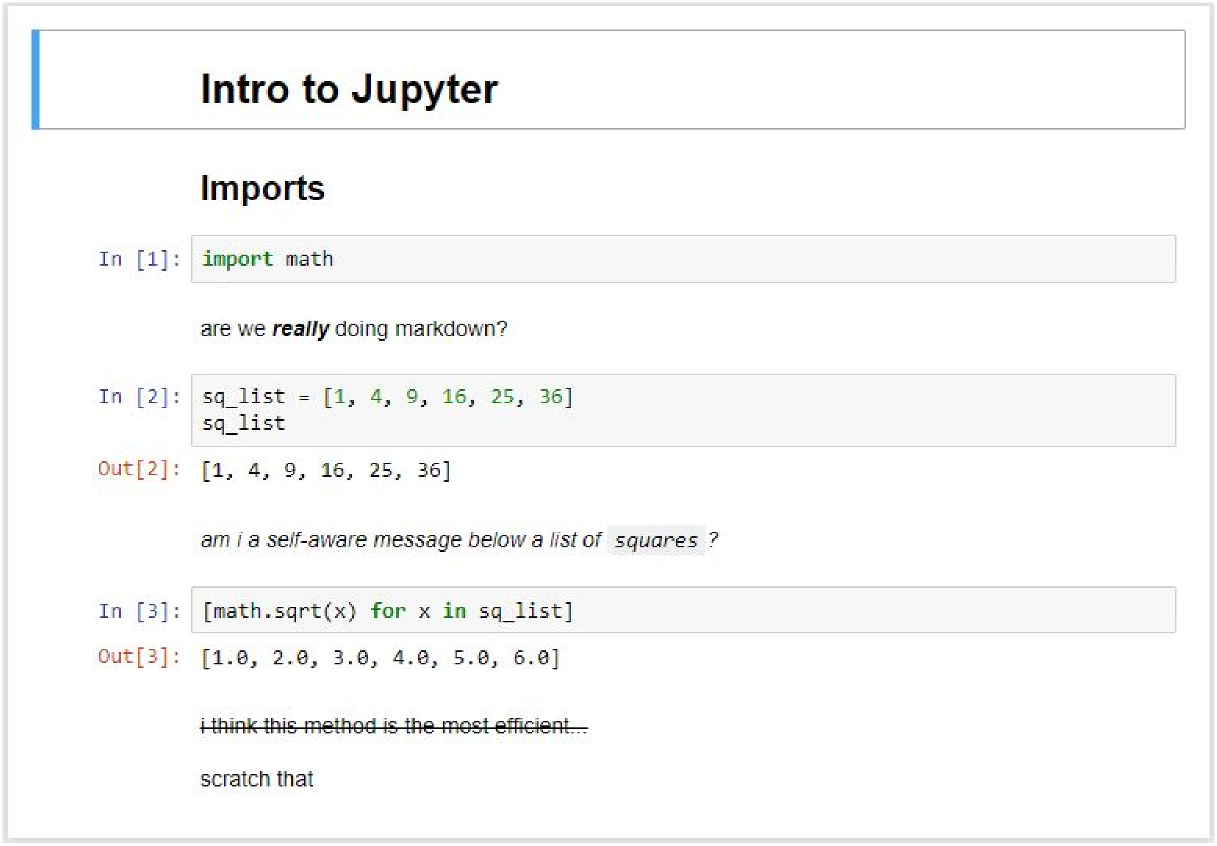
A few example cells

With all the cells we've filled in, we should probably save this somewhere… Press `Ctrl+S` to create a checkpoint, and all your progress will be saved. That simple, eh.

Welcome to Markdown land (5 minutes)

Let's do some annotating! Select cell 1, then [Insert] ➜ [Insert Cell Above], select the new empty cell, then the dropdown that says [Code] and switch to [Markdown]. Annotate to your heart's content. Cheatsheet included: github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet

Aim to get something descriptive going, plus points for creativity:



Intro to Jupyter

Imports

```
In [1]: import math
```

are we ***really*** doing markdown?

```
In [2]: sq_list = [1, 4, 9, 16, 25, 36]
        sq_list
Out[2]: [1, 4, 9, 16, 25, 36]
```

am i a self-aware message below a list of `squares`?

```
In [3]: [math.sqrt(x) for x in sq_list]
Out[3]: [1.0, 2.0, 3.0, 4.0, 5.0, 6.0]
```

~~i think this method is the most efficient...~~

scratch that

Octocat was here

Use markdown to help organise your notebook and keep track of your train of thought through the notebook (or to help others understand if they are looking through your notebook). Again, keep it intelligible and descriptive.

Bonus (are you not entertained?)

At this point you may have already completed the short exercise. That was fast... If your session leaders have not yet moved onto the next activity, and while you wait for everyone else to catch up and get ready for the next activity, download the `short-mandelbrot.ipynb` notebook from

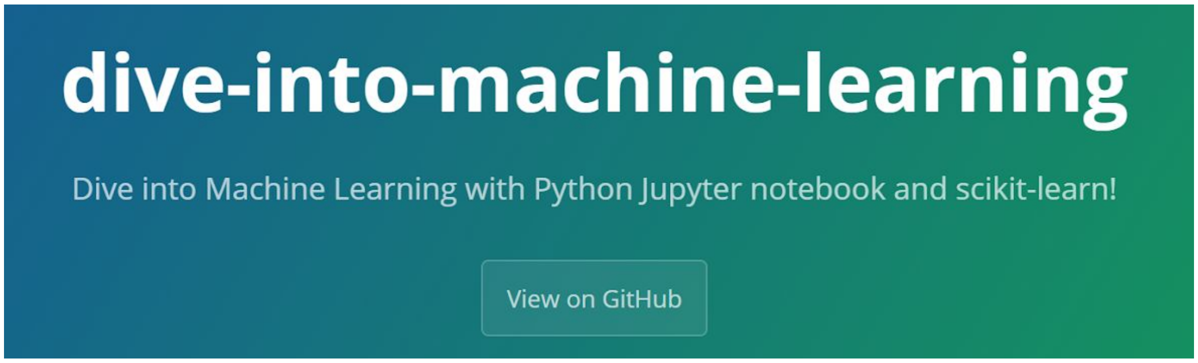
notebooks.azure.com/jmaio/libraries/random

Try it out by either running each cell at a time, or click [Cells] ➔ [Run All]. Look through the code, maybe mess around a little, see how much you can break it.

On the last cell should be a pretty picture with a slider underneath. You know what sliders do. I'll leave you to it.

Bonus Bonus

Here's someone else's take on an intro to ML by using Jupyter just like we're doing:


dive-into-machine-learning
Dive into Machine Learning with Python Jupyter notebook and scikit-learn!
View on GitHub

hangtwenty.github.io/dive-into-machine-learning/

If you complete all the activities, have a look through it at the end of the session. Although it contains more info on ML specifics aka how to use Jupyter for classification problems, etc., it has good general resources.