00a_intro

February 4, 2019

1 Geographic Data Science - Lab 01

Dani Arribas-Bel

```
In [1]: from IPython.display import Image, IFrame
```

2 Computational tools for Geographic Data Science

In this tutorial we will introduce the main tools we will be working with throughout the rest of the course. Although very basic and seemingly abstract, everything showed here will become the basis on top of which we will build more sophisticated (and fun) tasks. But, before, let us get to know the tools that will give us data super-powers.

2.1 Open Source

This course will introduce you to a series of computational tools that make the life of the Data Scientist possible, and much easier. All of them are open-source, which means the creators of these pieces of software have made available the source code for people to use it, study it, modify it, and re-distribute it. This has allowed a large eco-system that today represents the best option for scientific computing, and is used widely both in industry and academia. Thanks to this, this course can be taught with entirely freely available tools that you can install in any of your computers.

If you want to learn more about open-source and free software, here are a few links:

- **[Video]**: brief explanation of open source.
- **[Book]** The Cathedral and the Bazaar: classic book, freely available, that documents the benefits and history of open-source software.

2.2 `Jupyter` Lab

The main computational tool you will be using during this course is Jupyter Lab. The Lab is an app to interact with scientific computing, mainly through the so-called Jupyter notebooks. Notebooks are a convenient way to thread text, code and the output it produces in a simple file that you can then share, edit and modify. You can think of notebooks as the Word document of Data Scientists.

A very good account of the history and philosophy behind computational notebooks can be found at this "The Atlantic" article:

```
In [2]: IFrame('https://www.theatlantic.com/science/archive/2018/04/the-scientific-paper-is-obsc
                800, 600)
```

```
Out[2]: <IPython.lib.display.IFrame at 0x7f2c240724a8>
```

2.2.1 Cells

The main building block of notebooks are cells. These are chunks of the same time of content which can be cut, pasted, and moved around in a notebook. Cells can be of two types:

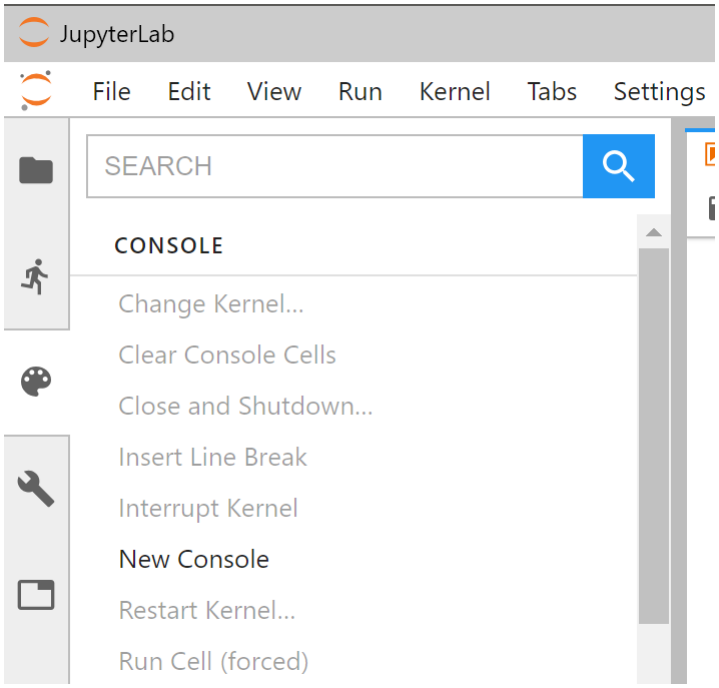
- **Text**, like the one where this is written.
- **Code**, like the following one below:

```
In [3]: # This is a code cell
```

The notebook allows to run several commands through the "Command Palette", which can be found on the third tab of the left pane:

```
In [4]: Image('../figs/lab01_command_palette.png')
```

Out[4]:



For example, you can create a new cell by searching for "Insert Cell". By default, this will be a code cell, but you can change that on the `Cell -> Cell Type` menu. Choose `Markdown` for a text cell. Once a new cell is created, you can edit it by clicking on it, which will create the cursor bar inside for you to start typing.

Pro tip!: cells can also be created with shortcuts. If you press `<escape>` and then `b` (a), a new cell will be created below (above). There is a whole bunch of shortcuts you can explore by pressing `<escape>` and `h` (press `<escape>` again to leave the help).

2.2.2 Code and its output

A particularly useful feature of notebooks is that you can save, in the same place, the code you use to generate any output (tables, figures, etc.). As an example, the cell below contains a snipet of Python that returns a printed statement. This statement is then printed below and recorded in the notebook as output:

```
In [5]: print("Hello world!!!")

Hello world!!!
```

Note also how the notebook has automatic syntax highlighting support for Python. This makes the code much more readable and understandable. More on Python below.

2.2.3 Markdown

Text cells in a notebook use the Github Flavored Markdown markup language. This means you can write plain text with some rules and the notebook renders a more visually appealing version of it. Let's see some examples:

- **BOLD**:

  `This is **bold**.`
  Is rendered:
  This is **bold**.

- **ITALIC**:

  `This is *italic*.`
  Is rendered:
  This is *italic*.

- **LISTS**:

  You can create unnumbered lists:

```
* Item 1
* Item 2
* ...
```

  Which will produce:

  - Item 1
  - Item 2
  - ...

  Or you can create numbered lists:

```
1. First element
1. Second element
1. ...
```

And get:

1. First element
2. Second element
3. ...

Note that you don't have to write the actual number of the element, just using `1.` always produces a numbered list.

You can also nest lists:

```
* First unnumbered element, which can be split into:

    1. One numbered element
    2. Another numbered element

* Second element.
* ...
```

- First unnumbered element, which can be split into:

    1. One numbered element
    2. Another numbered element

- Second element.
- ...

This creates many oportunities to combine things nicely.

- **LINKS**

`You can easily create hyperlinks, for example to [WikiPedia](https://www.wikipedia.org/).`
You can easily create hyperlinks, for example to WikiPedia.

- **HEADINGS**: including # before a line causes it to render a heading.

```
# This is Header 1
```
Turns into:

3 This is Header 1

```
## This is Header 2
```
Turns into:

3.1 This is Header 2

```
### This is Header 3
```

Turns into:

3.1.1 This is Header 3

And so on...

You can see a more in detail introduction in the following links:

https://help.github.com/articles/markdown-basics/

https://help.github.com/articles/github-flavored-markdown/

3.1.2 Rich content in a notebook

Notebooks can also include rich content from the web. For that, we need to import the `display` module:

```
In [6]: import IPython.display as display
```

This makes available additional functionality that allows us to embed rich content. For example, we can include a YouTube clip easily by passing it's ID:

```
In [7]: display.YouTubeVideo('iinQDhsdE9s')
```

```
Out[7]:
```

Or we can pass standard HTML code:

```
In [8]: display.HTML("""<table>
        <tr>
        <th>Header 1</th>
        <th>Header 2</th>
        </tr>
        <tr>
        <td>row 1, cell 1</td>
        <td>row 1, cell 2</td>
        </tr>
        <tr>
        <td>row 2, cell 1</td>
        <td>row 2, cell 2</td>
        </tr>
        </table>""")

Out[8]: <IPython.core.display.HTML object>
```

Note that this opens the door for including a large number of elements from the web, as an `iframe` is also allowed. For example, interactive maps can be included:

```
In [9]: osm = """
        <iframe width="425" height="350" frameborder="0" scrolling="no" marginheight="0" marginw
        """
        display.HTML(osm)

Out[9]: <IPython.core.display.HTML object>
```

Or sound content:

```
In [10]: sound = '''
         <iframe width="100%" height="300" scrolling="no" frameborder="no" allow="autoplay" src=
         display.HTML(sound)

Out[10]: <IPython.core.display.HTML object>
```

A more thorough exploration of them is available in this notebook.

3.1.3 Exercise to work on your own

Try to reproduce, using markdown and the different tools the notebook affords you, the following WikiPedia entry:

https://en.wikipedia.org/wiki/Chocolate_chip_cookie_dough_ice_cream

```
In [11]: display.IFrame('https://en.wikipedia.org/wiki/Chocolate_chip_cookie_dough_ice_cream',
                        700, 500)

Out[11]: <IPython.lib.display.IFrame at 0x7f2c24017898>
```

Do not over think it. Focuse and pay special attention to getting the bold, italics, links, headlines and lists correctly formated, but don't worry too much about the overall layout. Bonus if you manage to insert the image as well (it does not need to be properly placed as in the original page)!

3.2 Python

The main bulk of the course relies on the Python programming language. Python is a high-level programming language widely used today. To give a couple of examples of its relevance, it is underlying most of the Dropbox systems, but also heavily used to control satellites at NASA. A great deal of Science is also done in Python, from research in astronomy at UC Berkley, to courses in economics by Nobel Prize Professors.

This course uses Python because it has emerged as one of the main and most solid options for Data Science, together with other free alternatives such as R. Python is widely used for data processing and analysis both in academia and in industry. There is a vibrant and growing scientific community (example and example), working at both universities and companies, that supports and enhances its capabilities for data analysis by providing new and refining existing extensions (a.ka.a. libraries, see below). In the geospatial world, Python is also very widely adopted, being the selected language for scripting in both ArcGIS and QGIS. All of this means that, whether you are thinking of continuing in Higher Education or trying to find a job in industry, Python will be an importan asset that employers will significantly value.

Being a high-level language means that the code can be "dynamically interpreted", which means it is run on-the-fly without the need to be compiled. This is in contrast to "low-level" programming languages, which first need to be converted into machine code (i.e. compiled) before they can be run. With Python, one does not need to worry about compilation and can just write code, evaluate, fix it, re-evaluate it, etc. in a quick cycle, making it a very productive tool. The rest of this tutorial covers some of the basic elements of the language, from conventions like how to comment your code, to the basic data structures available.

The rest of the tutorial is partly inspired by the introductory lesson in this course by Lorena Barba's group.

3.2.1 Python libraries

The standard Python language includes some data structures (e.g. lists, dictionaries, etc. See below) and allows many basic operations (e.g. sum, product, etc.). For example, right out of the box, and without any further action needed, you can use Python as a calculator:

```
In [12]: 3 + 5

Out[12]: 8

In [13]: 2. / 3

Out[13]: 0.6666666666666666

In [14]: (3 + 5) * 2. / 3

Out[14]: 5.333333333333333
```

However, the strength of Python as a data analysis tool comes from the extensions provided separately that add functionality and provide access to much more sophisticated data structures and functions. These come in the form of packages, or libraries, that once installed need to be imported into a session.

In this course, we will be using many of the core libraries of what has been called the "PyData stack", the set of libraries that make Python a full-fledge system for Data Science. We will introduce them gradually as we need them for particular tasks but, for now, let us have a look at the foundational library, numpy (short for numerical Python). Importing it is simple:

```
In [15]: import numpy as np # we rename it in the session as `np` by convention
```

Note how we import it *and* rename it in the session, from `numpy` to `np`, which is shorter and more convenient.

Note also how comments work in Python: everything in a line *after* the `#` sign is ignored by Python when it evaluates the code. This allows you to insert comments that Python will ignore but that can help you and others better understand the code.

Once imports are out of the way, let us start exploring what we can do with `numpy`. One of the easiest tasks is to create a sequence of numbers:

```
In [16]: seq = np.arange(10)
         seq
```

```
Out[16]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

The first thing to note is that, in line 1, we create the sequence by calling the function `arange` and assign it to an object called `seq` (it could have been called anything else, pick your favorite) and, in line 2, we have it printed as the output of the cell.

Another interesting feature is how, since we are calling a `numpy` function called `arange` by adding `np.` in front. This is to note that the function comes explicitly from `numpy`. To find out how necessary this is, you can try generating the sequence without `np`:

```
In [17]: # NOTE: comment out to run the cell
         #seq = arange(10)
```

What you get instead is an error, also called a "traceback". In particular, Python is telling that it cannot find a function named `arange` in the core library. This is because that particular function is only available in `numpy`.

3.2.2 Variables

A basic feature of Python is the ability to assign a name to different "things", or objects. These can also be called sometimes "variables". We have already seen that in the example above but, to make it more explicit, let us make it even simpler. For example, an object can be a single number:

```
In [18]: a = 3
```

Or a name, also called "string":

```
In [19]: b = 'Hello World'
```

You can check what type an object is also easily:

```
In [20]: type(a)

Out[20]: int
```

`int` is short for "integer" which, roughly speaking, means an whole number. If you want to save a number with decimals, you will be using floats:

```
In [21]: c = 1.5
         type(c)

Out[21]: float
```

As mentioned, what we understand as letters in a wide sense (spaces and other signs count too) is called "strings" (`str` in short):

```
In [22]: type(b)

Out[22]: str
```