Data Science and Machine Learning for Engineering Applications

Python installation, Anaconda-Navigator, and Jupyter notebook: beginner's tutorial

March 8, 2023 - Politecnico di Torino

Introduction

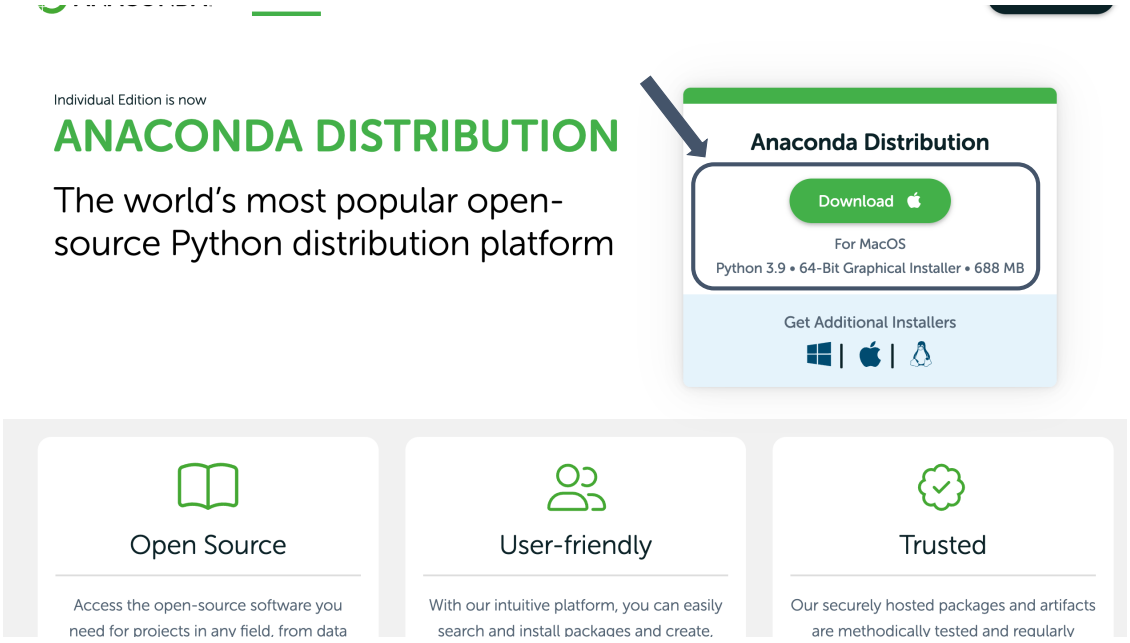
This tutorial will show you how to: i) install Python with Anaconda-Navigator (Section 1); ii) manage virtual environments with Anaconda (Section 2); iii) install python packages (Section 3); iv) use Jupyter Notebook (Section 4).

1 Install Anaconda-Navigator

Anaconda Navigator is a desktop GUI (Graphical User Interface) allowing you to **launch applications** and **manage conda packages and environments** without command-line commands. It includes a GUI, Anaconda Navigator, as a graphical alternative to the command line interface. Navigator can search for packages, install them in an environment, run the packages, and update them. The Anaconda guide can be found at the following URL: https://docs.anaconda.com/anaconda/user-guide/getting-started/.

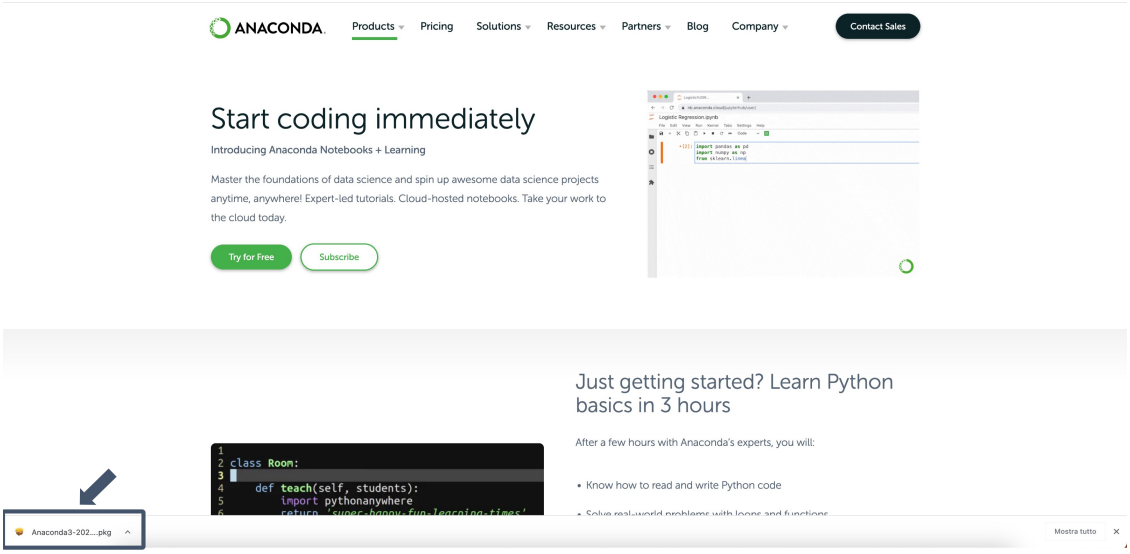
1.1 Download Anaconda-Navigator

From the Anaconda website at the following URL: https://www.anaconda.com/products/distribution, download the installation files for your operating system (i.e., MacOS, Linux, or Windows). Install the latest version of python with Anconda-Navigator. In this case, python 3.9.



1.2 Install Anaconda-Navigator

When the download is finished, double-click on the downloaded file in the bottom left-hand corner of your browser. This will start the installation of Anaconda-Navigator. The installation process depends on your operating system.



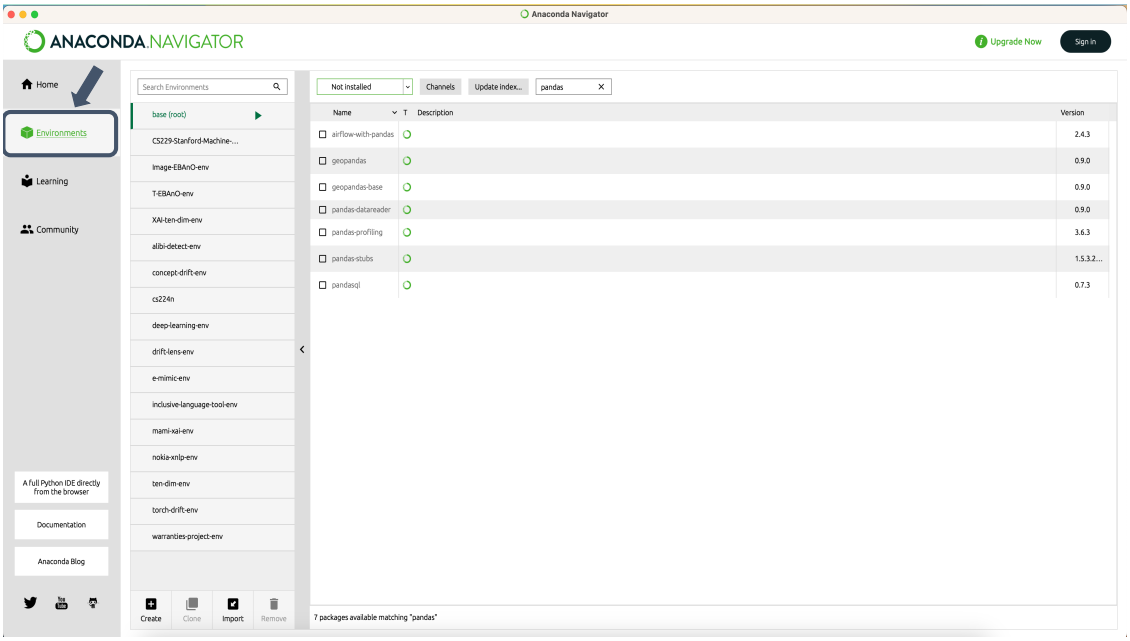
2 Create a virtual environment with Anaconda-Navigator

Python requires a different version for different kinds of applications. The application needs to run on a specific language version because it requires certain dependencies that are present in older versions but change in newer versions. **Virtual environments** make it easy to separate different applications and avoid problems with different dependencies [4]. Multiple ways of creating an environment include **virtualenv**, **venv**, and **conda**. However, the **conda** command is the preferred interface for managing installations and virtual environments with the Anaconda Python distribution.

This section shows how to create a virtual environment with Anaconda-Navigator, by exploiting the GUI (without the command line). If you want to learn more about creating a virtual environment with conda entirely with the command line, you can read more on this URL: `https://towardsdatascience.com/manage-your-python-virtual-environment-with-conda-a0d2934d5195`. This last option can be useful to run complex python projects on a remote server where the GUI is not available. However, it is not required for this course.

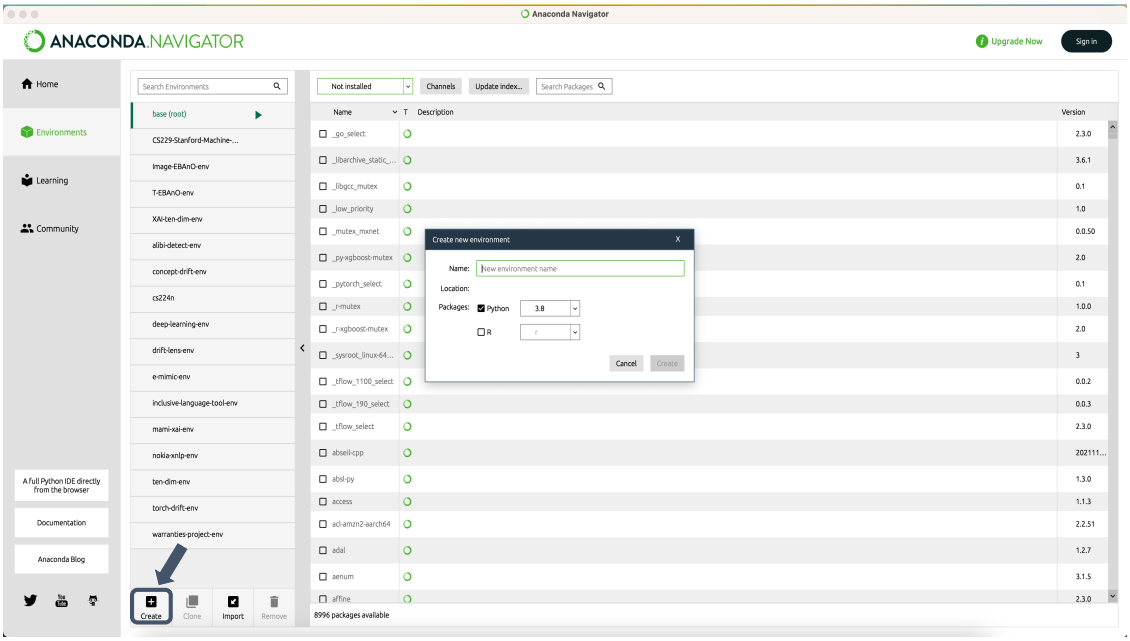
2.1 Select the environments

Click on the `"Environments"` button from the left menu. It will show the **list of all your environments**.



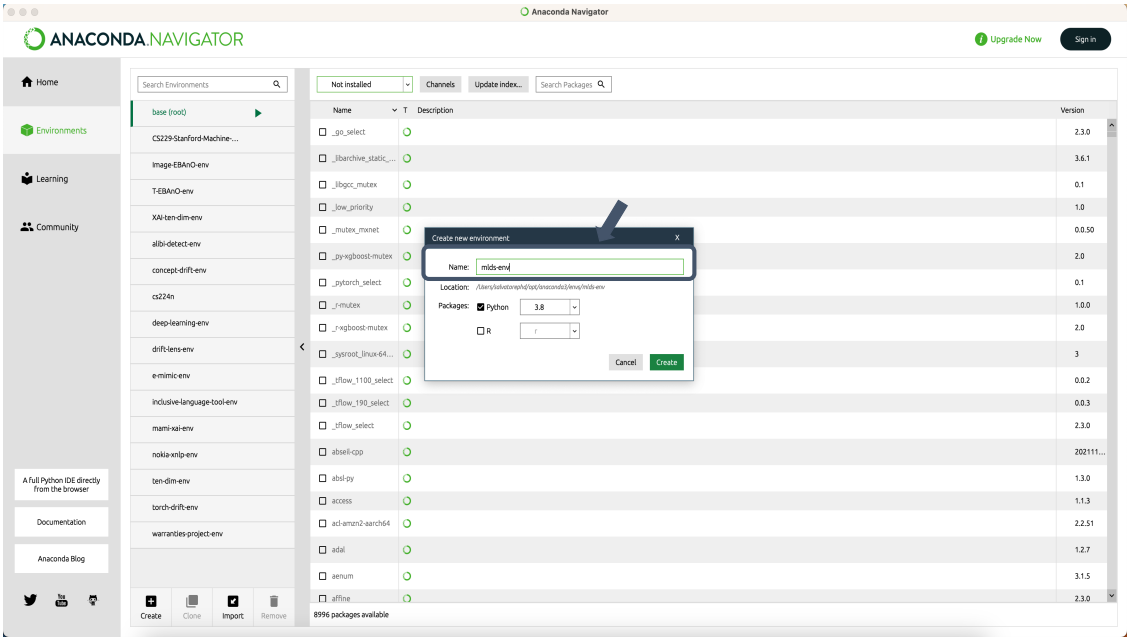
2.2 Create a virtual environment

Click the `"Create"` button in the bottom left-hand corner to create a new virtual environment.



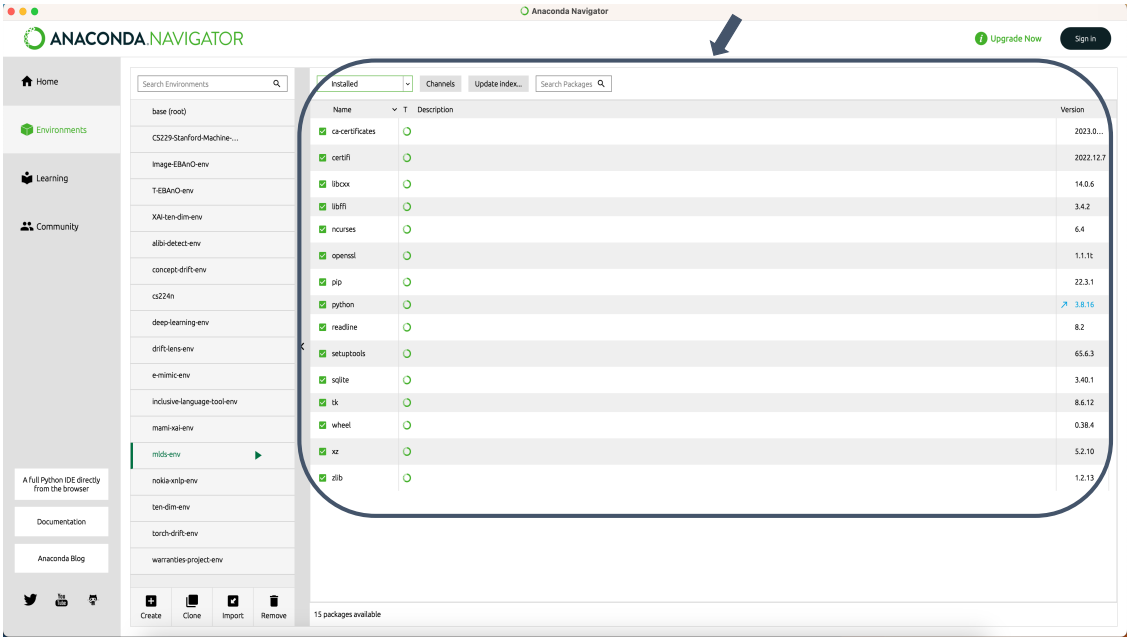
2.3 Choose a new name for your virtual environment

You have to specify the **environment name** and the **Python version**. Then, click the `"Create"` button.



2.4 Check the installed packages

Once created a new environment, the list of all **installed packages** in that environment will be shown. Notice that some packages are already installed.



3 Packages

To **install a new package** in the virtual environment, you have two options:

- Using the **Anaconda-Navigator GUI** directly (Section 3.1).
- Using the **command line** with the **conda** or **pip** commands (Section 3.2).

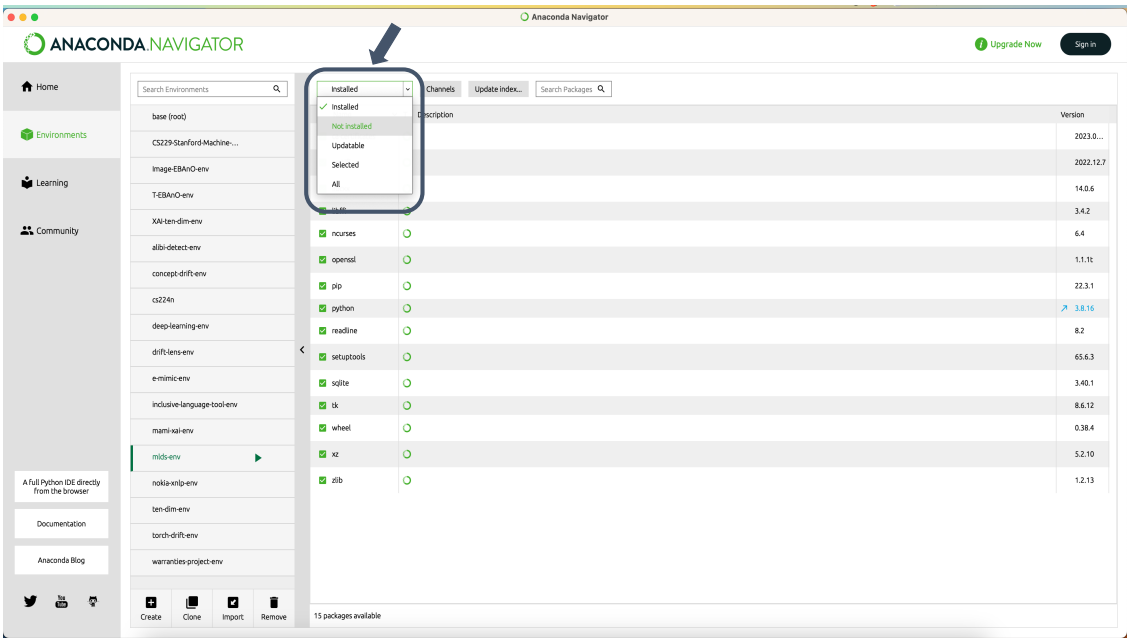
The main difference between **conda** and the **pip** package manager is how the package dependencies are managed. When **pip** installs a package, it also automatically installs any dependent Python packages without checking if these conflict with previously installed packages. Therefore, it will install a package and any of its dependencies regardless of the state of the existing installation. In contrast, **conda** analyzes the current environment, including everything currently installed and any version limitations specified. It works out how to install a compatible set of dependencies and shows a warning if this cannot be done [5]. Using the **Anaconda-Navigator GUI** to install a package will exploit the **conda** package manager. You can learn more about the differences between **conda** and **pip** at the following URL: https://www.anaconda.com/blog/understanding-conda-and-pip.

3.1 Install a package with the navigator GUI

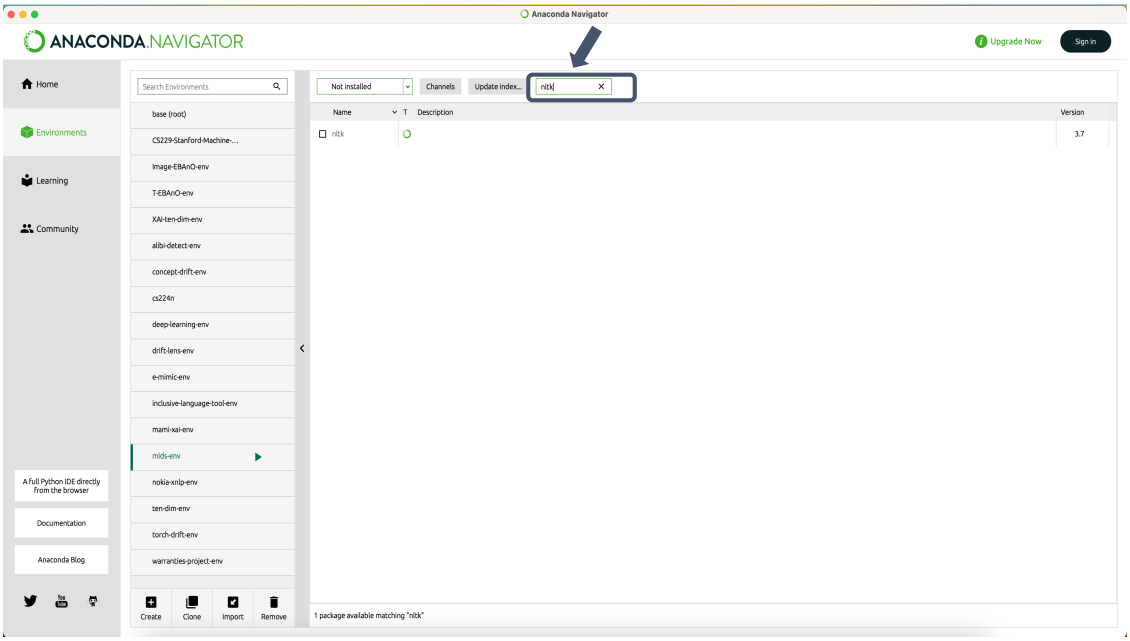
Installing any package through **Anaconda-Navigator GUI** is straightforward. You have to search for the required package, select a package, and click on `"Apply"` to install it

3.1.1 Search the required package

Select the option `"Not Installed"` in the top-center menu.

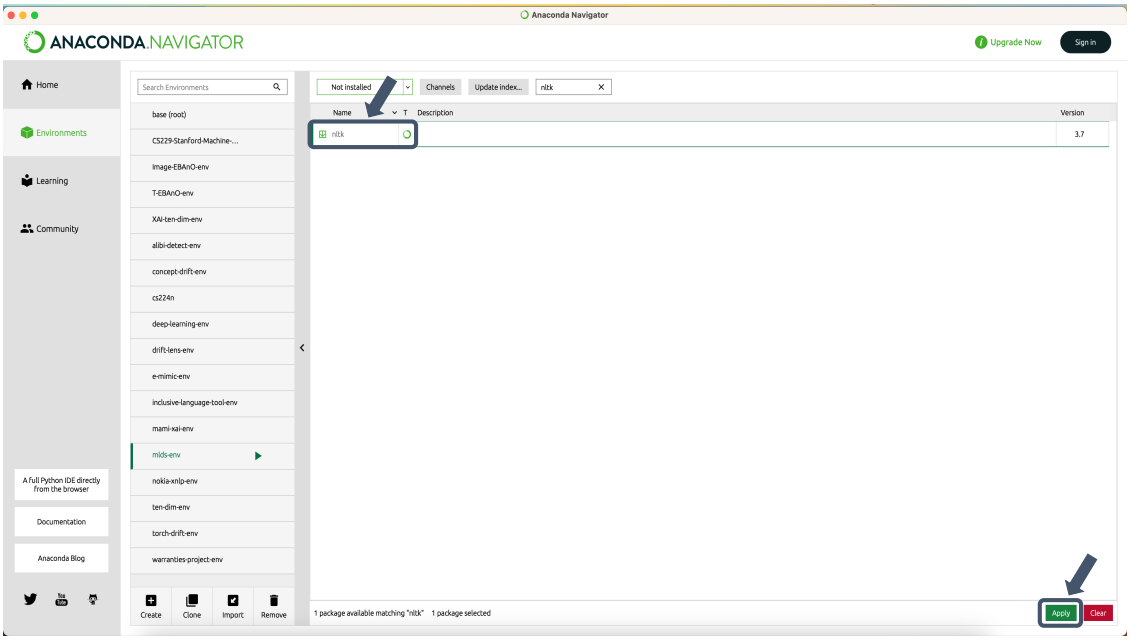


Then search for the package that you want to install by typing the name in the `textbox` (e.g., in this case, NLTK).

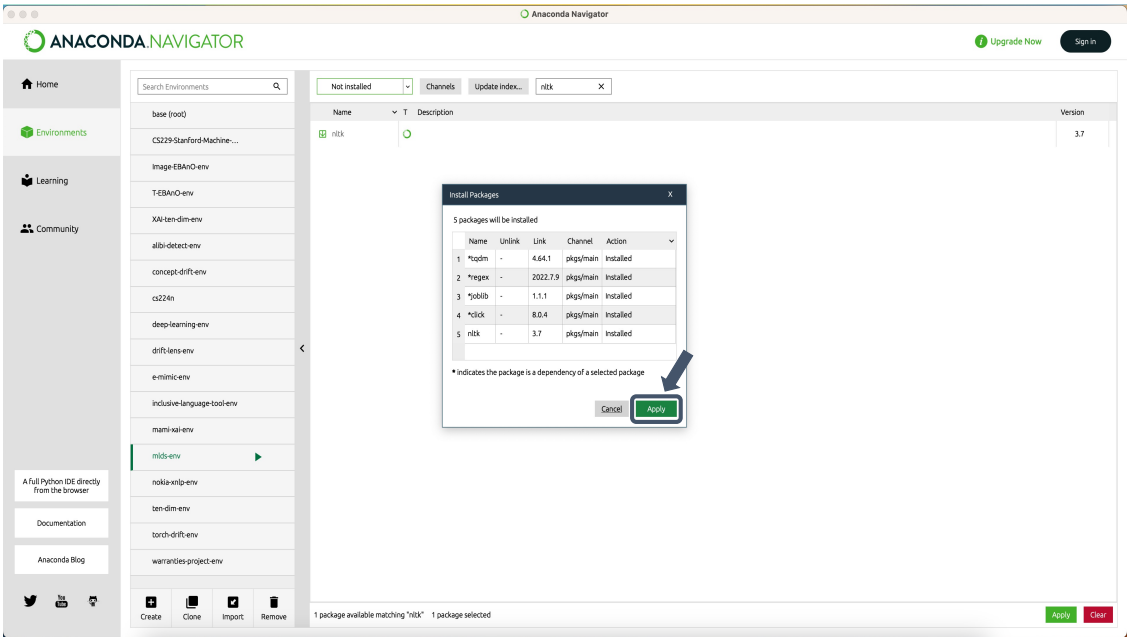


3.1.2 Select and install the required package

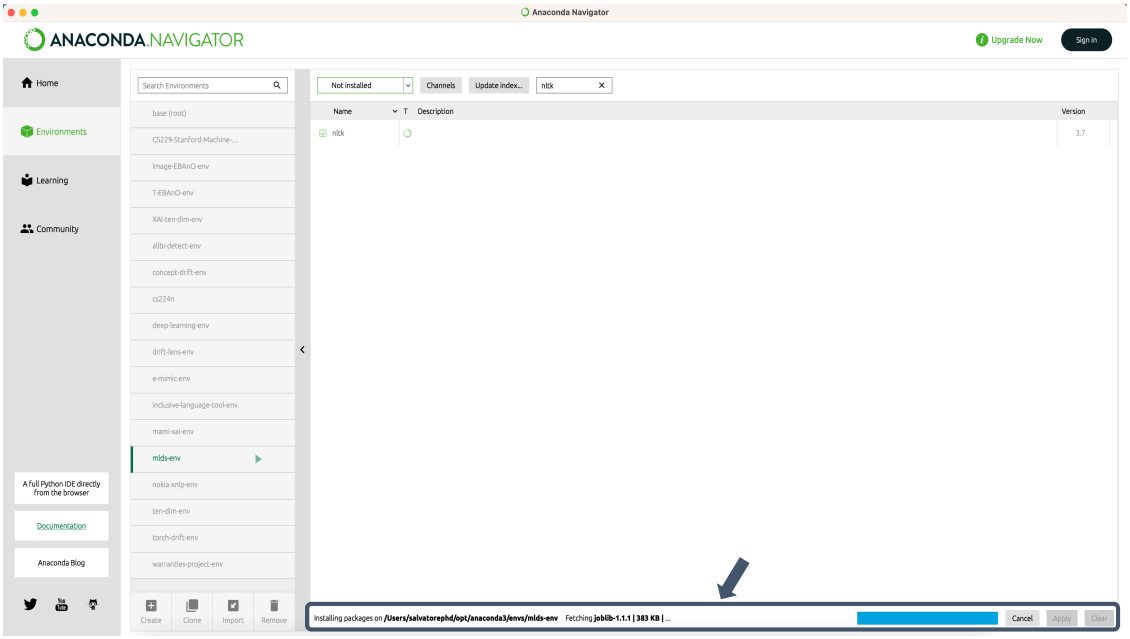
The **Anaconda-Navigator** will search in the **conda repository** for all the **conda packages** matching the typed name. Then, **select** the wanted package line and click on the `"Apply"` button in the right-hand bottom corner.



It will open a new window with all the **dependencies** for that package. The **conda package manager** will install **all** the dependencies for you. Click the `"Apply"` button to start the package installation.

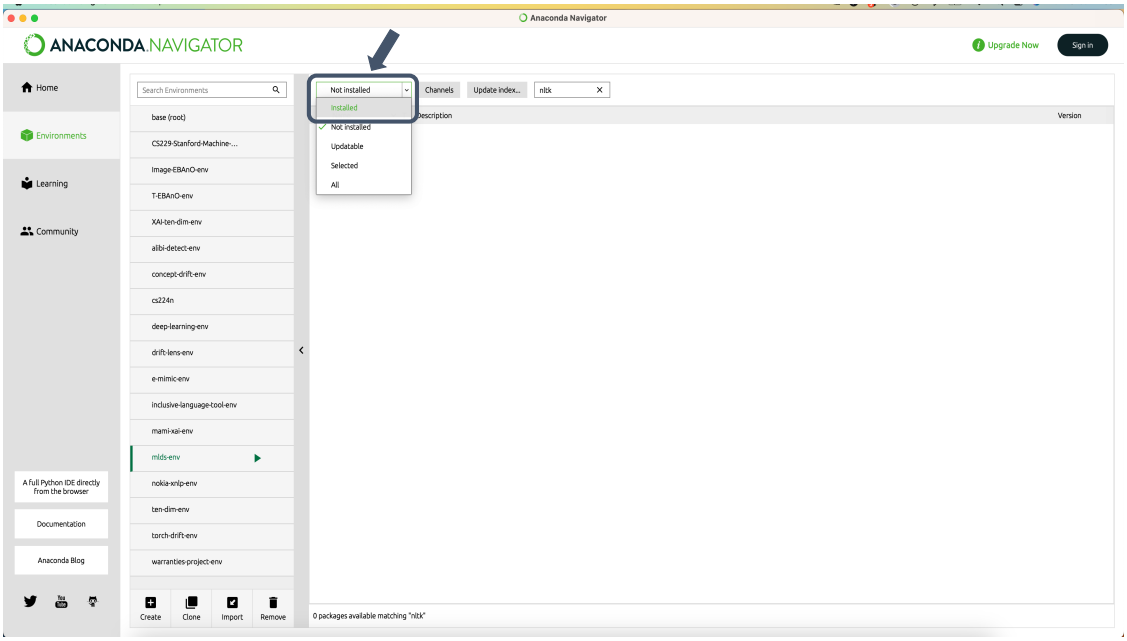


Wait for the download and installation. It could take some minutes.

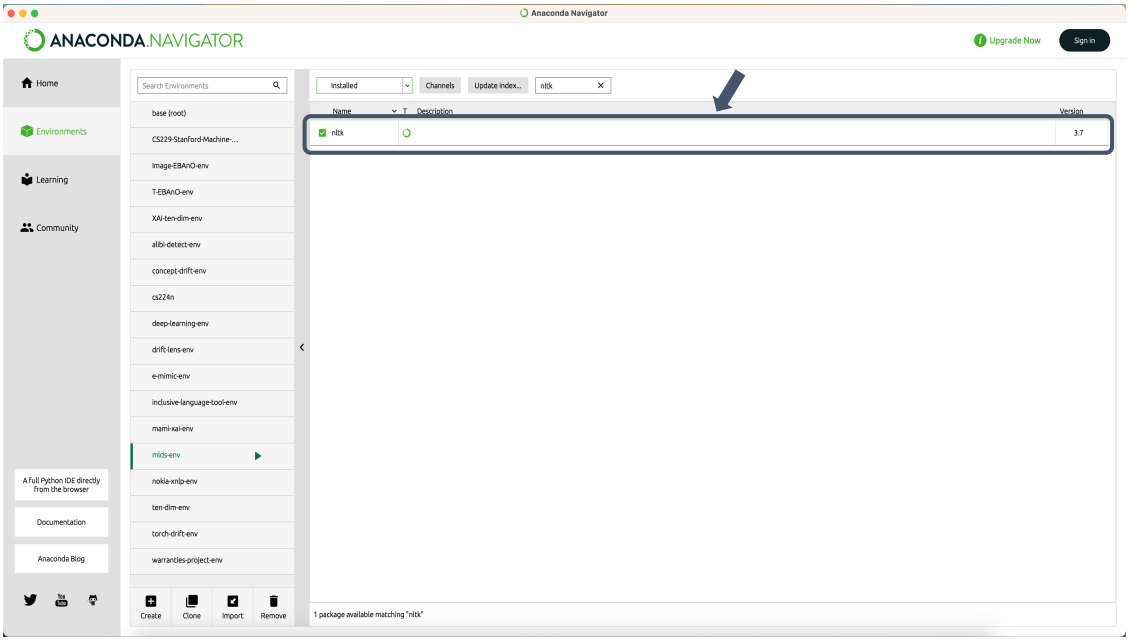


3.1.3 Check the installed package

You can check if the package has been correctly installed by selecting the `"Installed"` selection in the drop-down menu.

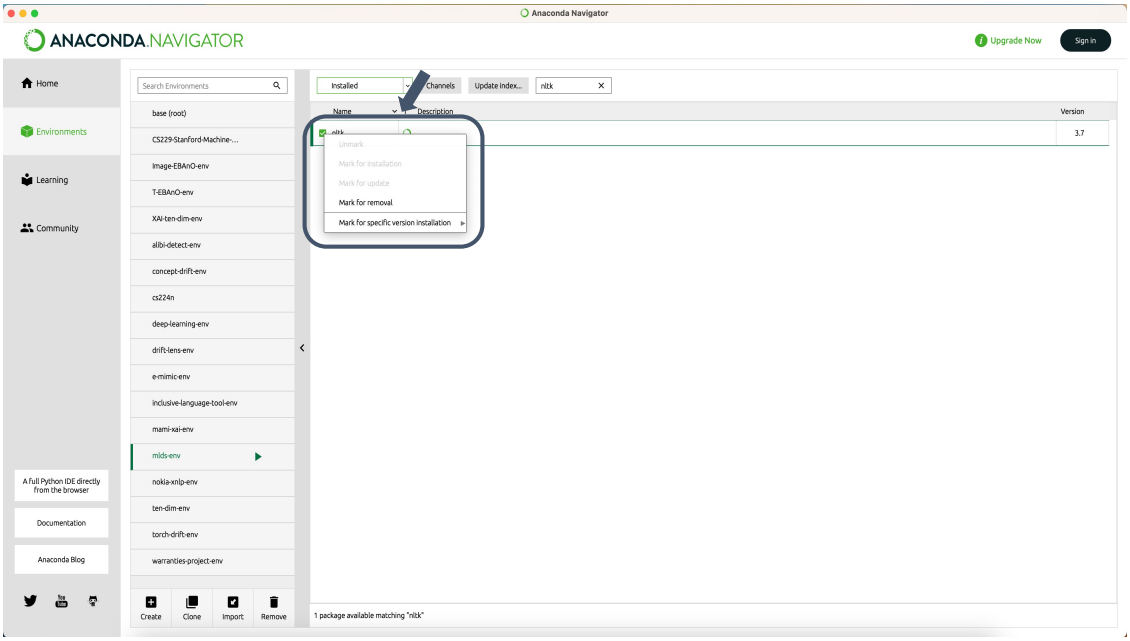


A new line corresponding to the installed package (in this case, NLTK) should appear.

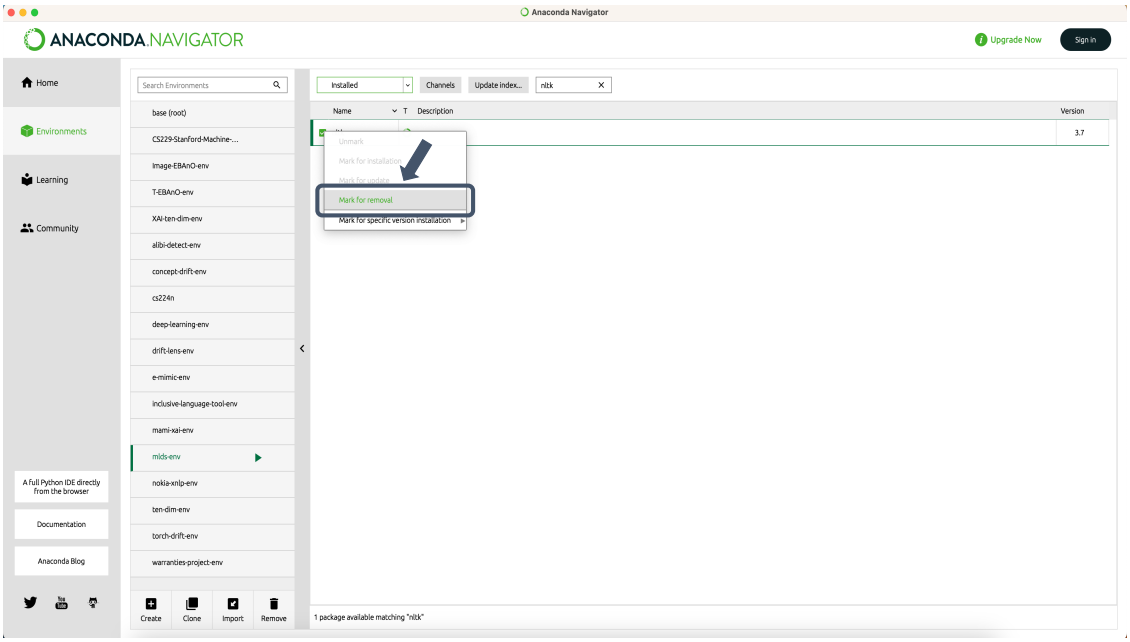


3.1.4 Uninstall the package

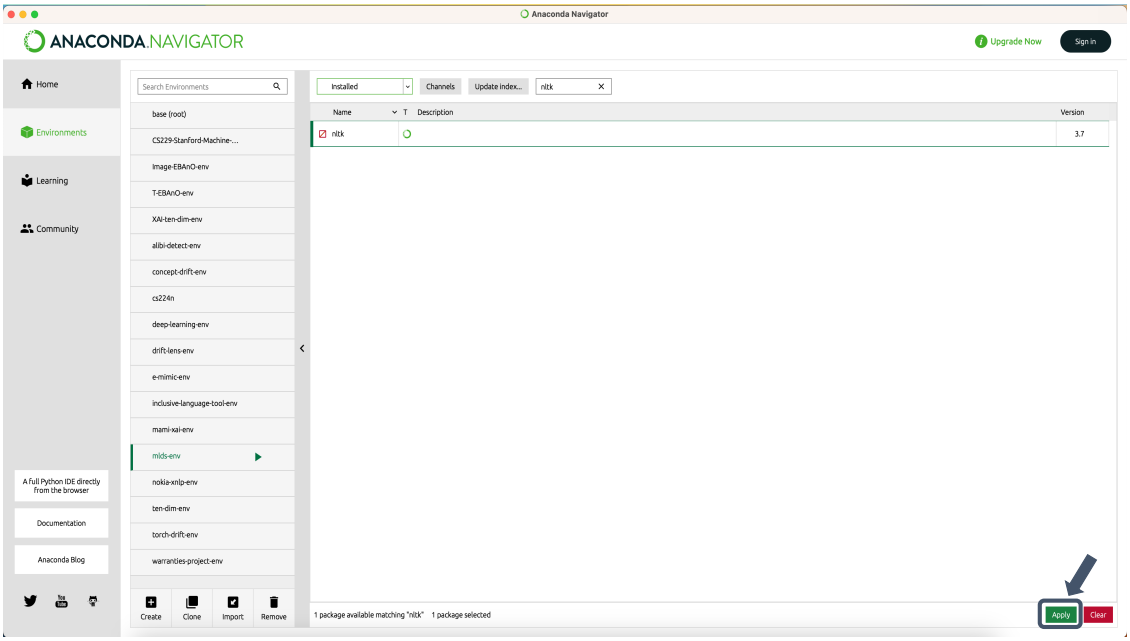
To **uninstall a package**, click the green ✓ on the line corresponding to the package you want to remove.



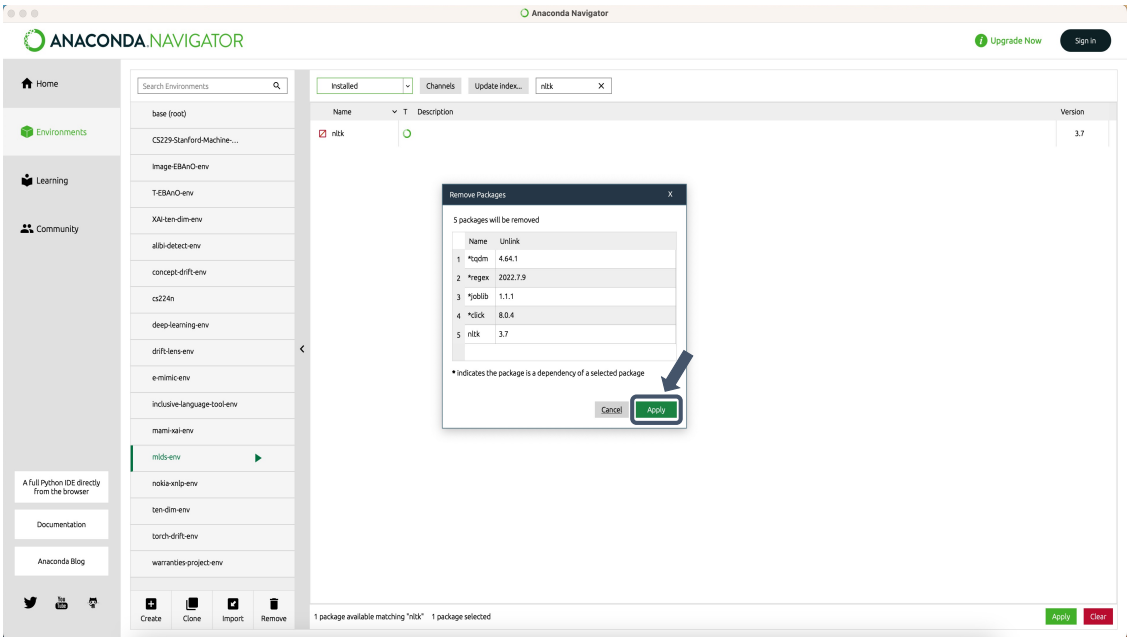
Then, click on the `"Mark for Removal"` option.



The green ✓ will become a red crossed box. Then, click the `"Apply"` button in the right-hand bottom corner.

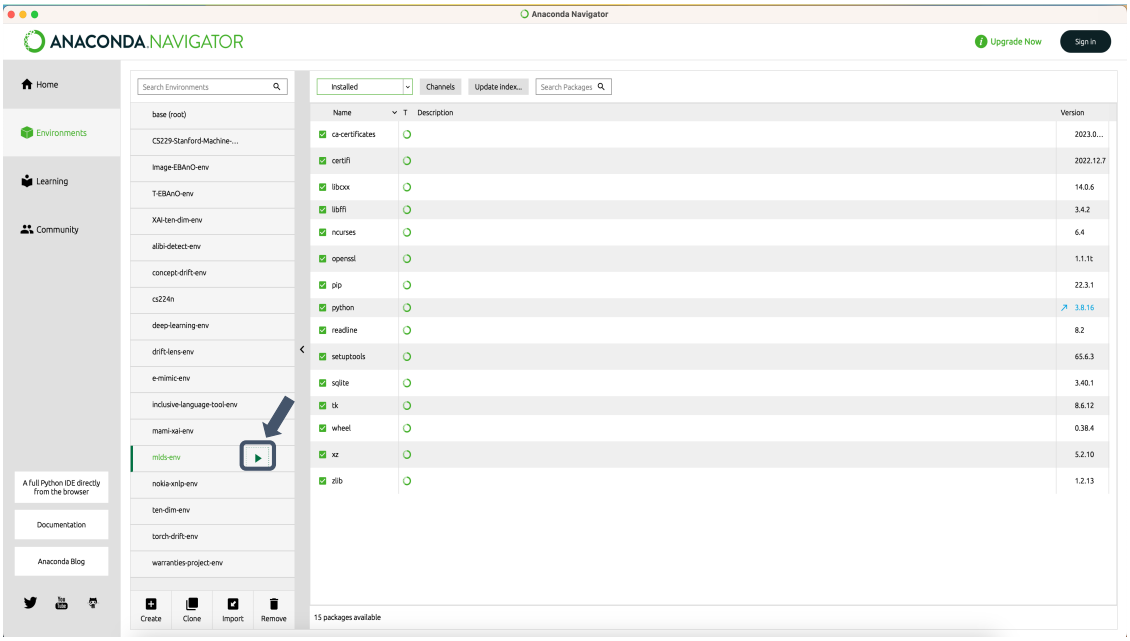


It will open a new window with all the packages that will be **removed**. Finally, click the `"Apply"` button in the right-hand bottom corner to start the package uninstallation.

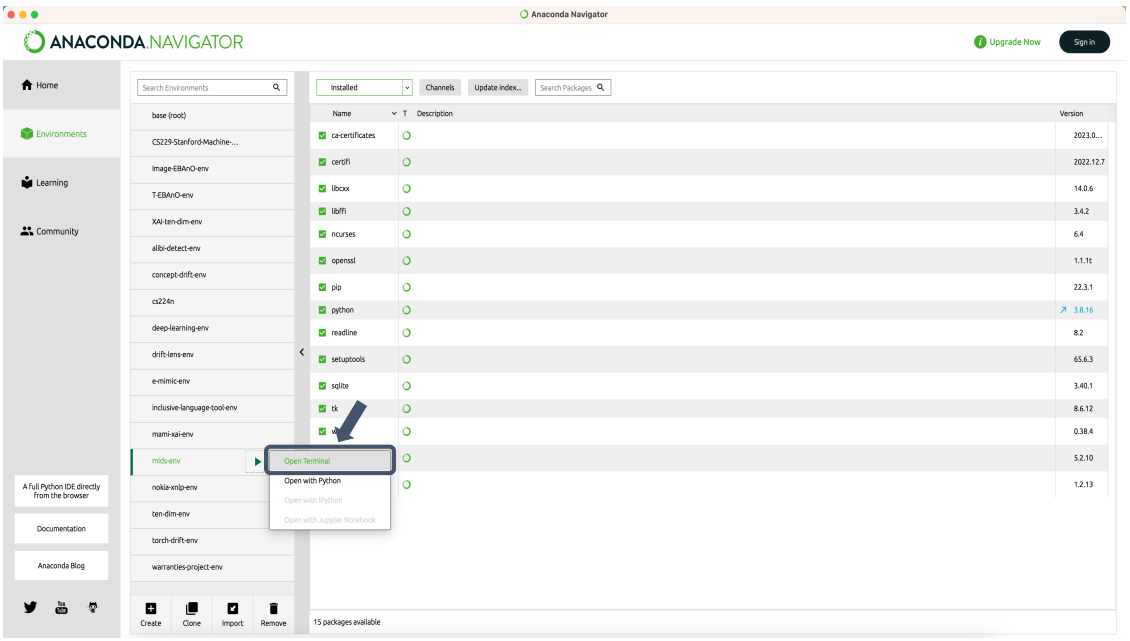


3.2 Install a package with the command line

This Section will show you how to install packages by **terminal**, with the **pip** (Section 3.2.1) and **conda** (Section 3.2.2) commands. To open the **terminal** for your environment, select the corresponding line and click the green ▷ symbol.



Then, select the `"Open Terminal"` option.

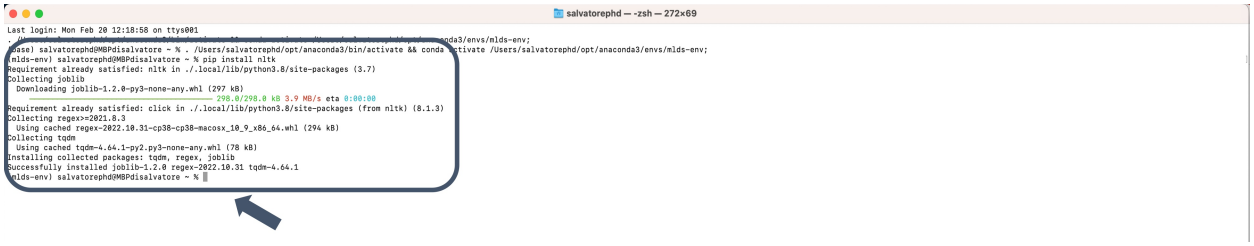


This will open the **terminal** with the selected **environment activated** (i.e., if you install a package, it will be installed in the activated environment). You can see the **activated environment** in the round brackets at the left of the line (e.g., *mlds-env*).

3.2.1 Install a package with the pip command

Some packages could **not** be available in the **conda environment**. You can find and install the package with another **package manager** like **pip**. To install a package with the **pip** command, type the command `pip install package-name`, in this case, NLTK. You can find the specific **pip** command for each package installation on the official documentation websites.

Press `enter` on your keyboard to start the download and the installation. It could take some minutes



3.2.2 Install a package with the conda command

Instead, to install a package with the **conda** command, type the command `conda install package-name`, in this case, NLTK. You can find the specific **conda** command for each package installation on the official documentation websites.

The terminal will show all the **dependencies** (i.e., other packages) that will be installed. press `y` and then `enter` to start the download and the installation.

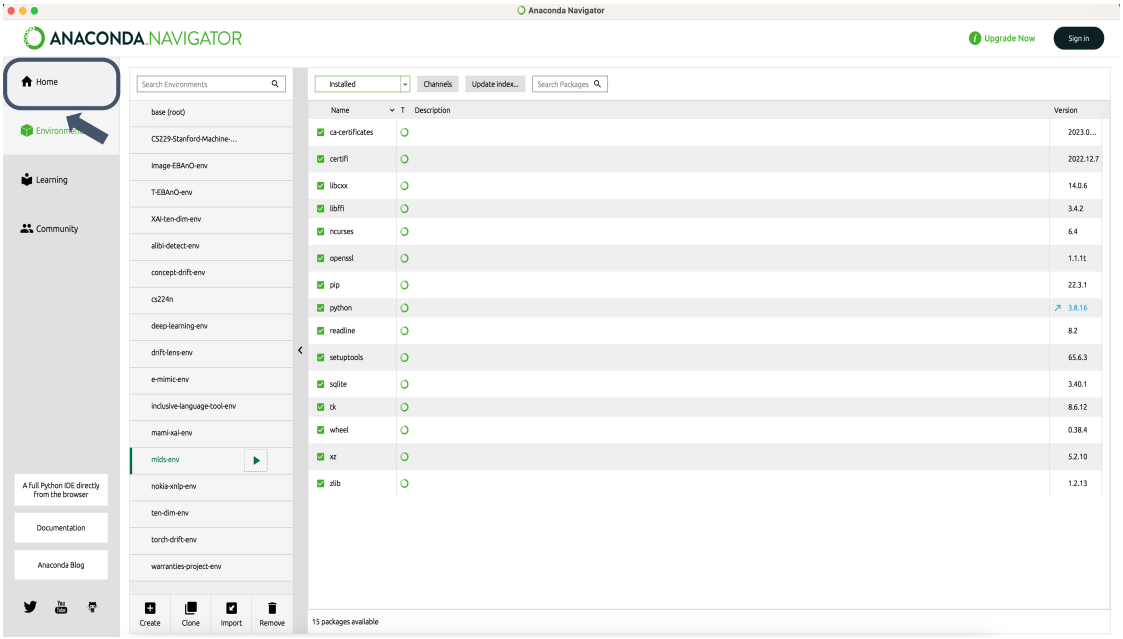
4 Jupyter notebook

Jupyter Notebook is a powerful tool for **developing** and **presenting** data science projects **interactively**. In a **Jupyter Notebook** document, you can combine **code**, **visualizations**, **texts**, and display **outputs**. You can find a good guide at the following URL [3]: `https://www.dataquest.io/blog/jupyter-notebook-tutorial/`.

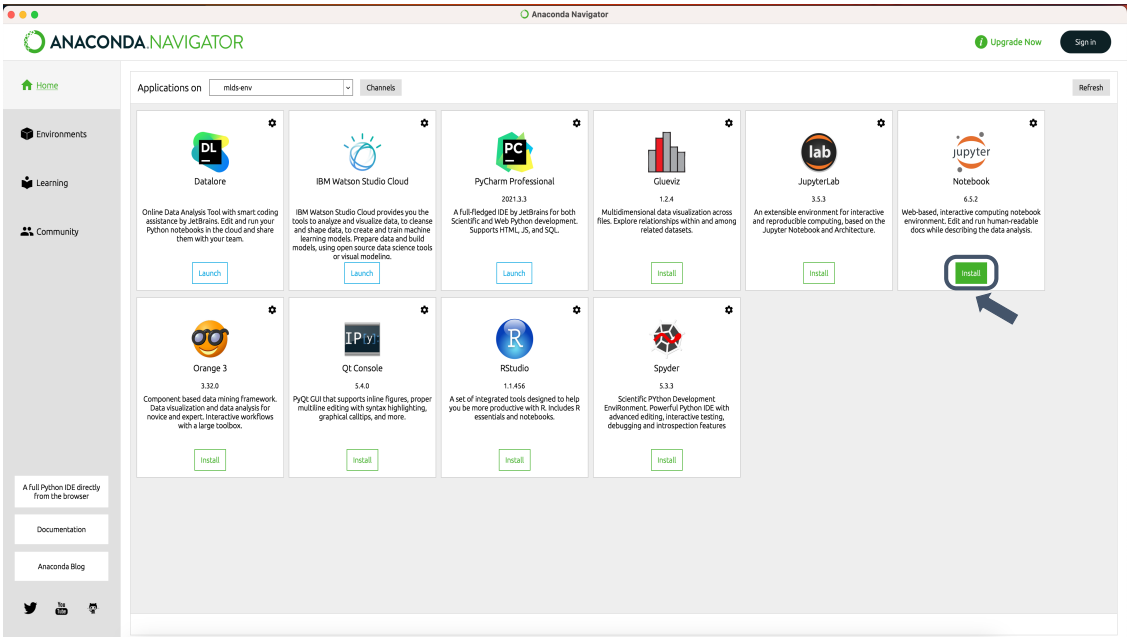
The following sections will show you how to: i) install Jupyter notebook using Anaconda (Section 4.1); ii) launch Jupyter from Anaconda 4.2; iii) create your first Jupyter notebook 4.3; iv) use cells and kernels to effectively exploit Jupyter notebooks (Sections 4.4 and 4.5); v) exploit advanced features of Jupyter notebooks (Section 4.6).

4.1 Install Jupyter

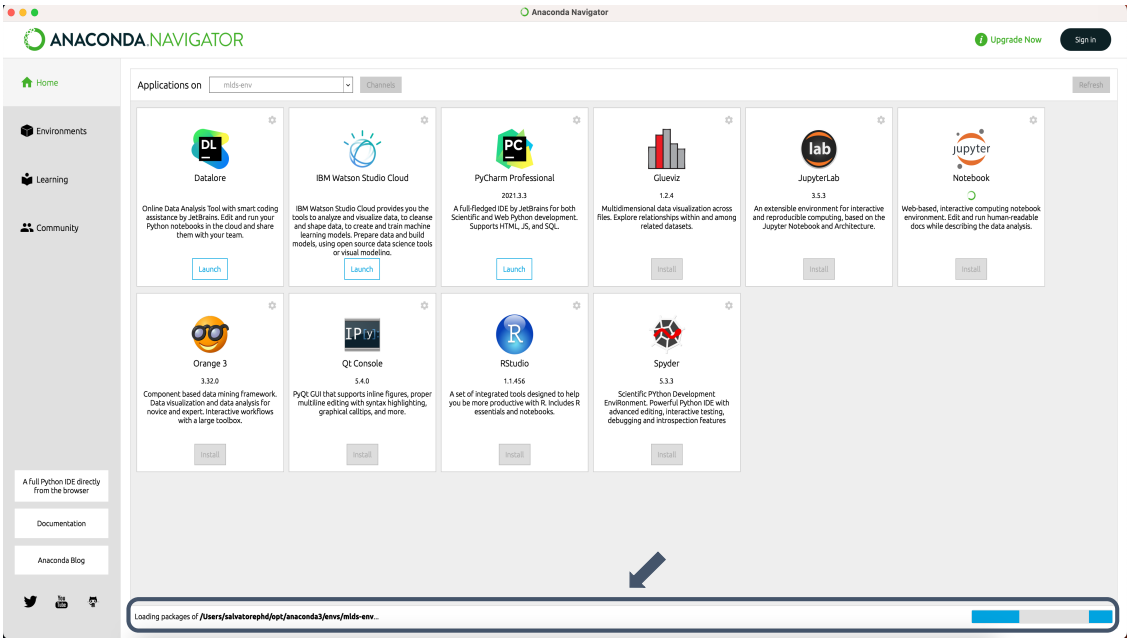
To install **Jupyter**, you must first go into the `Home` section, which contains all the applications for the current environment. Please check that the created virtual environment is selected (in this case, *mlds-env*).



Then, click the `"Install"` button under the Jupyter application box.

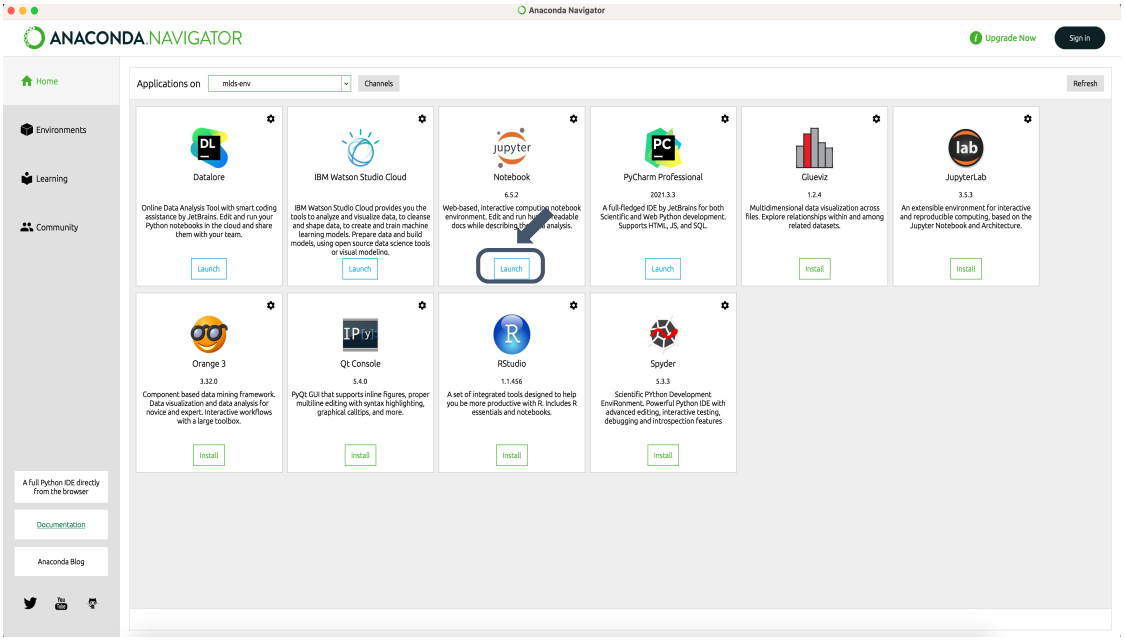


This will start the download and installation process. It may require some minutes.

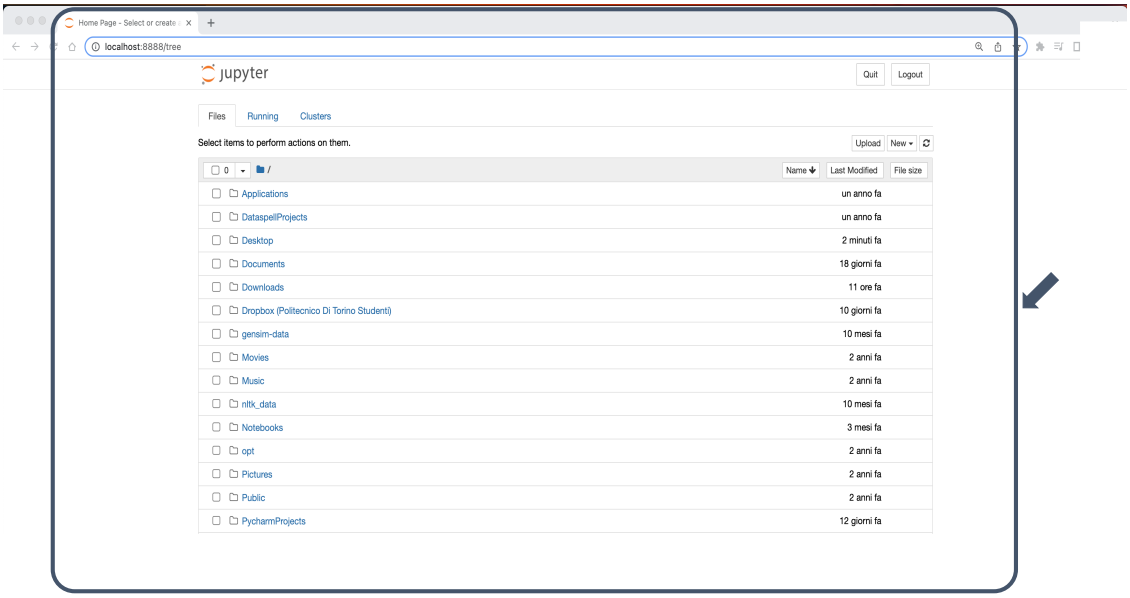


4.2 Launch Jupyter

After the installation, the **Anaconda dashboard** will show you the `"Launch"` button under the Jupyter Notebook box. Click on `"Launch"` to start Jupyter notebook.

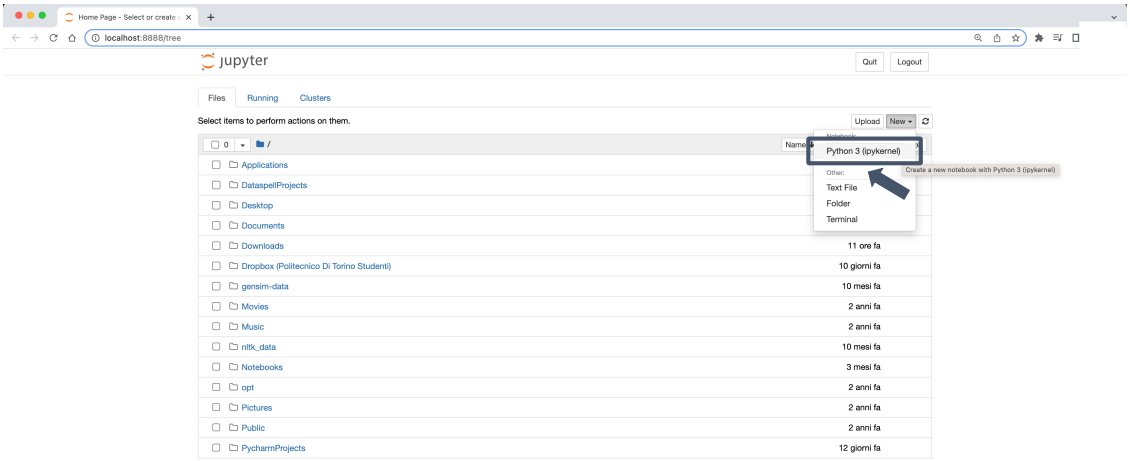


It will open the **Notebook Dashboard** for **exploring**, **editing**, and **creating** notebooks. Here you can create new folders, notebooks, etc. The URL for the dashboard is *https://localhost:8888/tree*. Localhost is not a website but indicates that the content is run on your local machine.

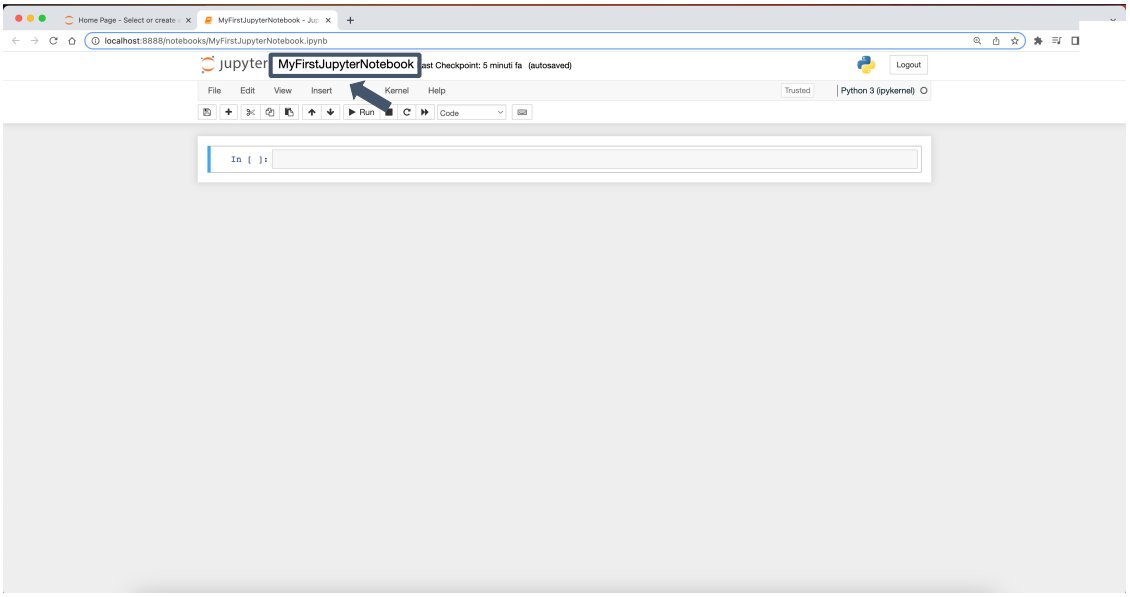


4.3 Create a new Jupyter notebook

To create your first Jupyter notebook click the `"New"` drop-down button in the top-right menu and select `"Python 3"`. This will open your first Jupyter notebook in a new tab. You can **open** and **run multiple notebooks simultaneously** in multiple tabs.



It will create a new file `Untitled.ipynb`. Each `.ipynb` file is a text file that describes the contents of your notebook in a format called JSON. Each time you create a new notebook, a new `.ipynb` file will be created. Notice that the notebook extension `.ipynb` is different from the normal python file extension `.py`. Please rename now your filename from the top text box, or, very soon, you will have several `Untitled.ipynb`, `Untitled (1).ipynb` notebooks. The notebook's name should explain the content.



4.3.1 The Jupyter Notebook interface

The two main concepts that you should learn to use notebooks properly are **cells** and **kernels**:

- The `cell` is a container for **code** to be executed or **text** to be displayed in the notebook by the kernel (Section 4.4).

- The `kernel` is a computational engine that **executes the code** contained in a notebook document (Section 4.5).

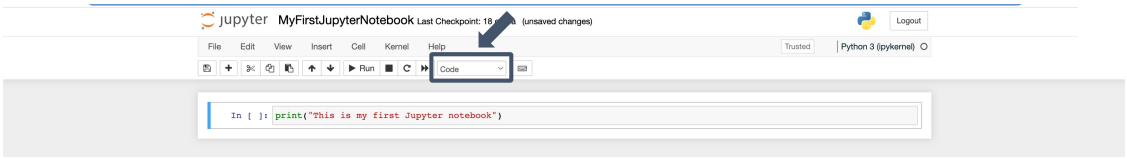
4.4 Notebook cells

Cells compose the body of the notebook. They could contain code, plain text, images, LaTeX, math formulas, etc. There are two main cell types that you should learn:

- **Code** cells (Section 4.4.1)
- **Markdown** cells (Section 4.4.2)

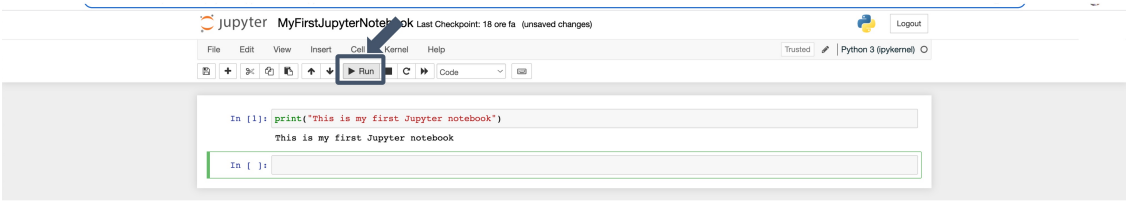
4.4.1 Code cell

Code cells contain code to be executed in the kernel. When the code is run, the notebook displays the output below the code cell that generated it. Note that cells do not have to be executed in order. It is also possible to execute a cell at the end and then one at the beginning of the notebook. The cell type is shown in the drop-down menu. The default type is **Code**.

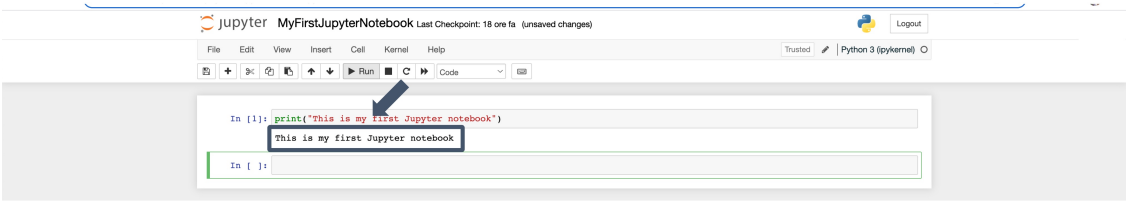


You can run a **Code** cell by:

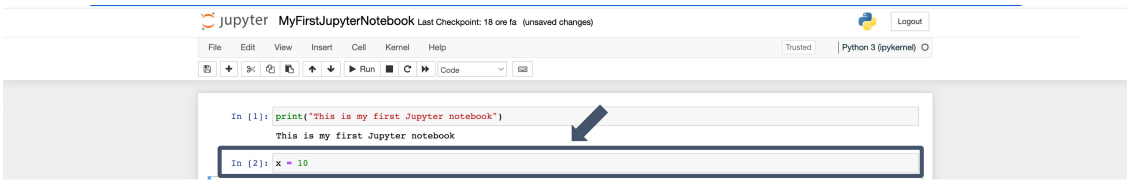
- clicking the `"Run"` button
- pressing `ctr + enter`
- pressing `maiusc + enter` (in this case, it also goes to the next cell)



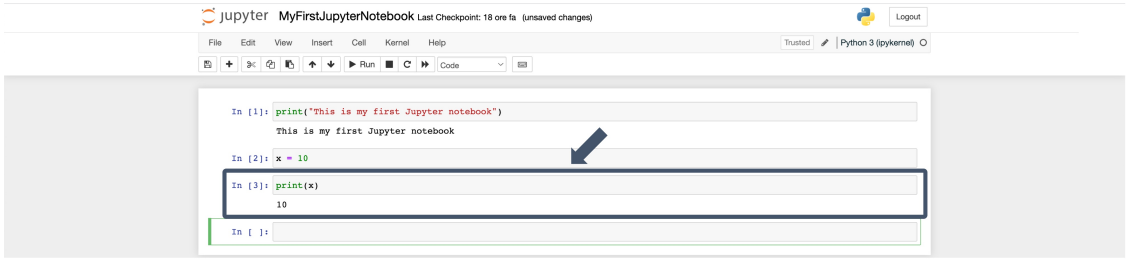
In this case, the execution of the cell will print the string `"This is my first Jupyter notebook"` as output. Each cell could produce an output.



The following cell will create a new variable called x and assigns the values of 10 to x. In this case, no output is produced by the cell.

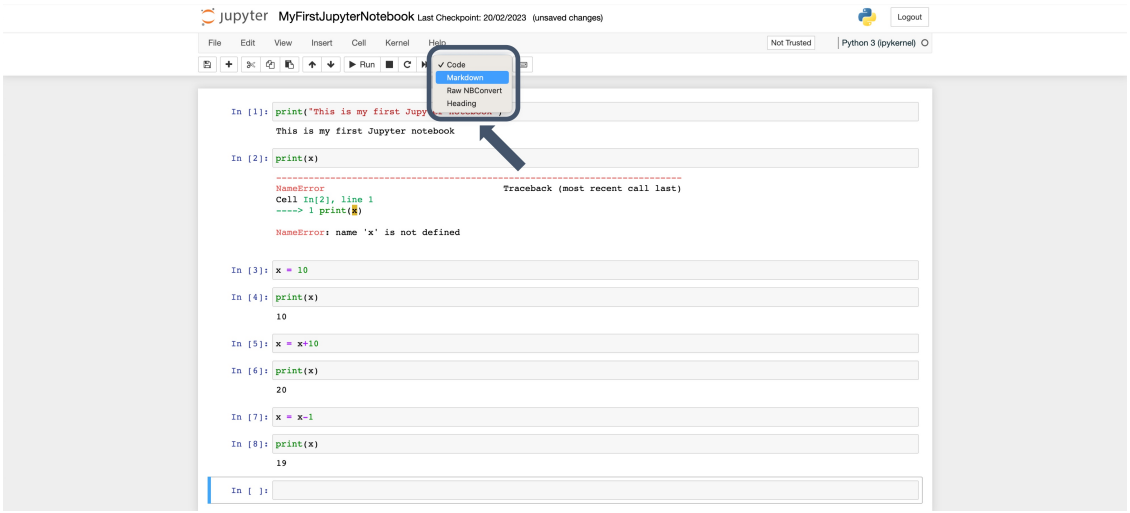


You should use the print function to output the value of x.



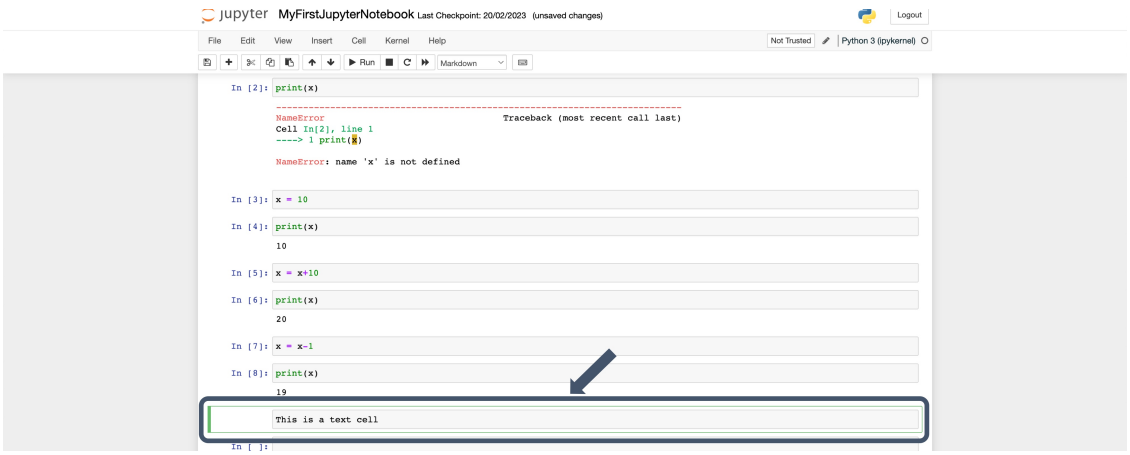
4.4.2 Markdown cell

Markdown cells contain **text** formatted using Markdown [2] and displays its output in-place when the Markdown cell is run. Markdown is a lightweight markup language that you can use to add formatting elements to plaintext text documents. This cheat sheet will cover the most common elements (cheatsheet). To define a **Markdown cell**, select the cell and click on the `"Markdown"` option in the top drop-down menu.

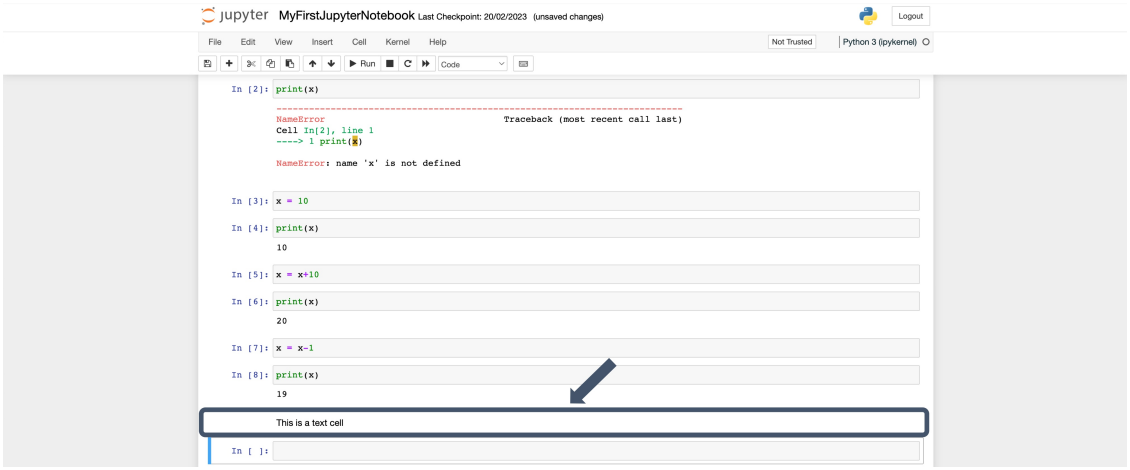


4.4.3 Text Markdown cell

You can write plaintext in a cell. This text is not a code. Therefore, it will not be really executed. You can also empathize text with **bold** or *italic* with `**bold**` and `*italic*` respectively.



If you run the cell containing **plaintext**, it will be displayed formatted as output. This can add **narrative** to your Jupyter notebook.

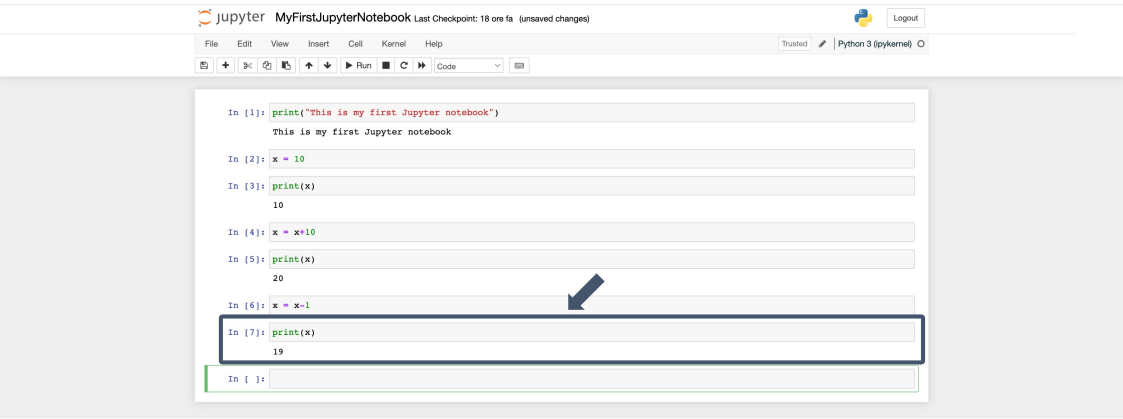


4.4.4 Heading Markdown cell

You can also add first, second, and third-level **headings**.

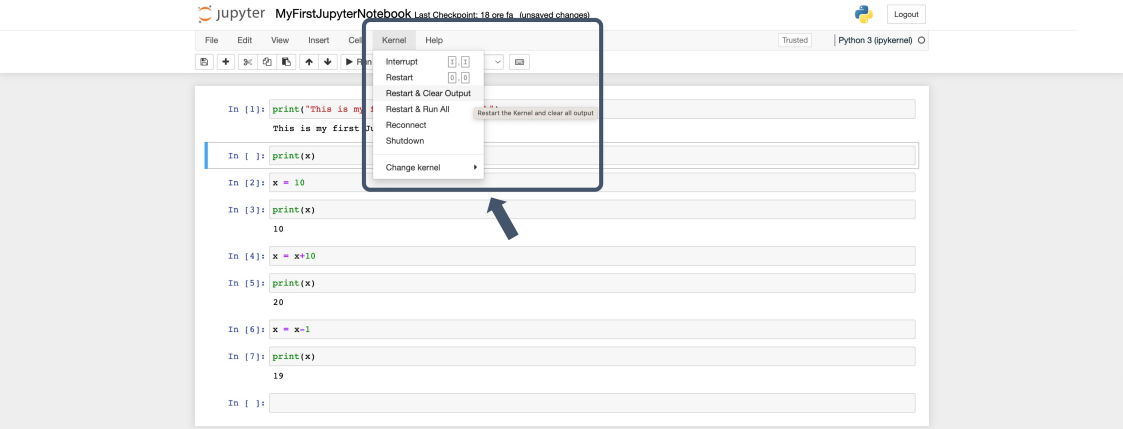
4.5 Notebook kernel

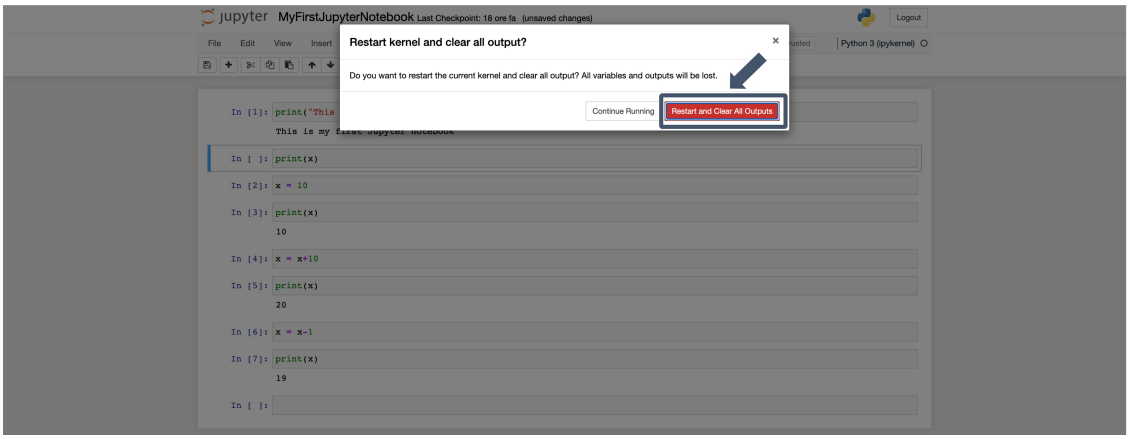
When you run a code cell, that code is executed within the **kernel**, and the **outputs** are returned to the cells to be **displayed**. The **kernel's state persists over time** between cells. It pertains to the document as a whole and not individual cells. For example, if you import libraries in one cell, they will be available in another. If you define the value of a variable in one cell, the variable's value also persists for the other cells.



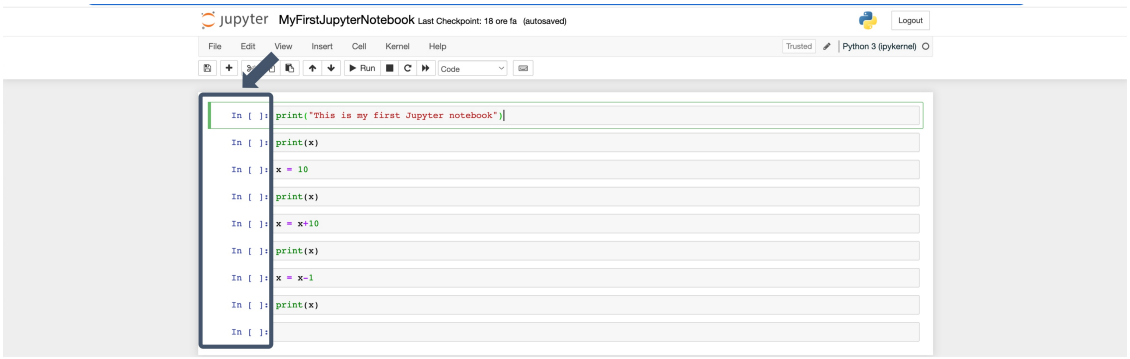
4.5.1 Restarting a kernel

If you **restart the kernel**, the **notebook's status is deleted**. After the kernel's restart, all the values of your variables are reset. To restart the kernel, click the `"Kernel"` button in the top menu. Then, select `"Restart & Clear Output"`. This will restart your kernel and clear all the outputs in the cells. You can also select `"Restart & Run All"` to restart your kernel and run all cells in order.

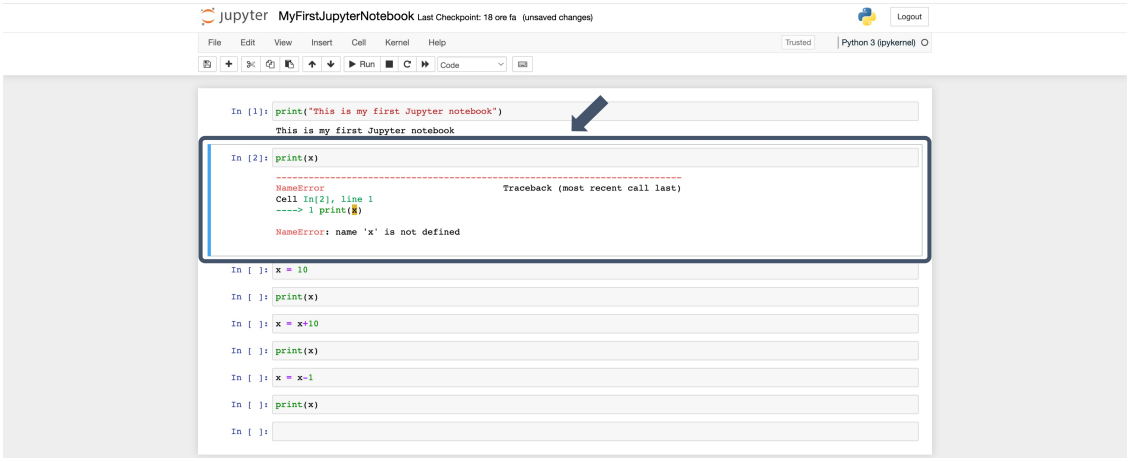




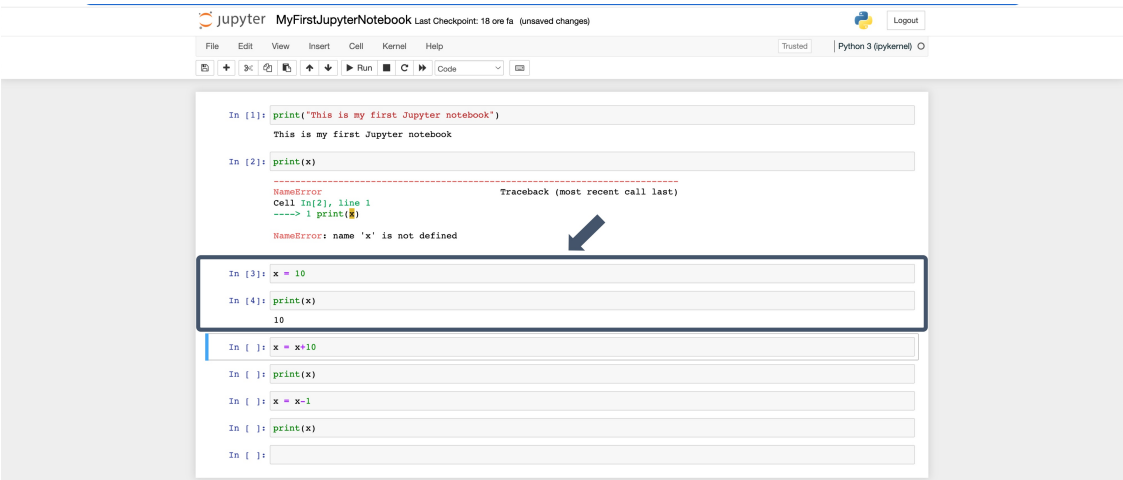
Restarting the kernel clears all the cells' outputs and initializes the run identification number of each cell.



What do you think will happen if you now print the value of the variable x again? It will raise an error message because restarting the kernel caused a reset of the notebook status and, consequently, all the previously defined variables.



Therefore, you should define x again to print its value.



4.6 Jupyter notebook advanced tips, tricks, and shortcuts

More advanced tips and commands such as keyboard shortcuts, pretty display, executing shell commands, using LaTeX could be found here [1] (https://www.dataquest.io/blog/jupyter-notebook-tips-tricks-shortcuts/)

References

[1] Dataquest. *28 Jupyter notebook tips, tricks, and shortcuts*. Feb. 2023. URL: https://www.dataquest.io/blog/jupyter-notebook-tips-tricks-shortcuts/.

[2] *Markdown guide*. URL: https://www.markdownguide.org/.

[3] Benjamin Pryke. *How to use Jupyter Notebook: A beginner's tutorial*. Feb. 2023. URL: https://www.dataquest.io/blog/jupyter-notebook-tutorial/.

[4] *Set up virtual environment for python using anaconda*. Apr. 2022. URL: https://www.geeksforgeeks.org/set-up-virtual-environment-for-python-using-anaconda/.

[5] *What is Anaconda?: Domino data science dictionary*. URL: https://www.dominodatalab.com/data-science-dictionary/anaconda#:~:text=Anaconda%20Navigator%20is%20included%20in,the%20packages%20and%20update%20them..