Anaconda Navigator

Anaconda is one of the most used package managers for Python which comes with all the necessary packages for day-to-day data science and machine learning operations. Having a good understanding of Anaconda will help you organize and maintain your machine learning projects efficiently.

Below, you will find the instructions for installing anaconda on different Operating Systems.

Windows: [Installing on Windows — Anaconda documentation]

MacOS: [Installing on macOS — Anaconda documentation]

Linux: [Installing on Linux — Anaconda documentation]

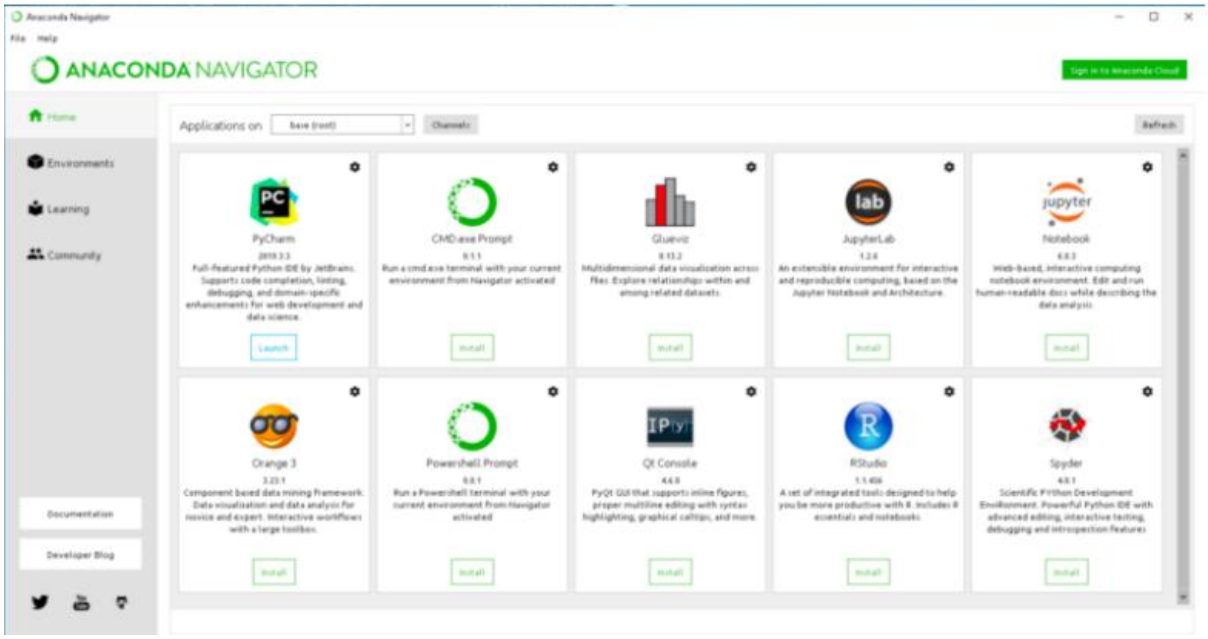


Fig 1.0 Anaconda navigator

After installing Anaconda, launch the Jupyter notebook from the anaconda navigator shown in the image above.

Jupyter notebook

The Jupyter Notebook is an incredibly powerful tool for interactively developing and presenting data science projects.

A notebook integrates code and its output into a single document that combines visualizations, narrative text, mathematical equations, and other rich media. In other words, it's a single document in which one can run code, display the output, and also add explanations, formulas, charts, and make work more transparent, understandable, repeatable, and shareable.

Using Notebooks is now a major part of the data science workflow in companies across the globe. If your goal is to work with data, using a Notebook will speed up your workflow and make it easier to communicate and share your results.

Best of all, as part of the open source project Jupyter, Jupyter Notebooks are completely free. The software can be downloaded on its own or as part of the Anaconda data science toolkit.

Creating a notebook

In this section, we're going to learn to run and save notebooks, familiarize ourselves with their structure and interface.

Running Jupyter

On windows, you can run Jupyter via the shortcut Anaconda adds to your start menu, which will open a new tab in your default web browser. It should look like the screenshot below.

Fig 1.1 Jupyter notebook

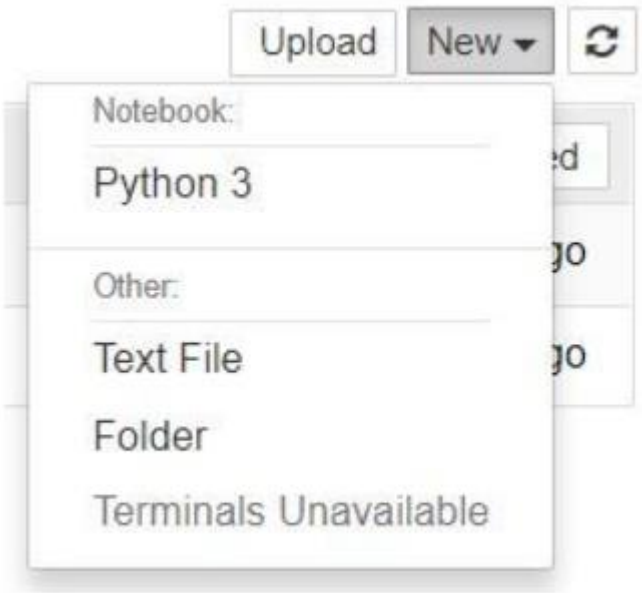
It is also possible to start the dashboard on any system via the command prompt (or terminal on Unix systems) by entering the command ***jupyter notebook***; in this case, the current working directory will be the start-up directory.

With Jupyter Notebook open in your browser, you may have noticed that the URL for the dashboard is something like ***http://localhost:8888/tree***. Localhost is not a website, but indicates that the content is being served from your local machine: your own computer.

Jupyter's Notebooks and dashboard are web apps, and Jupyter starts up a local Python server to serve these apps to your web browser, making it essentially platform-independent and opening the door to easier sharing on the web.

(If you don't understand this yet, don't worry — the important point is just that although Jupyter Notebooks opens in your browser, it's being hosted and run on your local machine. Your notebooks aren't actually on the web until you decide to share them.)

The dashboard's interface is mostly self-explanatory — though we will briefly discuss it later. So, what are you waiting for? Browse the folder in which you would like to create your first notebook, click the "New" drop-down button in the top-right and select "Python 3":



Hey presto, here we are! Your first Jupyter Notebook will open in a new tab — each notebook uses its own tab because you can open multiple notebooks simultaneously.

If you switch back to the dashboard, you will see the new file **Untitled.ipynb**, some green text that indicates your notebook is running should also be seen.

What is an ipynb File?

Basically, each .ipynb file is one notebook. Hence, each time you create a new notebook, a new .ipynb file will be created.

The Notebook Interface

Now that you have an open notebook in front of you, its interface will hopefully not look entirely alien. After all, Jupyter is essentially just an advanced word processor.

Why not take a look around? Check out the menus to get a feel of it, take a few moments to especially scroll down the list of commands in the command palette, which is the small button with the keyboard icon (or Ctrl + Shift + P).

There are two fairly prominent terms that you should notice, which are probably new to you: cells and kernels are key both to understanding Jupyter, and to what makes it more than just a word processor. Fortunately, these concepts are not difficult to understand.

1. A kernel is a "computational engine" that executes the code contained in a notebook document.
2. A cell is a container for text to be displayed in the notebook or code to be executed by the notebook's kernel.

Cells

We'll return to kernels a little later, but first, let's come to grips with cells. Cells form the body of a notebook. There are two main cell types that we will cover:

1. A code cell contains code to be executed in the kernel. When the code is run, the notebook displays the output below the code cell that generated it.

2. A Markdown cell contains text formatted using Markdown, and displays its output in-place when the Markdown cell is run.

The first cell in a new notebook is always a **code cell**.

Let's test it out with a classic hello world example: Type print ('Hello World!') into the cell and click the run button in the toolbar above or press Ctrl + Enter.

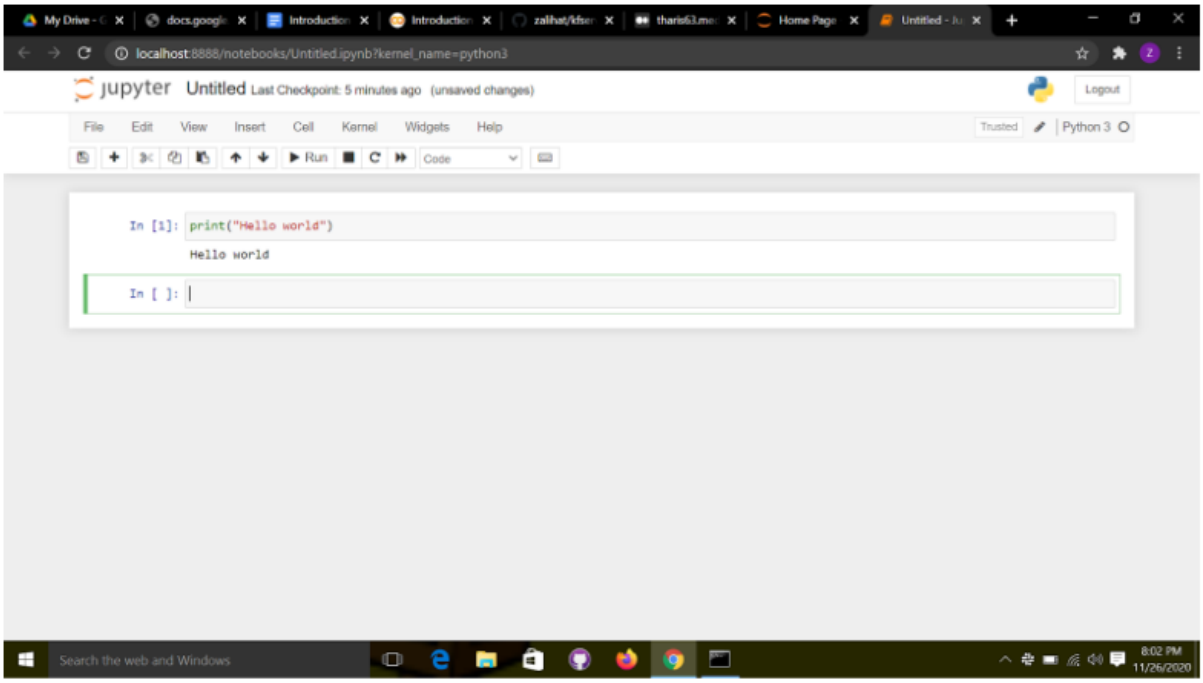


Fig 1.2 A typical Jupyter notebook

When we run the cell, its output is displayed below, and the label to its left will have changed from In [ ] to In [1].

The output of a code cell also forms part of the document, which is why you can see it in this course. You can always tell the difference between code and Markdown cells, because code cells have that label on the left and Markdown cells do not.

The "**In**" part of the label is simply short for "**Input**," while the label number indicates when the cell was executed on the kernel — in this case the cell was executed first.

Run the cell again and the label will change to In [2] because now the cell was the second to be run on the kernel. It will become clearer why this is so useful later on, when we take a closer look at kernels.

From the menu bar, click Insert and select Insert Cell Below, to create a new code cell underneath your first and try out the following code to see what happens. Do you notice anything different?

import time

time.sleep(3)

This cell doesn't produce any output, but it does take three seconds to execute. Notice how Jupyter signifies when the cell is currently running by changing its label to In [*].

In general, the output of a cell comes from any text data specifically printed during the cell's execution, as well as the value of the last line in the cell, be it a lone variable, a function call, or anything else. For example:

You'll find yourself using this almost constantly in your own projects, and we'll see more of it later on.

Keyboard Shortcuts

So what can we do to a cell when it's in command mode? So far, we have seen how to run a cell with Ctrl + Enter, but there are a plethora of other commands we can use. The best way to use them is with keyboard shortcuts

Keyboard shortcuts are a very popular aspect of the Jupyter environment because they facilitate a speedy cell-based workflow. Many of these are actions you can carry out on the active cell when it's in command mode.

Below, you'll find a list of some of Jupyter's keyboard shortcuts. You don't need to memorize them all immediately, but this list should give you a good idea of what's possible.

Toggle between edit and command mode with Esc and Enter, respectively.

Once in command mode:

Scroll up and down your cells with your Up and Down keys.

Press A or B to insert a new cell above or below the active cell.

M will transform the active cell to a Markdown cell.

Y will set the active cell to a code cell.

D + D (D twice) will delete the active cell.

Z will undo cell deletion.

Hold Shift and press Up or Down to select multiple cells at once. With multiple cells selected, Shift + M will merge your selection.

Ctrl + Shift + -, in edit mode, will split the active cell at the cursor.

You can also click and Shift + Click in the margin to the left of your cells to select them.

Go ahead and try these out in your own notebook.

Kernels

Behind every notebook runs a kernel. When you run a code cell, that code is executed within the kernel. Any output is returned back to the cell to be displayed. The kernel's state persists over time and between cells — it pertains to the document as a whole and not individual cells.

Most of the time when you create a notebook, the flow will be top-to-bottom. But it's common to go back to make changes. When we do need to make changes to an earlier cell, the order of execution we can see on the left of each cell, such as In [6], can help us diagnose problems by seeing what order the cells have run in.

And if we ever wish to reset things, there are several incredibly useful options from the Kernel menu:

Restart: restarts the kernel, thus clearing all the variables etc. that were defined.

Restart & Clear Output: same as above but will also wipe the output displayed below your code cells.

Restart & Run All: same as above but will also run all your cells in order from first to last.

If your kernel is ever stuck on a computation and you wish to stop it, you can choose the Interrupt option.

Naming Your Notebooks

Before you start writing your project, you'll probably want to give it a meaningful name. Click the file name **Untitled** in the upper left of the screen to enter a new file name, and hit the Save icon (which looks like a floppy disk) below it to save.

Save and Checkpoint

Now we've got started, it's best practice to save regularly. Pressing Ctrl + S will save our notebook by calling the "Save and Checkpoint" command, but what is this checkpoint thing?

Every time we create a new notebook, a checkpoint file is created along with the notebook file. It is located within a hidden subdirectory of your save location called .ipynb_checkpoints and is also a .ipynb file.

By default, Jupyter will autosave your notebook every 120 seconds to this checkpoint file without altering your primary notebook file. When you "Save and Checkpoint," both the notebook and checkpoint files are updated. Hence, the checkpoint enables you to recover your unsaved work in the event of an unexpected issue.

You can revert to the checkpoint from the menu via "File > Revert to Checkpoint."

Sharing Your Notebooks

When people talk about sharing their notebooks, there are generally two paradigms they may be considering.

Most often, individuals share the end-result of their work, which means sharing non-interactive, pre-rendered versions of their notebooks. However, it is also possible to collaborate on notebooks with the aid of version control systems such as Git or online platforms like Google Colab.

Before You Share

A shared notebook will appear exactly in the state it was in when you export or save it, including the output of any code cells. Therefore, to ensure that your notebook is share-ready, so to speak, there are a few steps you should take before sharing:

1. Click “Cell > All Output > Clear”
2. Click “Kernel > Restart & Run All”

Wait for your code cells to finish executing and check ran as expected

This will ensure your notebooks don’t contain intermediary output, have a stale state, and execute in order at the time of sharing.

Exporting Your Notebooks

Jupyter has built-in support for exporting to HTML and PDF as well as several other formats, which you can find from the menu under “File > Download As.”

If you wish to share your notebooks with a small private group, this functionality may well be all you need. Indeed, as many researchers in academic institutions are given some public or internal webspace, and because you can export a notebook to an HTML file, Jupyter Notebooks can be an especially convenient way for researchers to share their results with their peers.

But if sharing exported files doesn’t cut it for you, there are also some immensely popular methods of sharing .ipynb files more directly on the web.

What Are Extensions?

Extensions are precisely what they sound like — additional features that extend Jupyter Notebooks's functionality. While a base Jupyter Notebook can do an awful lot, extensions offer some additional features that may help with specific workflows, or that simply improve the user experience.

For example, one extension called "**Table of Contents**" generates a table of contents for your notebook, to make large notebooks easier to visualize and navigate.

Another one, called **Variable Inspector**, will show you the value, type, size, and shape of every variable in your notebook for quick easy reference and debugging.

Another, called **ExecuteTime**, lets you know when and for how long each cell ran — this can be particularly convenient if you're trying to speed up a snippet of your code.

These are just the tip of the iceberg; there are many extensions available.

Where Can You Get Extensions?

To get the extensions, you need to install Nbextensions. You can do this using pip and the command line. If you have Anaconda, it may be better to do this through Anaconda Prompt rather than the regular command line.

Close Jupyter Notebooks, open Anaconda Prompt, and run the following command: pip install jupyter_contrib_nbextensions && jupyter contrib nbextension install.

Once you've done that, start up a notebook and you should see an Nbextensions tab. Clicking this tab will show you a list of available extensions. Simply tick the boxes for the extensions you want to enable, and you're off to the races!

Installing Extensions

Once Nbextensions itself has been installed, there's no need for additional installation of each extension. However, if you've already installed Nbextensions but aren't seeing the tab, you're not alone. This thread on Github details some common issues and solutions [cannot see jupyter nbextension configurator tab · Issue #72 · Jupyter-contrib/jupyter_nbextensions_configurator]

To learn more on jupyter notebook checkout this article [Advanced Jupyter Notebook Tutorial – Dataquest].

Creating a new notebook using Google Colaboratory

Running jupyter on Anaconda is great for testing things out locally, but installing and running anaconda can be stressful, mainly because of its size. There is an easier and cooler way to run Jupyter notebooks, and that is using the cloud based jupyter notebook called Google colaboratory simply called colab.

Colab is ideal for everything from improving your Python coding skills to working with deep learning libraries, like PyTorch, Keras, TensorFlow, and OpenCV. You can create notebooks in Colab, upload notebooks, store notebooks, share notebooks, mount your Google Drive and use whatever you've got stored in there, import most of your favorite directories, upload your personal Jupyter Notebooks, upload notebooks directly from GitHub, upload Kaggle files, download your notebooks, and do just about everything else that you might want to be able to do.

Working in Google Colab for the first time has been totally phenomenal and pretty shockingly easy, but it hasn't been without a couple of small challenges! If you know Jupyter Notebooks at all, you're pretty much good to go in Google Colab, but there are just a few differences.

Let's create your first notebook on colab.

To get started, head over to the [Colab home page], where you'll see an interface with a variety of Notebooks. You can select one of the already-existing examples built by the Google Colab team. Or, if you already have a Jupyter Notebook, you can upload it using the Upload option. Recently-used notebooks are listed in the Recent tab.

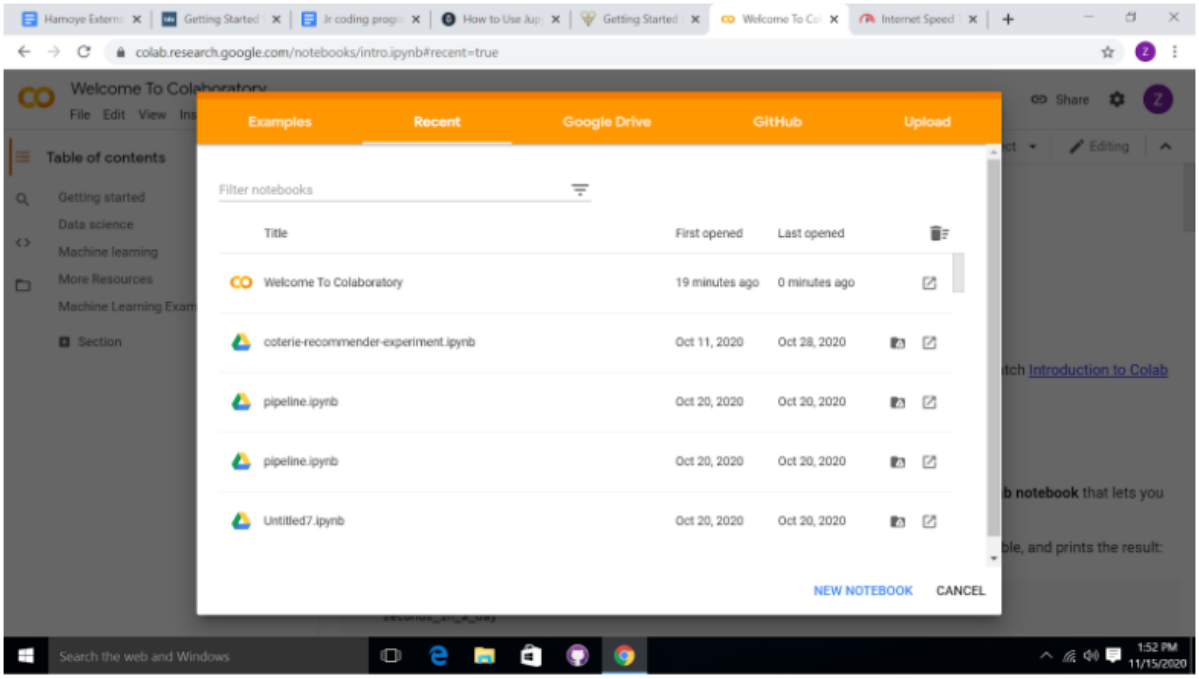


Fig 1.3: Your first notebook on colab

Note that Google Colab could be integrated with Google Drive. Within Google Colab, you can access all your files in Google Drive and manipulate them (i.e. edit, delete). This is in addition to the ability to create new files and directories. This makes life easier, and we'll explore using Colab with Drive later.

Once you've decided which Python version to use and are ready to create a new Notebook, click on the "NEW PYTHON x NOTEBOOK". Congratulations! You've created a Notebook in Google Colab that should look like this:

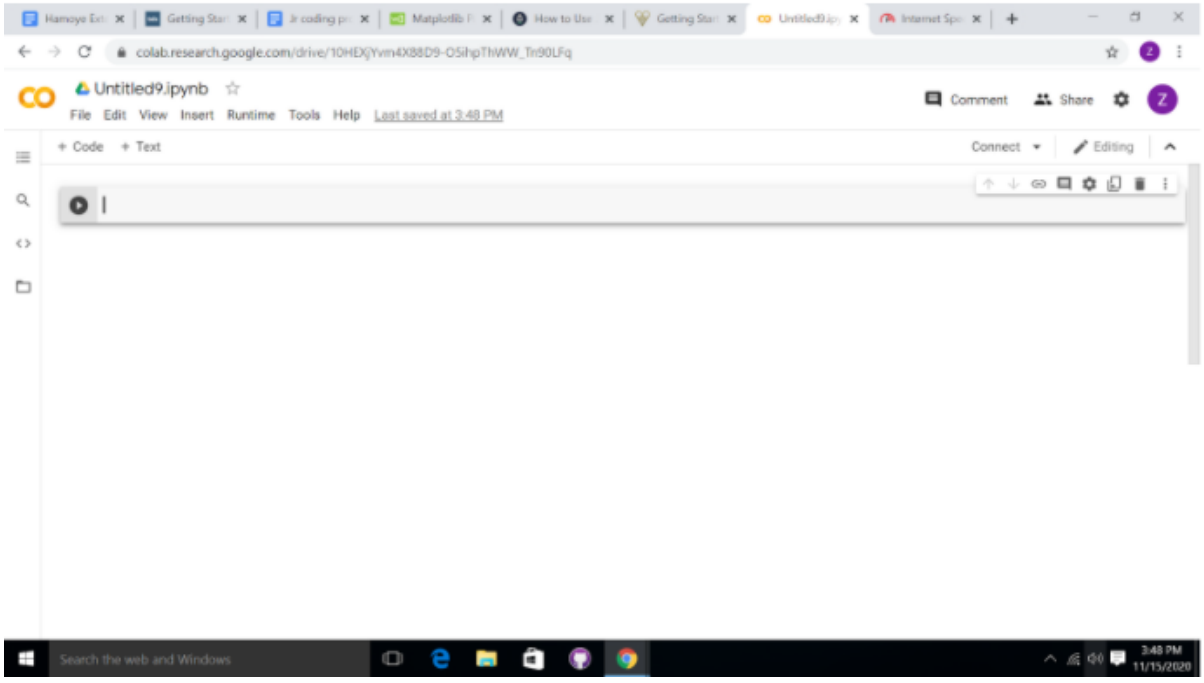


Fig 1.4: Your first notebook

For the purposes of this course we will be using colab, and we will explore the rich features of colab as we move forward.