Visualizing and Animating Data in Python

There is a need for mathematicians and scientists, especially computer scientists, to be able to visualize and animate data. This hackathon will teach you how to set up your own Python development environment and use it to develop your own visualizations and animations.

Prerequisites: To participate, you should be familiar with basic programming, say in Java, C, or Python. Prior experience in Python is not required.

The Development Environment: Anaconda

Python is available in many places, including directly from source at the Python Software Foundation. Python code can be developed with little more than a text editor- I often work with just Notepad++ and a command prompt.

One of the powers of Python is that it is extendable with various modules. For example, if my data is in Excel files, I might use the `openpyxl` module if I am reading from a modern `.xlsx` document, or the `xlrd` module if I am reading from an older `.xls` document.

Anaconda is a tool that provides both a full featured development environment for Python code and provides an easy way to download and manage Python modules.

Download and install Anaconda from https://www.anaconda.com/download/.[1]

Anaconda has installers for Windows, MacOS, and Linux.

The download is substantial and will take a few minutes; the installer is almost half a gigabyte in size. The installation process also takes a few minutes to complete.

Building a custom environment

When Anaconda is first launched, you will be presented with a tool that looks like Figure 1. The default behavior for Anaconda Navigator is to set up a base (root) environment that is used by default. This environment will be used by all the tools that you install and will be the location of the Python modules that you download.

This default environment can be used as-is, but it can lead to problems. There are many (many) Python modules that can be downloaded and used. Some of these are important, and regularly maintained. Others can be as simple as a student's or a hobbyist's side project. Each package has one or more dependencies- these can be libraries or executables on your computer or can be other Python modules. The interactions between these dependencies are the problem. Installing or upgrading one Python module may require a

[1] Anaconda will *ask* you to register, but it can be skipped.

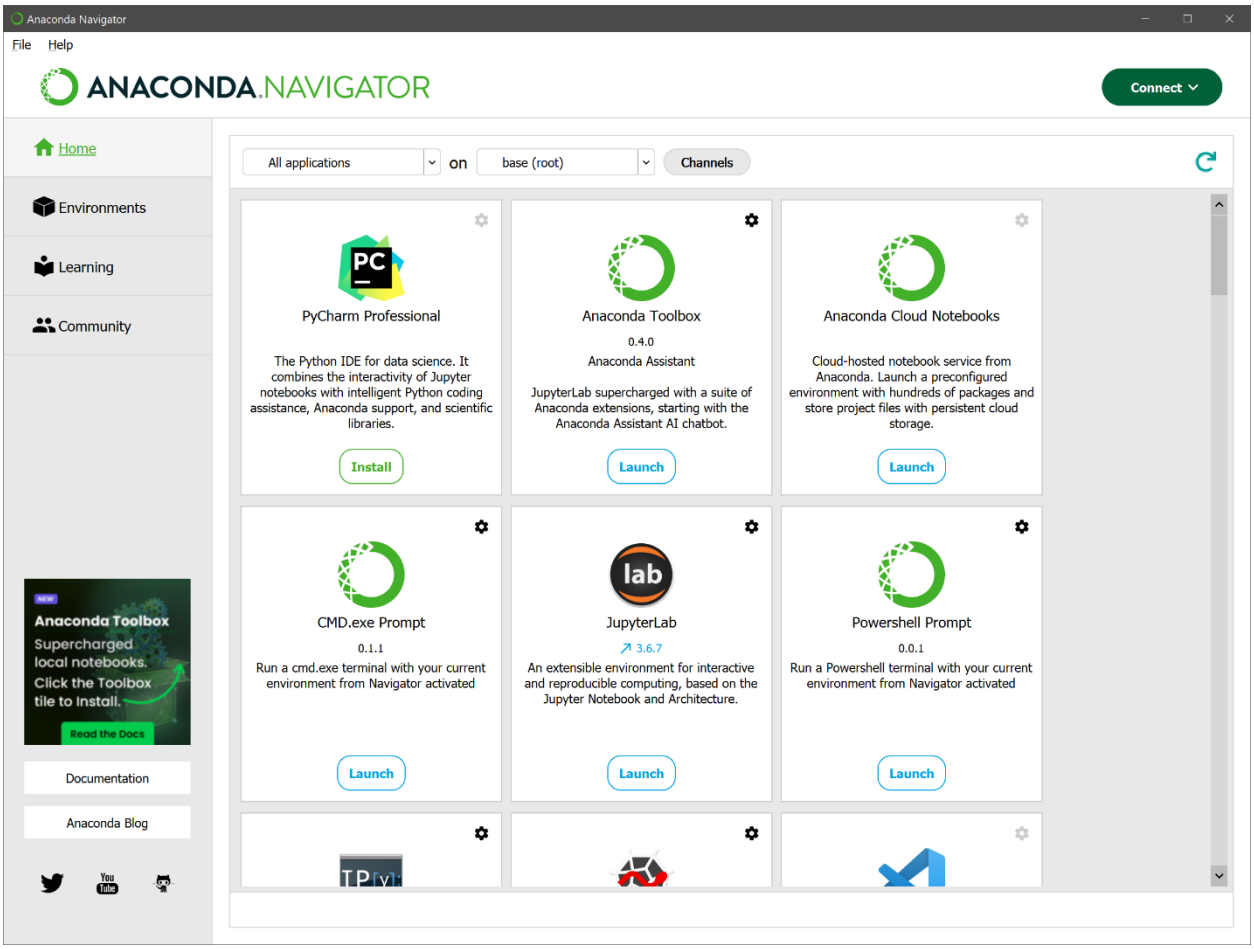

Figure 1: Anaconda Navigator 2.6.0

particular version for a dependent Python module, but you may have another Python module that requires a *different* version of the same dependency.

One solution is to set up custom environments for each major project or application; this the approach that will be outlined here.[2]

---

[2] I would love to say that the approach I am going to outline is foolproof and that if you follow all these steps then everything will work. *However*… The reason tools like Anaconda exist is because the interactions between the various dependencies *do* cause problems. Although Anaconda tries to resolve all of them, it does not always succeed. Despite all of my testing, it wouldn't surprise me if some unexpected error crops up and causes trouble. My debugging process would be the following- start by stopping/closing any existing Python programs. On Windows, CTRL+ALT+DEL will bring up Task Manager that would allow you to identify and close Python processes that might be running in the background without a corresponding window. Then try re-starting Anaconda. If the problem persists, it may be because of an un-caught dependency error; in this case you may wish to create a new Python environment and try again with different versions on Python and/or any installed packages.

Anaconda is recommended in this guide because is generally works well. If you prefer to install Python directly and to manage your dependencies manually (e.g. with pip) then feel free to do so.

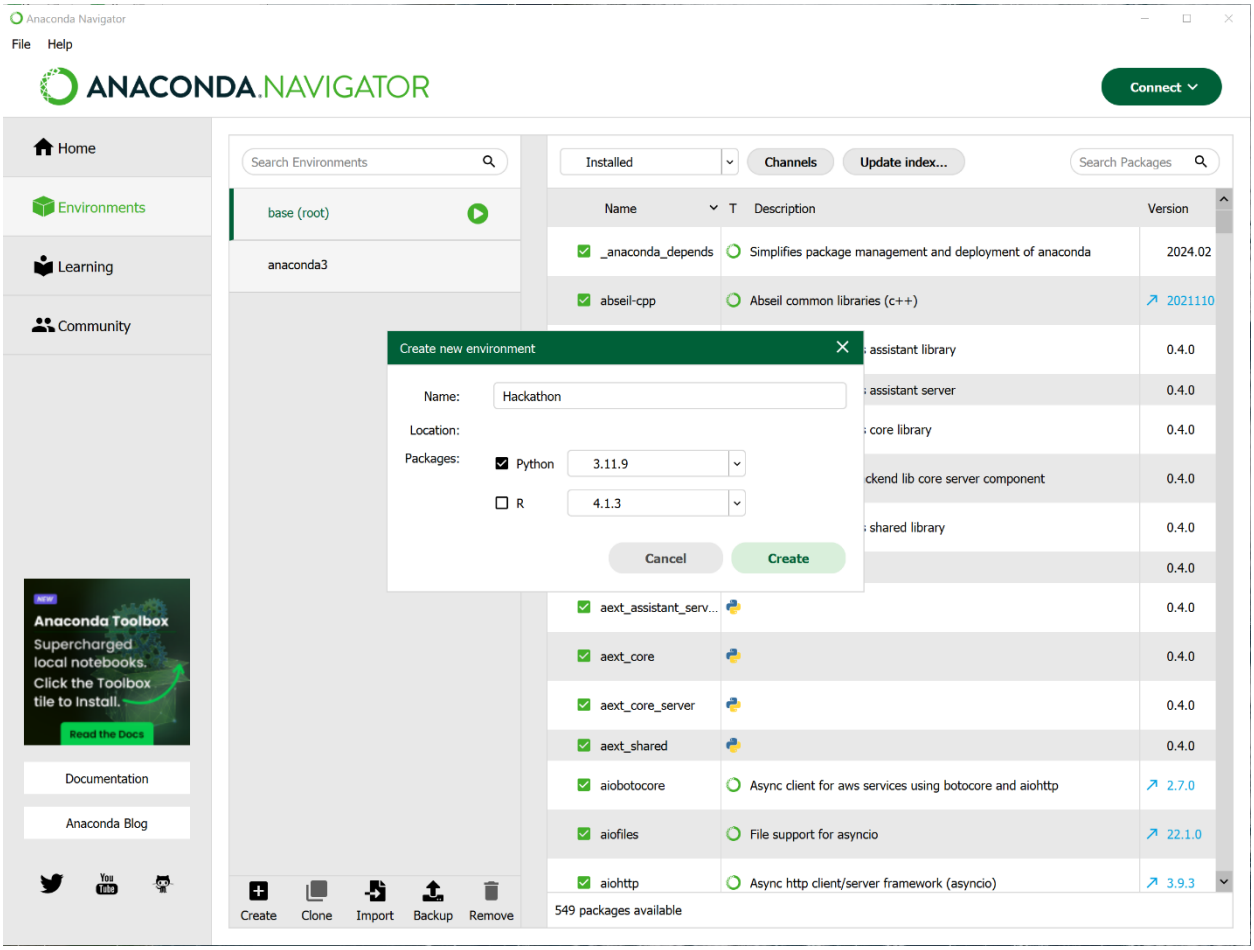

Figure 2: Creating a development environment in Anaconda

Select Environments from the menu on the left side of the navigator window. From the bottom select Create. Give your new environment a name; "Hackathon" as an example. Anaconda supports both Python and R; choose Python. You can specify a Python version. Most Python 3 releases are all reasonable choices; this example will use Python 3.11.9. The result is seen in Figure 2.

When the process completes, the new environment will be created and appear in the list of environments (Figure 3). The current active environment is marked with an equilateral triangle inside a circle.

The panel on the right shows the Python modules installed in that environment; the initial install shows just a small handful are present.

Installing the Spyder IDE

There are several integrated development environments (IDE) that can be used with Python; an excellent choice is Spyder, which is included with Anaconda.

Anaconda takes an interesting approach to installing and running Spyder. Spyder can be installed as a stand-alone tool, and then configured to use different Python environments.

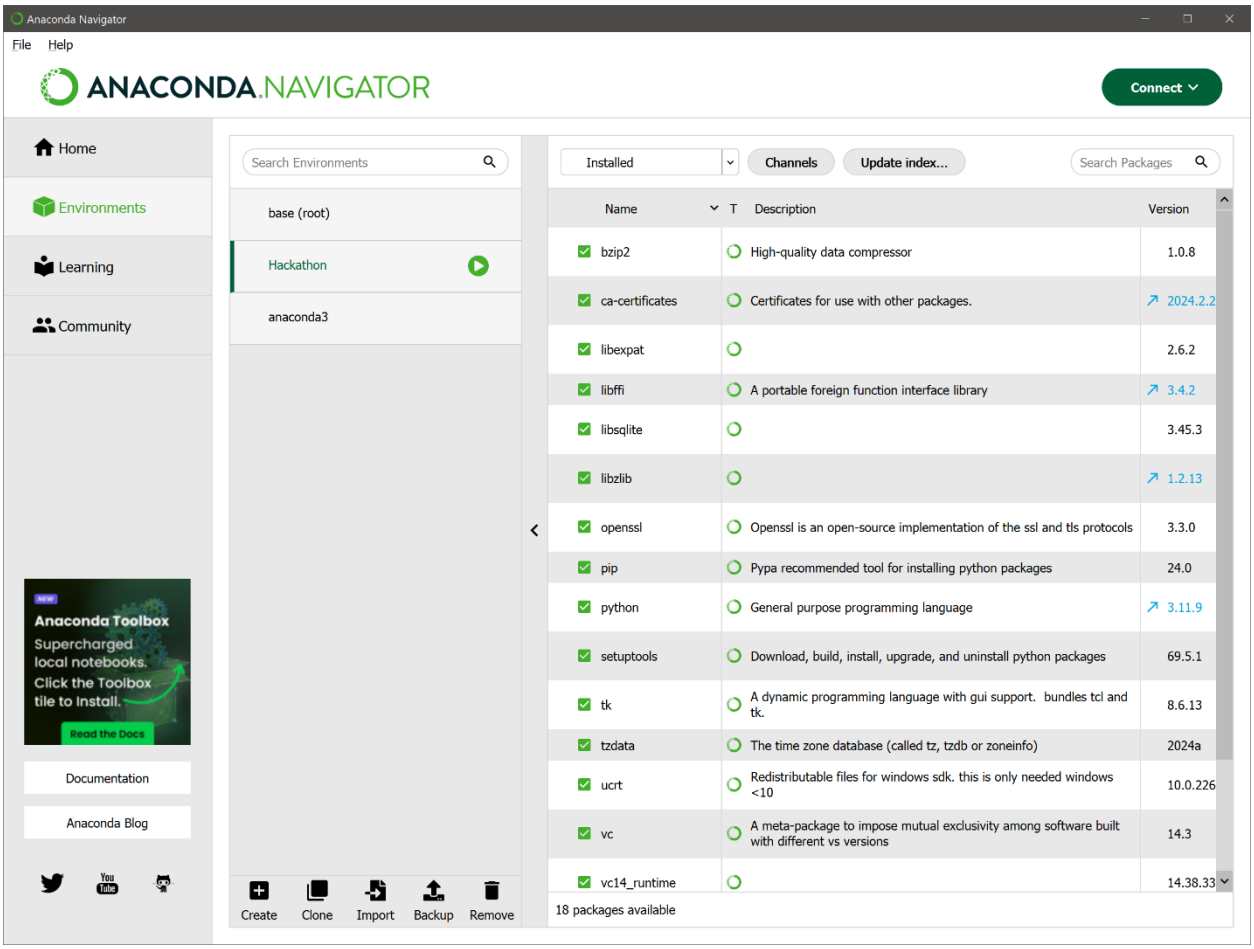

Figure 3: Anaconda Navigator environments

This all works, but to change the development environment, the user needs to know exactly where to make all of the manual configuration changes, both in Spyder, and in Python itself. By contrast, Anaconda automates this process.

Return to the home menu in Anaconda Navigator, however, change the channel from base (root) as seen in Figure 1 to the just installed Hackathon environment as seen in 3. Note that many of the applications that are installed in the base (root) environment have not been installed in the Hackathon environment.

Spyder is listed in the collection of available applications on Hackathon and can be installed by clicking on the Install button.

One issue that folks have had with Spyder, especially Spyder 5, are problems with the *fonts*. (Really!) Spyder 5 uses custom fonts, and Windows 10 has security settings that block them https://docs.microsoft.com/en-US/troubleshoot/windows-client/shell-experience/feature-to-block-untrusted-fonts. If you install Spyder 5 and try to run it, it may fail, silently. You can follow the Microsoft instructions to allow the fonts to be used, try to manually resolve the issue (there is plenty of help in Stack Overflow for this issue), or use

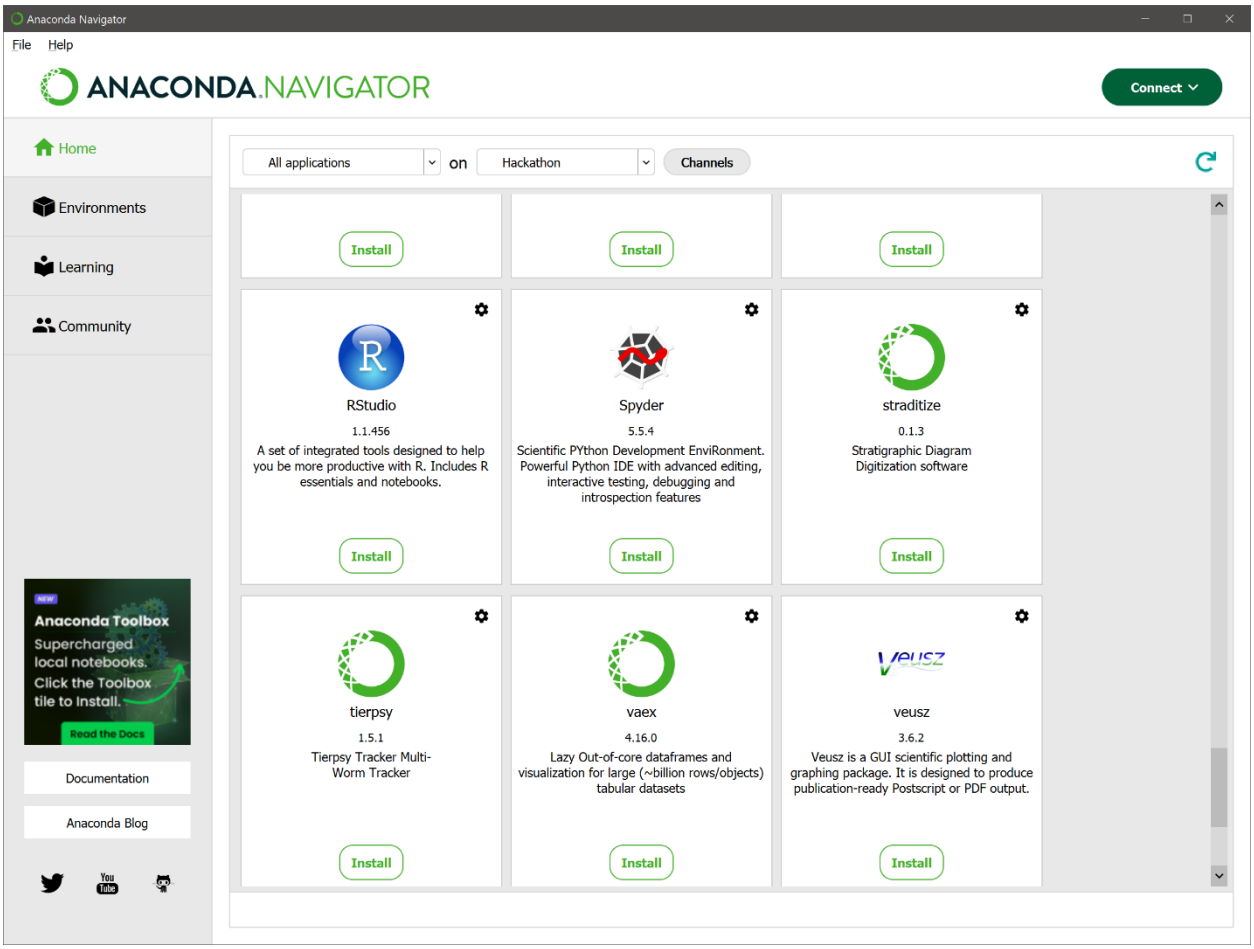

Figure 4: Applications in Anaconda Navigator

an older version of Spyder 3 or Spyder 4. Clicking on the gear icon in the top right of the panel for Spyder gives you an option to choose which version is installed.

Launch Spyder from the Anaconda Navigator using the Hackathon channel; the result is something like Figure 5.

A word about versions....

There are slight differences between versions of Python, and more substantial differences in the version of Spyder. I will be using Python 3.11.9 and Spyder 5.5.4 in this document. References to documentation will be to Python 3.11. When reading documentation, always take a moment to check the Python version.

Do not use Python 2.x. It has been deprecated, and there are enough differences between Python 2.x and Python 3.x that code in one version will not run in the other version.

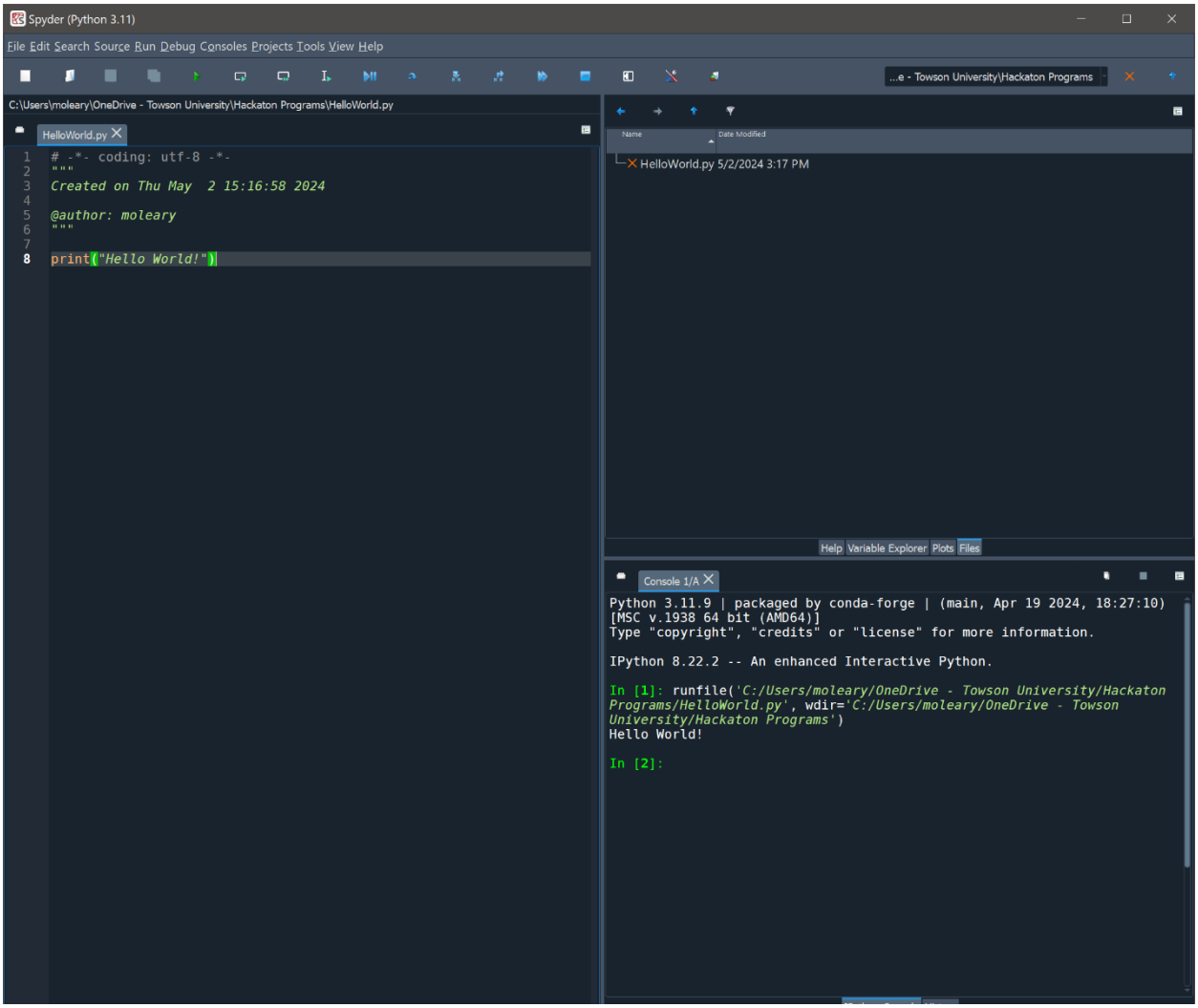

Figure 5: Spyder 5.5.4 running helloworld.py