

Jupyter Notebook

1. Installing Python and Setting up an Environment for Data Science Last Checkpoint: a few seconds ago (autosaved) Current Kernel Logo Python 3 File Edit View Insert Cell Kernel Help

the quandl package.

Installing Python and Setting up an Environment for Data Science

This document provides a guide for installing Python 3 onto your system and creating an environment suitable for data science work. It is important that you follow this guide precisely so that your system is setup to run all the code in this book. This document covers this setup for Windows, macOS, and Linux operating systems. A [\[video tutorial\]](#)^[-1] is available.

Conda

There are a number of ways to set up your system to do data science in Python. This tutorial relies upon a tool called [\[conda\]](#)^[0] which is both a **package manager** and an **environment manager**.

Package Manager

A [\[package manager\]](#)^[1] is a tool that installs, updates, and removes computer programs. For us, these computer programs will be third-party Python packages. A third-party Python package is any package that is not part of the Python standard library, which comes with every Python installation. There are many thousands of third-party packages available to be installed onto your system. Another popular package manager is **pip**, which existed long before conda and is the default package manager for new Python installations. Although pip is a good tool, we will not use it, as conda contains more features and resolves dependencies better.

Environment Manager

An environment manager is a tool that creates an environment (sometimes referred to as a virtual environment), a completely separate and isolated area of your computer with its own installation of Python and own third-party packages that are independent from any other Python installation on your machine.

Miniconda

Miniconda is a distribution of Python that includes conda and a small number of other packages that conda depends on.

Python Installation

Python comes preinstalled on macOS and most Linux operating systems, but it is not a good idea to use this preinstalled version for development as some critical system software might rely on it. It's also usually not the latest version of the software. Instead, we will install a completely new

version of Python in a different location. There are many ways to get Python such as directly from [Python.org][2], but for this book we will obtain it through the Miniconda distribution, which will also install conda.

Miniconda Download

[DOWNLOAD MINICONDA HERE][3] - Choose the installation for your operating system. Both Windows and macOS have graphical installers (.pkg file for macOS). macOS and Linux both have command line installers (.sh file). There will be instructions for both the graphical and command line installers, so you can choose either one. Many people will be aware of Anaconda, a different distribution of Python that contains hundreds of packages and other software. Both Anaconda and Miniconda are maintained by a company called Anaconda (formerly known as Continuum Analytics). They both install the same version of Python and conda. The reason I recommend installing Miniconda is that Anaconda installs many unnecessary packages and software that you will likely never use. This extra 'bloat' will amount to around 2GB of extra hard disk space. Any extra software provided by Anaconda may be installed from conda, so you needn't worry that you might be missing some special piece of software. You can get any software provided in the Anaconda distribution at a later date.

Keeping Anaconda

If you already have Anaconda you can either uninstall it or keep it. Even if you are happy with the current status, you might consider uninstalling it as there is quite a lot of excess software and it does not take too much effort to get a minimal clean installation. Use the next section to uninstall Anaconda. If you do not want to uninstall Anaconda, then do not install Miniconda. Instead, run the following from the command line:

```
conda update conda
```

This will update conda to the latest version which is necessary and important to ensure that your system is set up properly. You will still need to create the environment for this book, so please skip to the section **Creating a new environment just for data analysis**.

Uninstalling Anaconda

If you wish to uninstall Anaconda, navigate to the official [Uninstalling Anaconda][5] page. For macOS and Linux, use option B first and then complete the simple remove with option A.

Miniconda Installation

We will now continue with the Miniconda installation assuming you have downloaded the correct file for your operating system.

Windows

The name of the Windows file will begin with **Miniconda3-latest-Windows**. Start the setup and after agreeing to the license you will be given the choice to install for 'Just Me' or 'All Users'. It's best to select 'Just Me' as this will give you full control of the installation without needing to have

administrator rights to install new packages. ![][6] The next step will ask you for a file location for the installation. You should select the default location which will be `C:\Users\
<UserName>\Miniconda3` where `<UserName>` is the name of your user folder.

Add to PATH?

The next screen asks whether you'd like to add Anaconda (should say Miniconda) to the PATH and recommends that you do not do so. The main advantage of adding Miniconda to the path is to have access to it directly from the Command Prompt program. This is unnecessary as Miniconda provides a small program called **Anaconda Prompt** that does add the necessary file location to the PATH. Keep the defaults as they are and complete the installation. ![][7]

macOS

Graphical Installer

The graphical installer file will begin with **Miniconda3-latest-MacOSX** and end in **.pkg**. After agreeing to the license, you will be asked to choose to either 'Install for me only' or 'Install on a specific disk'. Choose to 'Install for me only' and select the default location which is `/Users/<UserName>/Miniconda3` where `<UserName>` is your specific user name. This will complete the installation.

Command Line Installer

The command line installer will end in **.sh**. Open up your terminal and navigate to the location of where you downloaded the installer and then run the following command. Make sure to use **bash** regardless of the shell you are using.

```
bash Miniconda3-latest-MacOSX-x86_64.sh
```

This will start a stream of text that you'll need to press **enter** to move through. You'll be prompted to agree to the license and whether to accept the default location for the installation which will be `/Users/<UserName>/Miniconda3`. Press ENTER exactly **once**. The installation will appear to have paused and tempt you to press enter again. Do not do this. Instead, just wait patiently. You will then be prompted to initialize Miniconda3. Enter 'yes' to complete the installation. Exit out of the shell.

Linux

If you aren't able to use a web browser should be able to download the installer with the following command:

```
wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
```

To begin the installation run the following command making sure to use **bash** regardless of the shell you are using.

```
bash Miniconda3-latest-Linux-x86_64.sh
```

This will start a stream of text that you'll need to press **enter** to move through. You'll be prompted to agree to the license and whether to accept the default location for the installation which will be `/home/<UserName>/miniconda3`. Press ENTER exactly **once**. The installation will appear to have paused and tempt you to press enter again. Do not do this. Instead, just wait patiently. You will then be prompted to initialize Miniconda3. Enter 'yes' to complete the installation. Exit out of the shell.

Python and Conda installation complete

After completing the steps above, you will have finished installing both Python and conda along with a small number of other Python packages.

Test Installation

Let's test that we have a successful installation.

Windows Users

Windows users must always begin by starting the program **Anaconda Prompt**. This program is easily found by tapping the windows key and then typing in the exact name 'Anaconda Prompt'. !
[][8]

macOS/Linux Users

Start your terminal program.

All users

Run the command `conda list` which will return the name, version, build number, and channel for each package currently installed. ![][9] This is a list of all the packages that come installed with the default Miniconda distribution. Notice that Python is considered a package. This might appear to be a large number of packages, but it is a fraction of what is installed with the full Anaconda distribution.

The PATH

All operating systems have something called a **PATH** which is a list of directories that the operating system looks through in order to find executable programs. The directories in Windows paths are separated by a semi-colon, while macOS and Linux directories are separated by a colon. Let's take a look at the path now.

- Windows users - `echo %PATH%`

- macOS/Linux - `echo $PATH` All operating systems should have the following two directories in their path that end with the following.
- `miniconda3/bin`
- `miniconda3/condabin` It is the executable programs within these folders that are available for us to use on the command line. Listing out these programs on my macOS machine yields the following:

```
ls /Users/Ted/miniconda3/bin
```

```
![][10]
```

Verifying the Installation

By default, Miniconda creates an environment with the name 'base' that has all the packages displayed from `conda list` installed. Notice that '(base)' has been prepended to the prompt to indicate that this environment is active. When an environment is active, it means that your Python code will be interpreted by the Python executable in that environment. Let's verify this by starting a Python interpreter by running the command `python`. `![][11]` We can verify the location of the Python executable by importing the `sys` library and fetching the `executable` attribute. `![][12]` While it is possible to get started doing data analysis within this environment, I recommend creating a completely separate environment.

A note on directory path

For the rest of this tutorial, I will use `miniconda3/bin` and `miniconda3/condabin` as shortcuts referring to the full path to these locations. If you followed the installation instructions from above, the full path will look similar on all operating systems. The full path for `miniconda3/bin` will be one of the following:

- Windows - `C:\Users\<UserName>\miniconda3\bin`
- macOS - `/Users/<UserName>/miniconda3/bin`
- Linux - `/home/<UserName>/miniconda3/bin`

Deactivating the base environment

By default, the base environment will always be active upon opening the terminal. Specifically, the `miniconda3/bin` directory will be added to your path and allow you to start Python and all the other programs listed above. In my opinion, this isn't good practice and it's better to explicitly activate the environment. We can change conda's configuration settings so that it does not automatically activate the base environment upon opening of the terminal. On the command line run the following:

```
conda config --set auto_activate_base false
```

Exit the shell, re-enter it, and output the path again. You should notice that only the `miniconda3/condabin` directory remains. The `miniconda3/bin` is no longer in the path. This command seems to have no effect on Windows Anaconda Prompt. Windows users will have to manually deactivate their base environment with `conda deactivate`. Also, notice that `'(base)'` is no longer prepended to the prompt.

Activating the base environment

You can reactivate the base environment with the following command:

```
conda activate base
```

This will prepend the `miniconda3/bin` directory back to your path and add `'(base)'` to the prompt. You can deactivate it again with the command:

```
conda deactivate
```

Creating a new environment just for data analysis

While it's possible to use the base environment to do all of our data science work, we will instead create a new environment where all of the packages are installed from the conda-forge channel. But, before we do that, it's important to understand what a conda channel is.

Conda Channels

A conda **channel** is simply a repository of Python packages. There are dozens (if not hundreds) of channels available each with their own collection of Python packages. Whenever you install a new package using conda, its contents will come from exactly one channel. By default, conda will only install from the **defaults** channel. You can verify that the defaults channel is the only one available by running the following command:

```
conda config --show channels
```

![][13] All channels have at least one URL available where the repository is located. You can find these URLs with the following command:

```
conda info
```

![][14] The above results are from my macOS. Linux and Windows channel URLs will look very similar. Notice that there are multiple URLs for this one channel. There's even a URL for R

packages, which seems bizarre, but conda is not a tool just for managing Python packages. It is a general purpose package manager that can work with any other programming language. Navigate to one of the URLs in a browser and you will see a list of the packages available to download. ![] [15] The defaults channel contains a hand-picked list from the team at Anaconda of popular and powerful packages to do scientific computing. However, there are many thousands of packages that exist that are not available in the defaults channel. This is where the conda-forge channel becomes important.

The conda-forge channel

Anaconda the company allows anyone to create a channel and will host these packages in the [Anaconda Cloud][16]. You can create an account right now and start your own channel with your specific collection of packages. [conda-forge][17] is the most popular channel outside of the defaults and contains [many more packages][18]. My recommendation, at the time of this writing, is to install packages only from conda-forge if possible and not from the defaults. The reasons for this are described in the conda-forge documentation reprinted below:

- all packages are shared in a single channel named conda-forge
- care is taken that all packages are up-to-date
- common standards ensure that all packages have compatible versions
- by default, we build packages for macOS, linux amd64 and windows amd64
- many packages are updated by multiple maintainers with an easy option to become a maintainer
- an active core developer team is trying to also maintain abandoned packages One of the main reasons to use a single channel such as conda-forge is the consistency it provides with package compatibility. For packages that have components written in a compiled language like C, compatibility improves when they are all compiled from the same base C library.

Create a new environment

It's finally time that we create our new environment that we will use for data science. There are a few different ways to successfully accomplish this. One way will be shown now, with other alternative ways shown later on.

Create an empty environment

Let's create an environment with the name 'minimal_ds' that has no packages in it, not even Python.

```
conda create -n minimal_ds
```

Confirm the creation and notice that its location in your file system will be `miniconda3/envs/minimal_ds`. Any downloads for the environment will be located here. Activate the environment with:

```
conda activate minimal_ds
```

By default, this environment will install packages from the defaults channel.

Add the conda-forge channel

Let's add the conda-forge channel as an option for just this environment. The `--env` option ensures that conda-forge is added only to our currently active environment.

```
conda config --env --add channels conda-forge
```

Running this command will create a configuration file named `.condarc` in the environment's home directory (`miniconda3/envs/minimal_ds/`). You can verify this by outputting its contents to the screen.

```
cat miniconda3/envs/minimal_ds/.condarc
```

Instead of outputting the configuration file's contents, we can use the following command which will show the same thing.

```
conda config --show channels
```

![][19] Adding a channel will not remove any previous channels. Instead, it will become the first channel that conda looks to find packages. Currently, if it cannot find a package in conda-forge it will then look in defaults for it. But, if the same package exists in both, then it will choose to install it from the channel with the newest version. For instance, if conda-forge has pandas version 0.23 and the defaults has version 0.24 then conda will install pandas 0.24 from the defaults channel. This behavior is unintuitive to me and it makes more sense to always use the channel that appears first in the channels list regardless of the version. Conda gives us a way to change this with the following command:

```
conda config --env --set channel_priority strict
```

Let's verify the configuration change.

```
conda config --show channel_priority
```

The `.condarc` file has also been updated with the same information. Changing this setting will cause conda to always install packages from conda-forge unless they don't exist in it at all and then look to the defaults channel. There are some packages that only exist on the defaults channel. This behavior is changing when conda 4.7 is released. At the time of this writing, conda

is on version 4.6. The `channel_priority` variable will be defaulted to 'strict' so this step can be skipped if you are on version 4.7 or later. Check your version of conda by running `conda -V`.

Installing the packages

We are finally ready to install packages into our new environment. Conda will only look in the conda-forge channel unless a package is missing and then turn to defaults. Personally, a minimal data science environment has numpy, scipy, pandas, scikit-learn, and matplotlib along with the newest stable version of Python (which is 3.7 at the time of this writing). It will also have jupyter notebooks available. Let's install these packages now.

```
conda install pandas scikit-learn matplotlib notebook
```

Notice that Python, numpy, and scipy weren't explicitly included in the list of packages to install. These packages are dependencies of at least one of the included packages and will be installed along with many other dependencies. Let's take a look at the packages to be installed before confirming. `![]`[20] Conveniently, this list shows the name of the package, the version number, the size, and the channel. If the last column has a blank value, it indicates that it is being sourced from the defaults channel.

Other packages to install

There are a few other packages used during the book that you will want to install. Specifically, we will install seaborn, a visualization library, sqlalchemy (a library to connect to SQL databases), and jupyter_contrib_nbextensions, a library for adding more functionality to the notebook. They are available on conda-forge. Make sure the `minimal_ds` is activated before running the following install command.

```
conda install seaborn sqlalchemy jupyter_contrib_nbextensions
```

Only installing from conda-forge

Our current setup allows for packages not found on conda-forge to be searched for on defaults. It may improve compatibility issues to only use the conda-forge channel. To ensure that packages only come from conda-forge, you'll have to specify the channel name (with the `-c` option) together with the `--override-channels` option.

```
conda install -c conda-forge --override-channels <package_name>
```

Conda always asks for confirmation of the installation after showing you the plan, which allows you to verify the channel and package versions before proceeding.

Installing packages not in conda-forge or defaults

As discussed, if a package does not exist in conda-forge, it will be searched for in the defaults channel. If it does not exist in the defaults channel, then the installation will fail with an error message. You can specify a different channel to use, as long as you know its name. The easiest way to find the channel name of a package is to visit [\[anaconda.org\]\[21\]](https://anaconda.org) and search for it at the top of the page. For instance, `plotly_express` is not available in either conda-forge or defaults. Searching for it on anaconda.org reveals its channel as `plotly`. Let's install it by specifying the channel.

```
conda install -c plotly plotly_express
```

Note that the package `plotly` is a dependency and will also be installed from the `plotly` channel. If you search for the package `plotly`, you will see that it is available on conda-forge, but in this instance will be installed from the `plotly` channel and not with conda-forge. Any channels provided to the `-c` option will take precedence over the channels in the `.condarc` file.

Installing Quandl for financial data with pip

A few times within the book, we will read financial data using [\[Quandl\]\[23\]](#) package. Although this package is available on both conda-forge, one of its dependencies has not been updated at the time of this writing. Installing it from the defaults channel results in a host of other compatibility issues. Instead, we will install it with **pip**, the original package manager for Python. Make sure `minimal_ds` is activated and run the following command:

```
pip install quandl
```

Quandl registration and API key

Once it's installed, you'll need to register for an account at the [\[Quandl home page\]\[23\]](#). Once, you've registered for an account, visit [\[your account profile\]\[24\]](#) and copy your API key. Paste the key in the file `api_key.txt` which is in the top-level directory for this book. It should be the only text in the file. Now, you'll be able to run any code in the book that uses the `quandl` package.

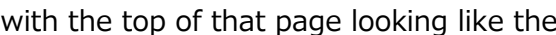
Environment Setup Complete

Your environment should now be set up correctly to do data science using Python for this book. It contains a minimal number of the most common and useful Python packages and will use conda-forge as its primary channel.

Other Considerations

There are a few other items that are worthy of discussion not mentioned above.

Verify that Jupyter Notebooks execute in correct environment

Although we have created our own environment with the ability to create Jupyter Notebooks, we are not guaranteed that they will execute Python in the same environment that they were launched. For instance, if we launch a Jupyter Notebook from the `minimal_ds` environment, it is possible that we are executing Python from the base environment. This is quite surprising behavior as you would expect environments to be isolated from one another, but this isn't quite the case. We need to verify that executing notebooks launched from the `minimal_ds` environment execute Python from the `minimal_ds` environment and not from anywhere else. With the `minimal_ds` environment activated, run the command `jupyter notebook`. This will start a Jupyter Notebook server running from your localhost at the default port 8888. Your default web browser should open up with the top of that page looking like the following image.  Click the **New** button on the right-hand-side of the page and start a new 'Python 3' notebook. In the first cell of the notebook, write the the following two lines of code:

```
import sys
sys.executable
```

Execute this code by pressing **shift + enter**. The result should return a path to the environment Python (`miniconda3/envs/minimal_ds/bin`). If it returns the location for the base environment (`miniconda3/bin`) then you aren't executing Python from your environment. The cause of this is a 'User' kernel that is masking the environment kernel. Exit out of the Jupyter Notebook by pressing `ctrl + c + c` in your terminal. Then run the following command to see the list of kernels:

```
jupyter kernelspec list
```

Check to see if your python3 kernel is indeed a User kernel. [Find the default locations][22] for User kernels for your OS. The User kernel has the highest precedence over the Environment and System kernels. Yes, that's right, even when you have an active environment, the User kernel takes precedence and allows you to execute Python from other environments. If you indeed have a User kernel, then I recommend removing it with the following command:

```
jupyter kernelspec remove python3
```

Rerun the command `jupyter kernelspec list` and you will see the Environment kernel with the same name (`python3`). You should now launch another Jupyter Notebook and verify that the Python executable is located in the active environment.

Alternative Environment Creation

Above, we created an empty environment first and then installed the packages with a separate command. We did this so that we could add the conda-forge channel and set its priority. It is

possible to do this in a single step.

```
conda create -n minimal_ds -c conda-forge --strict-channel-priority pandas
scikit-learn matplotlib notebook
```

An issue with this method is that no configuration file for the environment is created. You would have to specify the option to use the conda-forge channel and strict channel priority for each new package installed. Of course, you could set the configuration file after the package installation. Another method for creating an environment is with a text file usually given the name `environment.yml`. The contents of the file contain the name, channel(s) and packages. The contents of the file that would have created our environment would look like this:

```
name: minimal_ds
channels:
  - conda-forge
  - plotly
dependencies:
  - pandas
  - scikit-learn
  - matplotlib
  - notebook
  - seaborn
  - sqlalchemy
  - jupyter_contrib_nbextensions
  - plotly_express
  - pip:
    - quandl
```

We would then run the command:

```
conda env create -f environment.yml
```

A major issue with this method is that there is no (current) way to set the `channel_priority` to strict and like the previous one no configuration file for the environment will be created.

Updating packages

All the popular data science packages are under constant development and make new releases from time to time. You can update all of the packages at once with the following command:

```
conda update --all
```

Before the update happens, conda will show you a list of all the packages that it will update. This allows you to view all the latest versions of each package and make a decision on whether to

update or not. To update an individual package run `conda update packagename`.

Updating conda

It's also important to update the tool conda itself from time to time. Before updating conda, you'll need to know if you installed conda in the current environment. If you followed the instructions on this page, then you won't have conda installed in the `minimal_ds` environment. It'll only be installed in the `Miniconda/condabin` directory. Therefore, to update conda, deactivate the `minimal_ds` environment with `conda deactivate`. Then run:

```
conda update conda
```

If you do have conda installed in your environment then you can update both installations with the same command. Just activate the environment first and rerun the above command.

Summary of Steps



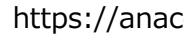



There were a lot of text that separated the steps in this tutorial, so below is a summary of just the commands.

1. [Install Miniconda][3] for your OS with the default settings
2. Prevent the base environment from automatically activating `conda config --set auto_activate_base false`
3. Create an empty environment `conda create -n minimal_ds`
4. Activate the environment `conda activate minimal_ds`
5. Add conda-forge as first channel `conda config --env --add channels conda-forge`
6. Ensure that conda-forge is used if the package is available `conda config --env --set channel_priority strict` (Not necessary for conda version 4.7+)
7. Install packages `conda install pandas scikit-learn matplotlib notebook seaborn jupyter_contrib_nbextensions`
8. Install packages not in conda-forge/defaults `conda install -c plotly plotly_express`
9. Install Quandl with `pip install quandl`

Using the book with Jupyter Notebooks

In my opinion, the best way to work through the contents of this book is within Jupyter Notebooks. They allow you to read the material, run the code, and write solutions to the exercises and projects all in one place. If you'd like to use the Jupyter Notebooks for this book, make sure that you have activated the `minimal_ds` environment and have navigated to the directory where the contents of this file are located. By default, the folder is titled, "Master Data Analysis with Python by Ted Petrou". Launch the Jupyter Notebook with the following command:

```
jupyter notebook
```

This will start a Jupyter Notebook server and open up a new tab in your web browser. The contents of your file system of the directory where you ran the `jupyter notebook` command will be shown on the page in your browser. All the notebooks for the book are contained in a folder titled **Jupyter Notebooks**. This document is available as the first notebook in the **Environment Setup and Jupyter Notebooks** folder titled **01. Installing Python and Setting up an Environment for Data Science**. To launch a specific notebook, simply click its title. All notebooks have file names that end in `.ipynb` which references their origins as IPython Notebooks. New notebooks will open in their own browser tab. You may now move on to the second chapter, **02. Introduction to Jupyter Notebooks**. [-1]: <https://youtu.be/ePS6II4SEg> [0]: <https://docs.conda.io/projects/conda/en/latest/> [1]: https://en.wikipedia.org/wiki/Package_manager [2]: <https://Python.org> [3]: <https://docs.conda.io/en/latest/miniconda.html> [4]:  [5]: <https://docs.anaconda.com/anaconda/install/uninstall/> [6]:  [7]:  [8]:  [9]:  [10]:  [11]:  [12]:  [13]:  [14]:  [15]:  [16]: <https://anaconda.org/> [17]: <https://conda-forge.org/> [18]: <http://conda-forge.org/feedstocks/> [19]:  [20]:  [21]: <https://anaconda.org> [22]: <https://jupyter-client.readthedocs.io/en/stable/kernels.html#kernel-specs> [23]: <https://quandl.com> [24]: <https://www.quandl.com/account/profile>