Environment Setup and Jupyter Notebook Extension Instructions

This document contains instructions for setting up your environment and adding Jupyter Notebook extensions for the bootcamp.

Environment Setup

We will be using the conda command line tool to create a separate environment for our code. An environment is an isolated location in your file system using specific versions of python and other libraries to help make results easily reproducible.

Create a new directory

In your file system, create a directory titled 'Dunder Data Bootcamp' that will hold all the material for the bootcamp.

Create a conda environment

In the 'Dunder Data Bootcamp' directory, create a file called 'environment.yml' and copy and paste the following into that file.

```
name: dunder_data_bootcamp
channels:
  - conda-forge
dependencies:
- python=3.7
- pandas
- jupyter
- scikit-learn
- seaborn
- pandas-profiling
- jupyter_contrib_nbextensions
```

Command to create new environment

On your command line, navigate to the directory where the 'environment.yml' file is located and run the following command.

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

The above command will take some time to complete. Once it completes, the environment `dunder_data_bootcamp` will be created with the above libraries and all of their dependencies.

List the environments

Run the command `conda env list` to show all the environments you have. There will be a `*` next to the active environment, which will likely be `base`, the default environment that everyone starts in.

Activate the dunder_data_bootcamp environment

Creating the environment does not mean it is active. You must activate in order to use it. Use the following command to activate it.

```
conda activate dunder_data_bootcamp
```

You should see `dunder_data_bootcamp` in parentheses preceding your command prompt. You can run the command `conda env list` to confirm that the `*` has moved to `dunder_data_bootcamp`.

Run python code from the dunder_data_bootcamp environment

Let's further confirm that we are executing code within the `dunder_data_bootcamp` environment. Start the IPython shell (ipython is a jupyter dependency and was installed during environment creation) by running the following command:

```
ipython
```

Within the IPython shell, you can use the `sys` module to find the location of the python executable. Run the following commands now.

```
In [1]: import sys

In [2]: sys.executable
Out[2]: '/Users/Ted/anaconda3/envs/dunder_data_bootcamp/bin/python'
```

The executable location should end with `anaconda3/envs/dunder_data_bootcamp/bin/python`.

Ensuring that Jupyter Notebooks run from this environment

We will now verify the location of the python executable within a Jupyter Notebook. You might be wondering why this is necessary since we did nearly the same thing in the previous step. Surprisingly, Jupyter Notebooks are able to run any python executable from any environment. Your environment might not be set up properly to run the python installation from the active environment.

Exit out of the IPython shell that you started above and ensure you are still in the `dunder_data_bootcamp` environment. Launch a jupyter notebook with the following command:

```
jupyter notebook
```

After the home page finishes loading in your browser, create a new notebook by clicking the **New** button in the top right hand side of the page. Choose 'Python 3' in the dropdown menu. Within the notebook's first cell, run the same commands as before:

```
import sys
sys.executable
```

One of two possibilities can happen

If the value outputted from `sys.executable` is the same as it was from the IPython shell, then your environment is set up properly and you are good to go and may skip down to the Deactivate environment section.

If the value outputted from `sys.executable` ends with the following:

```
anaconda3/bin/python
```

then you have more work to do and the code you are executing is from the base environment and not from `dunder_data_bootcamp`.

Deleting the "User" Kernel

Exit out of Jupyter and return to the command line. We need to delete a Kernel, a program that "runs and introspects the user's code".

As mentioned earlier, Jupyter Notebooks allow you to run any python executable regardless of the environment that it was launched from. It uses kernels to run these different python executables. Jupyter has three different locations in your file system to find the kernel to run. These locations are known as **User**, **Env**, and **System**. Open up the official documentation to find the locations for your operating system.

One issue is that kernels can have the same name. In fact, every single environment that you install jupyter in will have its own Env 'python3' kernel. But, the User kernels have the **highest priority** and will be run if there is a name collision with the Env kernels.

It's likely that you have a User 'Python 3' kernel that is interfering with your Env 'python3' kernel. Let's list the currently available kernels with the following command:

```
jupyter kernelspec list
```

You will see the name of the kernels followed by its location. You should see that the 'python3' kernel is located in the User location. On my Mac, it shows the following:

```
/Users/Ted/Library/Jupyter/kernels
```

This kernel takes precedence and will be run regardless of the environment you launch your notebook from. This is extremely confusing and frustrating in my opinion. It would make more sense if the only available kernels were from the active environment.

Running the above command will actually not list kernels with the same name that have lower priority. For instance, since you installed jupyter in the `dunder_data_bootcamp` environment, it will have its own 'python3' kernel in the Env location. It is not going to be shown in the list above. Instead, you can list this kernel manually. We can find the location of the Env kernels again by looking at the documentation. On my Mac, the following command lists all my kernels in the `dunder_data_bootcamp` environment.

```
ls /Users/Ted/anaconda3/envs/dunder_data_bootcamp/share
```

This returns 'python3', a kernel with the same name returned from `jupyter kernelspec list` but in a different location. For personal machines, there is usually not a need to have a User 'python3' kernel so we can delete it. The following command will delete the 'python3' kernel with the highest priority. Only run it if you have a User 'python3' kernel.

```
jupyter kernelspec remove python3
```

Rerunning `jupyter kernelspec list` should now return the Env 'python3' kernel. Check that it does so now.

Verify from a Jupyter Notebook

We should still verify that the python executable location has changed within the notebook. Start by running `jupyter notebook` again and starting the same notebook that you ran the command `sys.executable` in. Run this command again and verify that the python executable is now located in the `dunder_data_bootcamp` environment.

No need to do this in future environments

You will not have to repeat this procedure when creating new environments that have jupyter installed in them. The User 'python3' kernel is permanently deleted and jupyter now uses the Env location as its default. Every new environment that you create with jupyter will have its own 'python3' kernel that will be the default.

Alternative - Create a new kernel with a different name

Instead of deleting the User 'python3' kernel, you may create a new User(or Env) kernel with a distinct name. The following command creates a new User kernel named 'dunder_data_bootcamp_python3' displayed as 'Python 3 (dunder_data_bootcamp)'.

```
python -m ipykernel install --user --name dunder_data_bootcamp_python3 --
display-name "Python 3 (dunder_data_bootcamp)"
```

Since the above is a User kernel, it will be available from any environment. To run a notebook with this kernel, select it from the dropdown menu from the 'New' button. You can instead install an Env kernel by changing the `--user` option to `--sys-prefix`.

Deactivate environment

You should only use the `dunder_data_bootcamp` environment for this tutorial. When you are done with this session, run the command `conda deactivate` to return to your default conda environment.

Jupyter Notebook Extensions

A variety of extra functionality is available to notebooks through Jupyter Notebook extensions. These were installed during environment creation with the `jupyter_contrib_nbextensions` library.

Verify the NBextensions tab is showing

Activate the `dunder_data_bootcamp` environment and launch a Jupyter Notebook from the command line with `jupyter notebook`. Once the home page opens in the browser, look at the top of the page. You should see a tab that's title **NBextensions**.

If you do not see it, exit out of Jupyter by pressing `ctrl + c` twice while on the command line. The `dunder_data_bootcamp` should still be active. Run the following command:

```
jupyter nbextensions_configurator enable --user
```

Rerun `jupyter notebook` and confirm that that the **NBextensions** tab is available.

Enable Skip-Traceback Extension

Click on the **NBextensions** tab to reveal all the available extensions. These all provide additional functionality within the notebook. Clicking on an extension name will reveal some documentation and options on the bottom half of the page.

Activate the Skip-Traceback extension by checking it's box. Look below to the see the documentation and options for the Skip-Traceback extension. Click the checkbox that begins with 'add a button to the toolbar...'.

This extension is very useful as it allows you to collapse the very long tracebacks that pandas produces and only shows the error message, which is the most important part.