Virtual Environments

As is per the norm with Python applications, you'll want to create a virtual environment, which is an isolated working copy of Python. This is useful for a couple reasons:

1. Typically, when you load packages in Python, you're loading them across your entire system. A virtual environment allows you to set specific package versions within the environment. This is useful if you need different versions of Python or packages for different projects.
2. It helps with reproducible code! This is very valuable to ensure correct dependencies are loaded, such that if you review the code years down the road, the scripts should still run even if the package versions have since been updated. This is also useful for collaboration and code review, since the reviewer will know which libraries to install.

With `venv`

Create the directory and then navigate to it:

```
mkdir projectfolder
cd projectfolder
```

Then create a virtual environment with your chosen `projectname`. `source projectname/bin/activate` can be run to activate the virtual environment.

```
python -m venv projectname
source projectname/bin/activate
```

From here you can install packages as usual with `pip` or `conda`.

To deactivate the virtual environment, you can run `deactivate`.

With Anaconda

You can also create a virtual environment with `conda`.

```
conda create -n projectname
```

You can specify the python version as well.

```
conda create -n myenv python=3.6
```

After you have created the enviroment, you can activate it by typing:

```
conda activate myenv
```

To deactivate the environment you can type `conda deactivate`.

Keeping the environment consistent

In order to keep your environment consistent, it’s a good idea to "freeze" the current state of the environment packages. To do this, run:

```
pip freeze > requirements.txt
```

This will create a requirements.txt file, which contains a simple list of all the packages in the current environment, and their respective versions. You can see the list of installed packages without the requirements format using `pip list`.

To load the requirements:

```
pip install -r requirements.txt
```

This can help ensure consistency across installations, across deployments, and across developers.

Adding a virtual environment to Jupyter notebook

Now inside this virtual environment, we need to create a kernel that contains all the requirements, so that we can use this environment with a Jupyter notebook:

```
pip install ipykernel
ipython kernel install  --user --name=projectname
```

Then, load the requirements:

```
pip install -r requirements.txt
```

After the kernel is ready, it can be selected in your Jupyter notebook from the menu `Kernel -> Change Kernel`. Now you can start Jupyter notebook as per the usual.