Working with conda

I would suggest conda as package management system when working with python. For compiling software the system-installation of python should be used. This prevents mismatches between package versions, since programmers know what versions e.g. come with a Linux distro. Both installations can exists side-to-side with each other. Click here for installation instructions. Miniconda is a minimal installation, use it when you don't have much space available or know which packages you want to install. Anaconda includes a large collection of packages. During installation I would recommend choosing not to activate conda by default in every terminal session. You can activate your conda environment after installation as follows:

```
$ conda activate
(base) $
```

This will set all necessary paths. While inside of a conda environment it can be closed with

```
(base) $ conda deactivate
$
```

Furthermore, virtual environments are a useful tool to keep only the dependencies for a project that you need, since different projects might require different versions of the same package. This makes managing several projects easier. To create a new virtual environment use the following command:

```
$  conda create --name myenv
```

myenv can be replaced by the desired name. You can enter a specific env with

```
$ conda activate myenv
(myenv) $
```

Packages can be installed with

```
(myenv) $ conda install ...
```

By default conda will install the package in the active environment. With the -n flag a user can install packages in an environment without activating it. Same goes for commands like conda update, e.g.

```
$ conda update -n myenv --all
```

Will update all packages in myenv. To show all avlaible environments one can use

```
$ conda env list
```

To see all installed packages use

```
(myenv) $ conda list
```

Packages in conda come in channels from which they can be installed. The -c flag during installation can be used to mark a specific channel. A popular channel for example is conda-forge. There one can find many useful packages like pyroot and cuda-related packages. There can be a list of different channels, these are usualy given in decreasing priotiy. conda will try to install a package and all dependencies from the channel with highest priority first. To list all active channels use

```
(myenv) $ conda config --show channels
```

To add the channel conda-forge in the virtual environment use

```
(myenv) $ conda config --add channels conda-forge
```

I would recommend adding the following channel as highest priority to get the official pytorch releases

```
(myenv) $ conda config --add channels pytorch
```

If you want to delete an environment use the following command

```
$ conda env remove --name myenv
```

More detailed guides for working with virtual environments and channels can be found on the conda webpage.

Jupyter Notebook

Jupyter notebooks are a great resource for testing. Data has to be loaded only once and afterwards one can at leisure adjust plots or test the functionality of a function. When working with virtual environments one could install the notebook everywhere and use it from that environment, but it is also possible to install and setup the notebook just once and afterwards use kernels from every virtual environment with that single installation.

Install conda and activate your base environment or create a environment for the notebook. Then use the following command to install jupyter notebook

```
(base) $ conda install notebook
(base) $ conda install nb_conda_kernels
```

Additionally one can install Notebook extensions. These can be useful, but are not necessary

```
(base) $ conda install jupyter_contrib_nbextensions
```

To be able to use kernels from different virtual environments in the Jupyter Notebook one needs to install one additional package. Activate the virtual environment and use

```
(myenv) $ conda install ipykernel
```

Now one should be able to choose a kernel from the virtual environment for a notebook in Jupyter. Also when using new to create a new notebook one should be able to choose between kernels from the different environments. The package ipykernel has to be installed in every environment that the Jupyter Notebook should have access to.

Workin with the example scripts