To create the environment for minimal functioning python data analysis needs, which will probably / hopefully include all you python modules you'll need for analyzing, downloading and plotting model data, and using most of the scripts I've made for the NOPP project so far, do the following steps:

##To install:

#In linux (e.g., Poseidon) / WSL:

wget "https://github.com/conda-forge/miniforge/releases/latest/download/Miniforge3-$(uname)-$(uname -m).sh"

bash Miniforge3-$(uname)-$(uname -m).sh

#In Windows:

Just go straight to the following website in a browser:

conda-forge/miniforge: A conda-forge distribution. (github.com)

And click on, to download, the Windows x86_64 install package link under Miniforge 3 (**not** Miniforge-pypy3).

Then, start up a command prompt (search 'cmd', click on Command Prompt icon), and run the following from the directory you downloaded the install package into (if Downloads, 'cd Downloads' first):

start /wait "" Miniforge3-Windows-x86_64.exe /InstallationType=JustMe /RegisterPython=0 /S /D=%UserProfile%\Miniforge3

##To build the environment used:

First, do an update, just in case something changed since the last install package was built.

(base) <prompt> conda update -n base conda

Then, build the coawst env:

(base) <prompt> conda env create -f min_coawst.yml

(base) <prompt> conda activate coawst1

and then add roguewave and wavespectra for scripts that plot 2d wave spectra

(coawst1) <prompt> pip --no-cache-dir install wavespectra

(coawst1) <prompt> pip --no-cache-dir install roguewave

[where <prompt> is whatever your normal shell prompt is, and (coawst1) is before the usual prompt, to show that you are in that conda environment]

#------

Lastly, if you need other python modules, generally first try from **outside** the coawst1 env:

(base) <prompt> conda install -n coawst1 <module name>

And if that doesn't work, then activate the environment (conda activate coawst1), and try pip

Pip install <module name>

The wavespectra and roguewave modules seem to not be part of conda or conda-forge's base package list yet, so we used 'pip' above. It's a separate python module installer from conda.

One such module that you might need sometimes is one with the equations of state, thermal expansion or haline contraction coefficient functions, etc. Seawater is a decent one, though older (uses EOS80), so

(base) <prompt> conda install -n coawst1 seawater

There are other google-able options too.

#####USAGE

IPYTHON

For the most ‘matlab’-like environment to work within, I recommend using Ipython at a terminal (like in MobaXterm or something similar). It has tab-completion, and the member functions and attributes of an object are tab-searchable after putting a ‘dot’ after the object. Otherwise, you can set up Jupyter notebooks (see below). Jupyter notebooks are the most popular way to use python scientifically nowadays (it seems), but I really don’t think it’s just always better. It has plusses and minuses, as does terminal Ipython. Both use much the same syntax.

For a sample ipython_config.py file to put in your .ipython/profile_default directory (off of your home directory, whether windows or linux), copy the following lines - down to the #----- into a file called ipython_config.py.

Configuration file for ipython.

c = get_config()

Pre-load matplotlib and numpy for interactive use, selecting a particular

matplotlib backend and loop integration.

#c.InteractiveShellApp.pylab = 'auto'

Configure matplotlib for interactive use with the default matplotlib backend.

#c.InteractiveShellApp.matplotlib = 'auto'

List of files to run at IPython startup.

c.InteractiveShellApp.exec_files = []

Make IPython automatically call any callable object even if you didn't type

explicit parentheses. For example, 'str 43' becomes 'str(43)' automatically.

```
# The value can be '0' to disable the feature, '1' for 'smart' autocall, where
# it is not applied if there are no more arguments on the line, and '2' for
# 'full' autocall, where all callable objects are automatically called (even if
# no arguments are present).
c.TerminalInteractiveShell.autocall = 1

# Start logging to the default log file.
c.TerminalInteractiveShell.logstart = True

c.InteractiveShellApp.exec_lines = [
        'import datetime',
        'import numpy as np',
        'from numpy import *',
        'from matplotlib.pyplot import *',
        'import matplotlib.pyplot as plt',
        'plt.ion()',
        'import scipy as sp',
        'import netCDF4 as nc4',
        'import xarray as xr',
        'import pandas as pd',
        'from glob import glob',
        'from datetime import datetime as dt',
        'from scipy.io import loadmat',
        'import rioxarray as rio',
```

```
'import cartopy',

'from cartopy import crs',

'from importlib import reload',

'print "Welcome to the SciPy Scientific Computing Environment."'

]

#--------
```

These modules will be loaded by default, with the ‘as xx’ part at the end of many import statements being a short name for the module. These short names are my working names for these modules for over a decade now, but you can change them to something else if you like. Just check to see if another desired name is used by python first by typing it and seeing if it returns “name ‘xxx’ is not defined” or not. Also, a lot of matlab-named functions are available and ‘exposed’ to the namespace due to the lines 'from numpy import *' and 'from matplotlib.pyplot import *'. If you don’t like this, you can remove those lines. They do lead to problems in command-line scripts, though, and if you are uncertain to what module, say, ‘plot’ belongs to, you can type plot at a prompt and hit enter inside of ipython, and it will return:;

Out[2]: <function matplotlib.pyplot.plot(*args: 'float | ArrayLike | str', scalex: 'bool' = True, scaley: 'bool' = True, data=None, **kwargs) -> 'list[Line2D]'>

So, it belongs to pyplot in matplotlib (in case you need to hardcode it sometime, like by

import matplotlib.pyplot as plt

(and then)

plt.plot(...)

Also, this whole ipython_config.py file can be found in the same directory as this file.

Other Helpful Tips

You can add things like the following aliases to your .bashrc file to make things a little shorter to type:

alias ipy='ipython --profile foo' #replace foo with the name of your profile

```
alias lh='ls -lhasrt'

alias cac= ' conda activate coawst1'

alias cde= 'conda deactivate'
```