Thursday, December 3, 2020 | **Class #17**

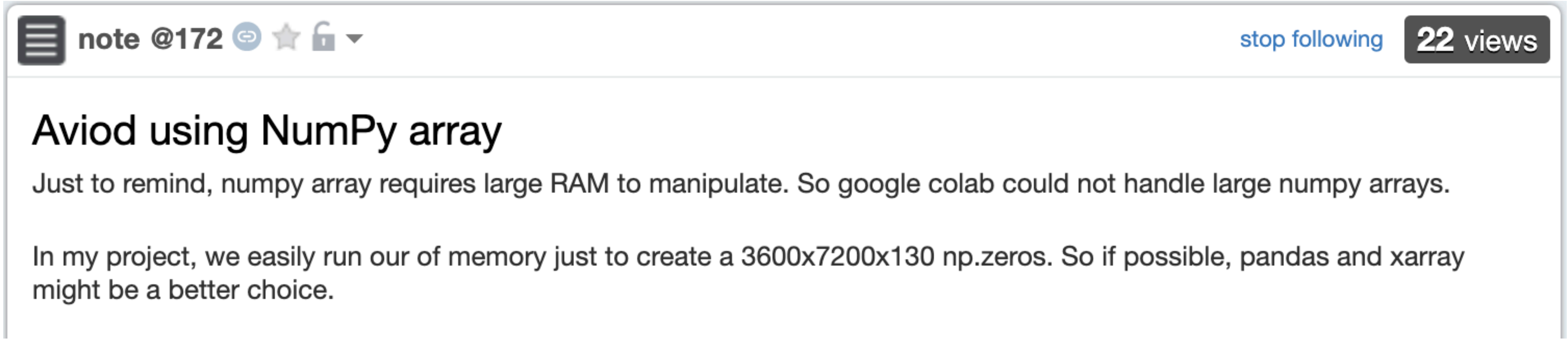
Last regular class!

Katy and Ethan share their code

OCEAN 215 | Autumn 2020

Ethan Campbell and **Katy Christensen**

Working with large netCDF files using `xarray`



note @172

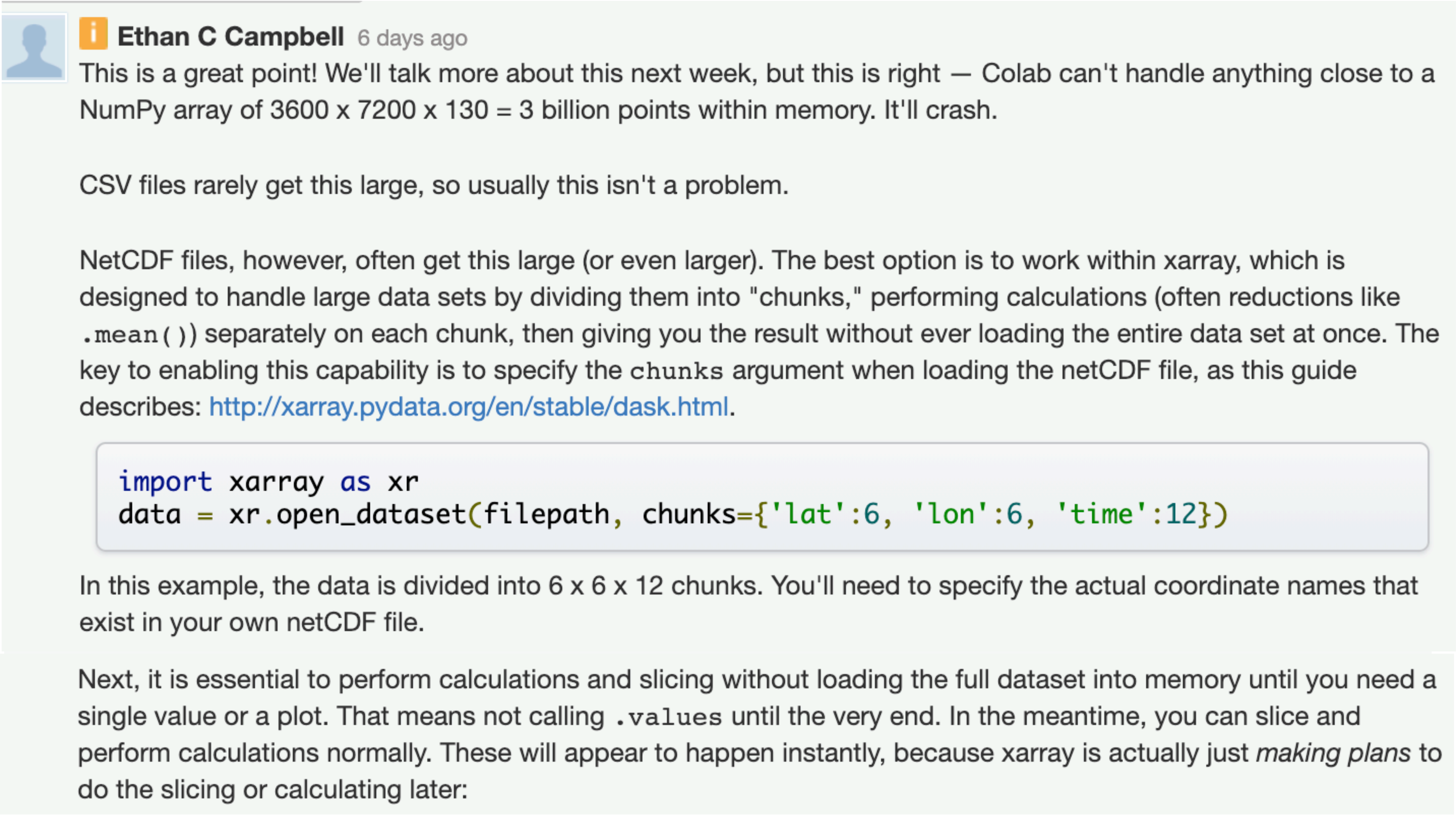
stop following

22 views

Aviod using NumPy array

Just to remind, numpy array requires large RAM to manipulate. So google colab could not handle large numpy arrays.

In my project, we easily run our of memory just to create a 3600x7200x130 np.zeros. So if possible, pandas and xarray might be a better choice.



Ethan C Campbell 6 days ago

This is a great point! We'll talk more about this next week, but this is right — Colab can't handle anything close to a NumPy array of 3600 x 7200 x 130 = 3 billion points within memory. It'll crash.

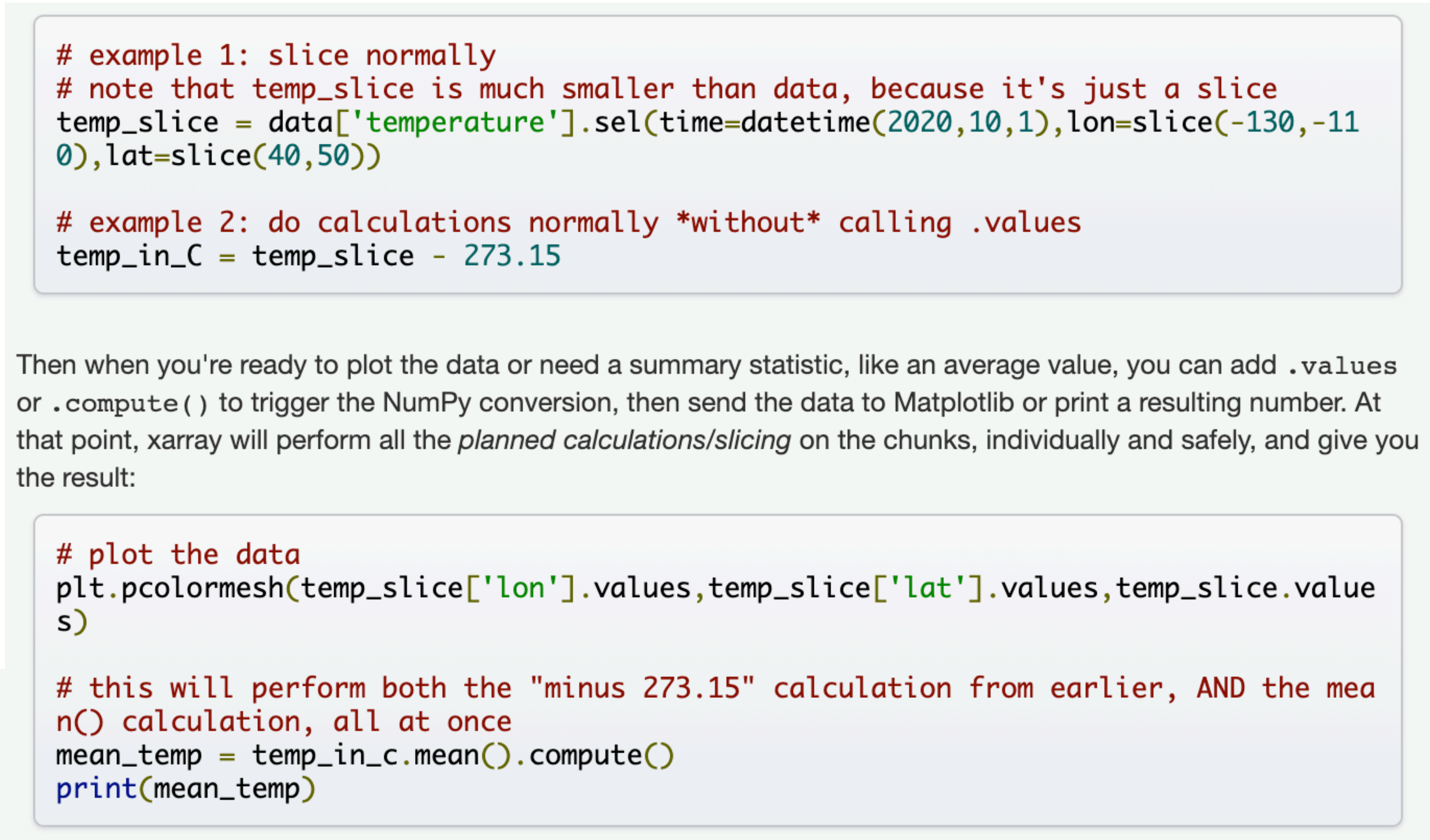
CSV files rarely get this large, so usually this isn't a problem.

NetCDF files, however, often get this large (or even larger). The best option is to work within xarray, which is designed to handle large data sets by dividing them into "chunks," performing calculations (often reductions like `.mean()`) separately on each chunk, then giving you the result without ever loading the entire data set at once. The key to enabling this capability is to specify the `chunks` argument when loading the netCDF file, as this guide describes: http://xarray.pydata.org/en/stable/dask.html.

```
import xarray as xr
data = xr.open_dataset(filepath, chunks={'lat':6, 'lon':6, 'time':12})
```

In this example, the data is divided into 6 x 6 x 12 chunks. You'll need to specify the actual coordinate names that exist in your own netCDF file.

Next, it is essential to perform calculations and slicing without loading the full dataset into memory until you need a single value or a plot. That means not calling `.values` until the very end. In the meantime, you can slice and perform calculations normally. These will appear to happen instantly, because xarray is actually just *making plans* to do the slicing or calculating later:



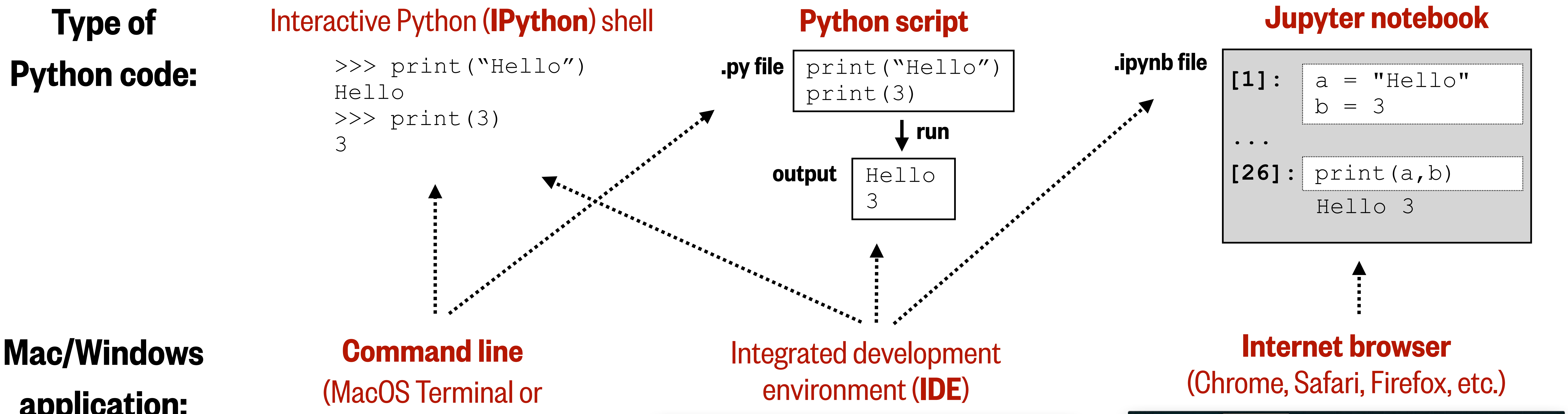
```
# example 1: slice normally
# note that temp_slice is much smaller than data, because it's just a slice
temp_slice = data['temperature'].sel(time=datetime(2020,10,1),lon=slice(-130,-11
0),lat=slice(40,50))

# example 2: do calculations normally *without* calling .values
temp_in_C = temp_slice - 273.15
```

Then when you're ready to plot the data or need a summary statistic, like an average value, you can add `.values` or `.compute()` to trigger the NumPy conversion, then send the data to Matplotlib or print a resulting number. At that point, xarray will perform all the *planned calculations/slicing* on the chunks, individually and safely, and give you the result:

```
# plot the data
plt.pcolormesh(temp_slice['lon'].values,temp_slice['lat'].values,temp_slice.value
s)

# this will perform both the "minus 273.15" calculation from earlier, AND the mea
n() calculation, all at once
mean_temp = temp_in_c.mean().compute()
print(mean_temp)
```

Link: https://piazza.com/class/kdhtk4p4izujg?cid=172

Different ways to code in Python



Type of Python code:

Interactive Python (**IPython**) shell

```
>>> print("Hello")
Hello
>>> print(3)
3
```

Python script

.py file

```
print("Hello")
print(3)
```

run

output

```
Hello
3
```

Jupyter notebook

.ipynb file

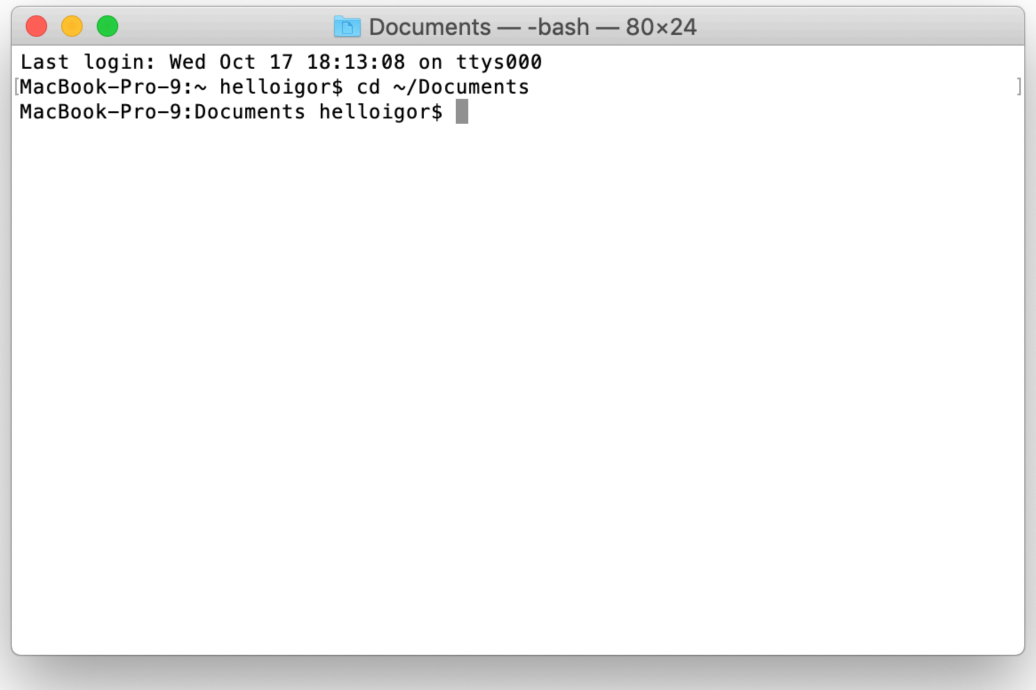
```
[1]:  a = "Hello"
      b = 3
...
[26]: print(a,b)
      Hello 3
```

Mac/Windows application:

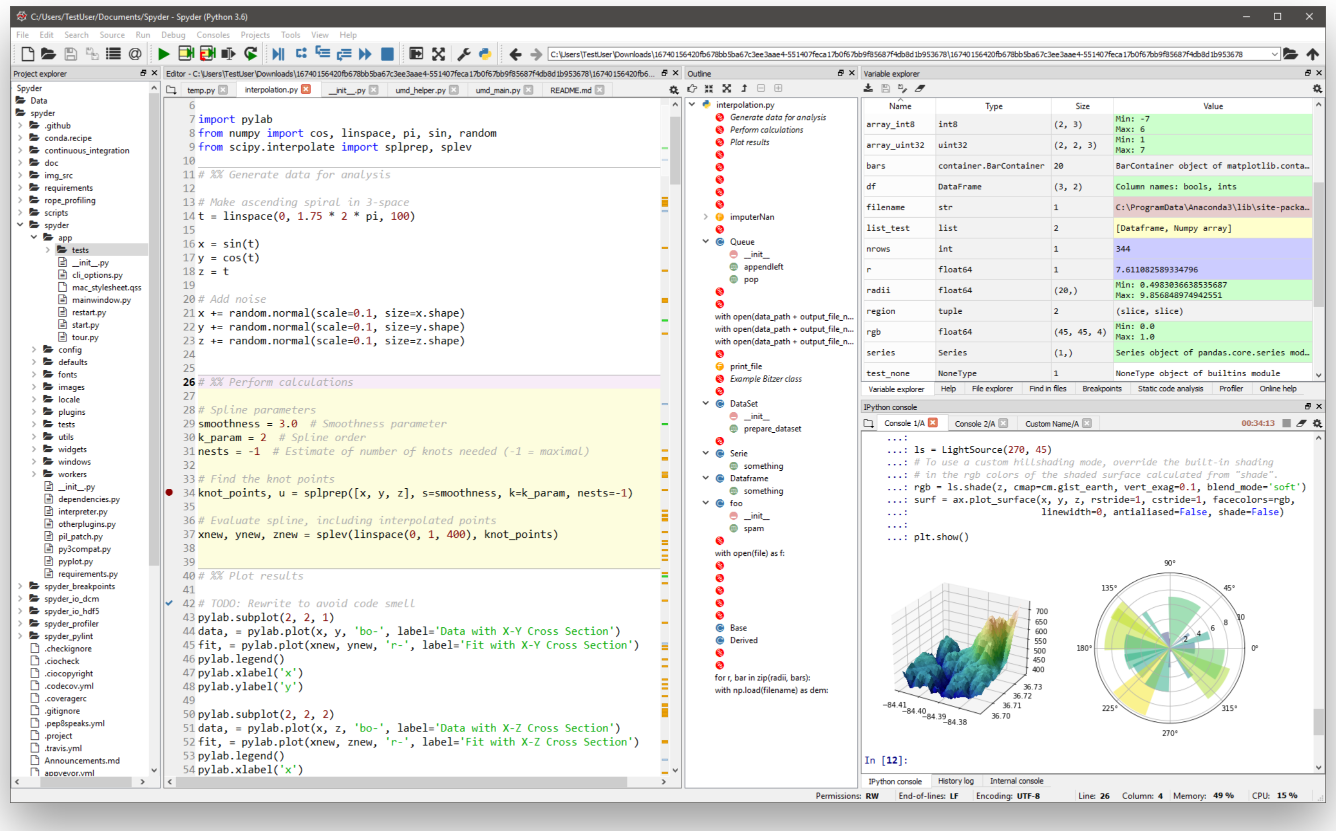
Command line
(MacOS Terminal or Windows Command Prompt)

Integrated development environment (**IDE**)

Internet browser
(Chrome, Safari, Firefox, etc.)

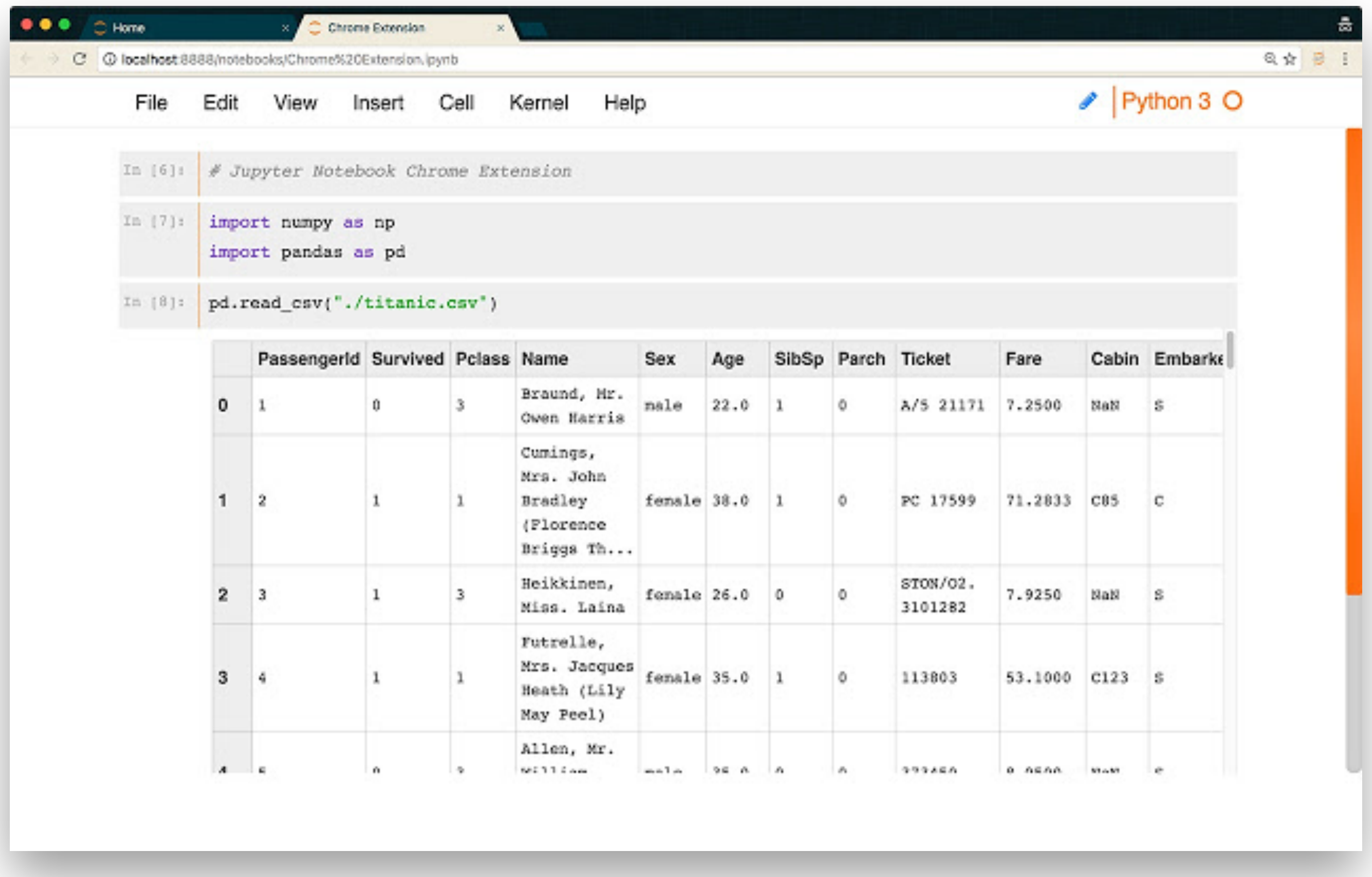


```
Last login: Wed Oct 17 18:13:08 on ttys000
MacBook-Pro-9:~ helloigor$ cd ~/Documents
MacBook-Pro-9:Documents helloigor$ 
```



```
import pylab
from numpy import cos, linspace, pi, sin, random
from scipy.interpolate import splprep, splev

# %% Generate data for analysis

# Make ascending spiral in 3-space
t = linspace(0, 1.75 * 2 * pi, 100)

x = sin(t)
y = cos(t)
z = t

# Add noise
x += random.normal(scale=0.1, size=x.shape)
y += random.normal(scale=0.1, size=y.shape)
z += random.normal(scale=0.1, size=z.shape)


# %% Perform calculations

# Spline parameters
smoothness = 3.0  # Smoothness parameter
k_param = 2  # Spline order
nests = -1  # Estimate of number of knots needed (-1 = maximal)

# Find the knot points
knot_points, u = splprep([x, y, z], s=smoothness, k=k_param, nests=-1)

# Evaluate spline, including interpolated points
xnew, ynew, znew = splev(linspace(0, 1, 400), knot_points)


# %% Plot results

# TODO: Rewrite to avoid code smell
pylab.subplot(2, 2, 1)
data, = pylab.plot(x, y, 'bo-', label='Data with X-Y Cross Section')
fit, = pylab.plot(xnew, ynew, 'r-', label='Fit with X-Y Cross Section')
pylab.legend()
pylab.xlabel('x')
pylab.ylabel('y')

pylab.subplot(2, 2, 2)
data, = pylab.plot(x, z, 'bo-', label='Data with X-Z Cross Section')
fit, = pylab.plot(xnew, znew, 'r-', label='Fit with X-Z Cross Section')
pylab.legend()
pylab.xlabel('x')
```

```
   ...:
   ...: ls = LightSource(270, 45)
   ...: # To use a custom hillshading mode, override the built-in shading
   ...: # in the rgb colors of the shaded surface calculated from "shade".
   ...: rgb = ls.shade(z, cmap=cm.gist_earth, vert_exag=0.1, blend_mode='soft')
   ...: surf = ax.plot_surface(x, y, z, rstride=1, cstride=1, facecolors=rgb,
   ...:                        linewidth=0, antialiased=False, shade=False)
   ...:
   ...: plt.show()
```



```
In [6]: # Jupyter Notebook Chrome Extension

In [7]: import numpy as np
        import pandas as pd

In [8]: pd.read_csv("./titanic.csv")
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarke
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4				Allen, Mr.								

Jupyter vs. Google Colab notebooks

	Where is the code run?	How to access them?	Advantages (+) and disadvantages (-)
Jupyter notebooks	Your computer (“the local machine”)	1. Install Jupyter 2. Open command line app (Terminal on Macs, Command Prompt on PCs) 3. Type “jupyter notebook,” which will start a local server 4. Open internet browser 5. Navigate to server address	● **(-) Some setup required** ● (+) No internet connection required ● (+) Code runs fast if your computer is fast ● (-) Code runs slow if your computer is slow ● (+) Bonus features, customizability, ability to install any package, etc. ● (+) Free
Google Colab notebooks We’ll be using these!	Google’s servers (“the cloud”)	1. Open internet browser 2. Navigate to: colab.research.google.com	● **(+) No setup required** ● (-) Requires internet connection ● (+/-) Code runs decently fast but not blazingly fast ● (-) Less customizability, more difficult package management ● (+/-) Free, as long as Google says it’s free ● (+) Google Drive integration; easy to share

Treating yourself to a local Python distribution

ANACONDA®

https://www.anaconda.com/products/individual

- Installs latest version of Python
- Installs **conda** and **pip** (package managers)
- Includes hundreds of common packages (NumPy, SciPy, Matplotlib, Pandas, etc.)
- Includes an IDE application (**Spyder**) and notebook environments (**Jupyter**, **JupyterLab**)

Then, you can write and edit Python code using:

A notebook environment like jupyterlab TNG (Comes with **Anaconda** installation)

An IDE application like PC PyCharm (https://www.jetbrains.com/pycharm/)

A command-line editor like ATOM (https://atom.io)