part-1

December 5, 2019

1 Getting Started with Python for Earth Sciences: Jupyter Notebooks, NumPy, pandas, and Matplotlib

1.1 Eviatar Bach

PhD student, Department of Atmospheric and Oceanic Science, University of Maryland, College Park

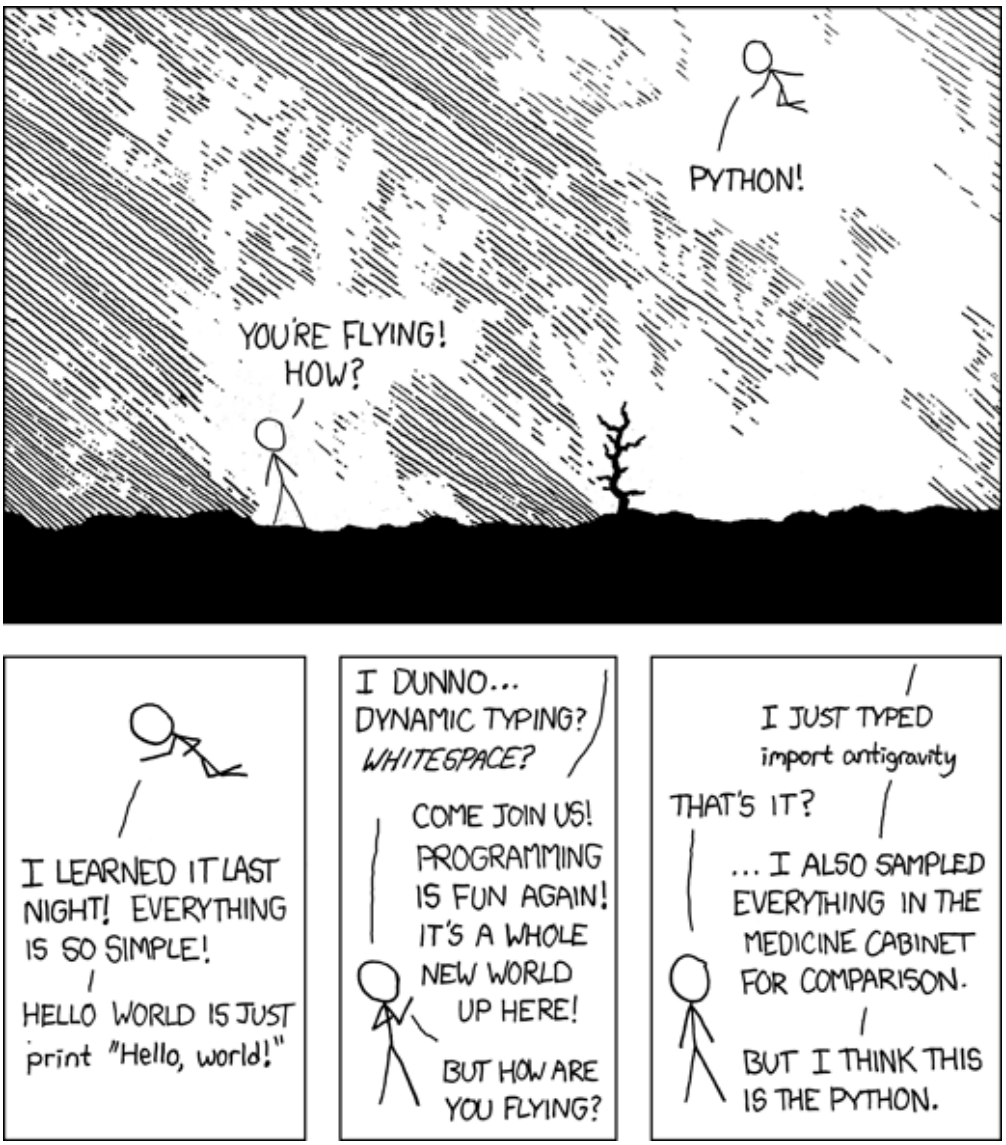
My website | My email | My Twitter | My GitHub

Based on previous courses prepared by **Rebekah Esmaili** (mostly Rebekah), Kriti Bhargava, and Eviatar Bach.

2 About me

PhD student with interests in climate dynamics, predictability, and data assimilation. Not satellites! But almost all my work has involved Python in some way. Feel free to email me with questions!

3 Introduction



3.1 Why Python?

Pros:

- General-purpose, cross-platform
- Free and open source
- Reasonably easy to learn
- Expressive and succinct code, forces good style
- Being interpreted and dynamically typed makes it great for data analysis
- Robust ecosystem of scientific libraries, including powerful statistical and visualization packages
- Large community of scientific users and large existing codebases

- Major investment into Python ecosystem by Earth science research agencies, including NASA, NCAR, UK Met Office, and Lamont-Doherty Earth Observatory. See Pangeo.
- Reads Earth science data formats like HDF, NetCDF, GRIB

Cons:

- Performance penalties for interpreted languages, although many libraries are wrappers for compiled languages. Avoid large loops in favor of matrix/vector operations when possible.
- Multithreading is limited due to the Global Interpreter Lock, but other parallelism is available
- See Julia for a modern scientific language which is trying to overcome these challenges

---

3.2 Objective: working with Earth science datasets

- You won't learn how to code in Python
- You will learn to:
  - Read/write ASCII and NetCDF data
  - Basic plotting and visualization
  - Perform basic operations on data

Python is an interpreted language, so you will need as a minimum to have Python on your computer.

3.3 What is Anaconda?

- Conda is a package manager
- Anaconda comes with conda, as well as Python, a lot of useful scientific/mathematical packages, and development environments.
- Easiest place to start if you're new

3.4 Development environments

- Spyder: most Matlab-like
- Jupyter notebooks: web based, runs code inline. Can also be run remotely over SSH; see here.
- Text editor + run with command line for scripting (IPython interpreter highly recommended)

3.5 Launching Jupyter Notebook

3.5.1 Linux/Mac

- Open terminal, **cd to the directory where you have your notebooks and data**, and type:

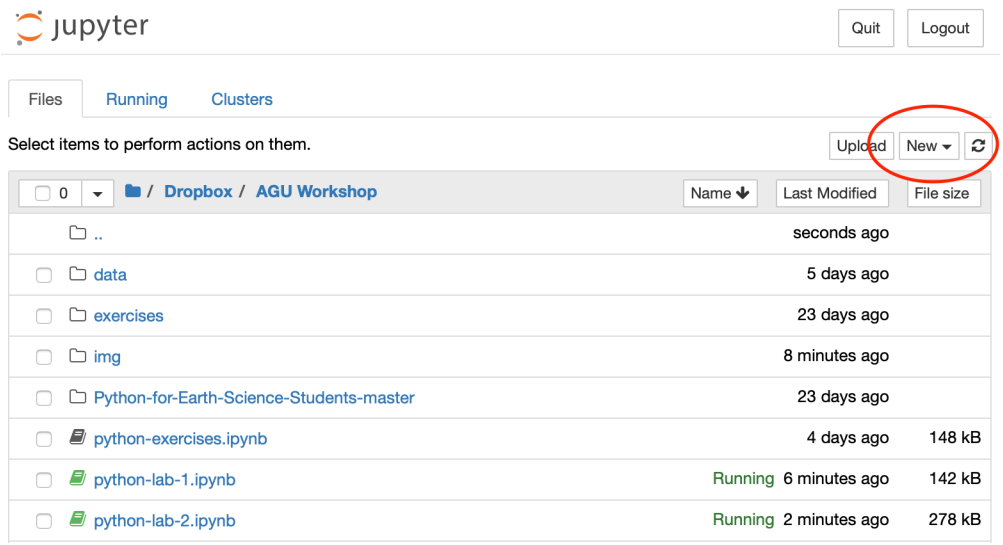
```
jupyter notebook
```

3.5.2 Windows

- Start Anaconda3 Jupyter Notebook

3.6 Jupyter Home Screen

- This will launch your default web browser with a local webserver that displays the contents of the directory that you're working in.
- Note: in all the examples, the path assumed that Jupyter is launched from the notebook directory. You will need to change the path to point to your data if this is different.
- Click on New on the top right.



Exercise 1: Set up your environment and create a notebook

- For your operating system, launch Jupyter Notebooks
- Create a new notebook
- Change the name from "untitled" to something better
- Save in the **same directory as the data folder** that we provided (or move the data directory to the same place at the file because we'll need it later!).

3.7 Basic Python commands

[1]:
```
# This is how we comment and below is how we print
print("Hello, world!")
```

```
Hello, world!
```

[2]:
```
# for loop
for i in range(5):
    print(i)
```

```
0
1
2
3
4
```

[3]:
```python
# iterating over list elements
print("List of hurricanes in 2019:")

# hurricanes is a list (in computer science terminology, a linked list)
hurricanes = ["Barry", "Dorian", "Humberto", "Jerry", "Lorenzo", "Pablo"]

for idx, name in enumerate(hurricanes): # notice the colon at the end
    print(idx + 1, name) # because index starts from zero in python
```

```
List of hurricanes in 2019:
1 Barry
2 Dorian
3 Humberto
4 Jerry
5 Lorenzo
6 Pablo
```

[4]:
```python
a = [1, 2, 3, 4, 5, 6]
b = a*2
```

[5]:
```python
# Is this what you expected to happen?
b
```

[5]:
```
[1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5, 6]
```

- Python's default list structure is for any type, not designed specifically for numeric operations
- Need to use an additional package to do vector/matrix operations

3.8 Importing packages

Packages give us additional functionality, saving us the trouble of writing procedures ourselves. There are ~6000 packages in the conda-forge repository alone!

We'll now import **NumPy**. NumPy provides high-performance multidimensional arrays and linear algebra operations (similar to Matlab). It is a fundamental package for scientific computing with Python.

[6]:
```python
# Importing NumPy
import numpy as np # np becomes the alias for numpy
```

[7]:
```python
# NumPy arrays
a = np.array(a) # a is now a NumPy array
b = a*2
```

```
print(b)
```

```
[ 2  4  6  8 10 12]
```

[8]:
```
# Reshaping arrays
a_reshaped = a.reshape(3, 2)
print(a_reshaped)
```

```
[[1 2]
 [3 4]
 [5 6]]
```

[9]:
```
# Indexing
print(a_reshaped[0, :])  # First row
print(a_reshaped[:, 0])  # First column
print(a_reshaped[2, 1])  # Third row, second column
print(a_reshaped[1:3, :])  # Second and third rows
```

```
[1 2]
[1 3 5]
6
[[3 4]
 [5 6]]
```

[10]:
```
# Sum vertically downwards across rows (axis 0)
a_reshaped.sum(axis=0)
```

[10]:
```
array([ 9, 12])
```

[11]:
```
print(a_reshaped.sum(axis=1))  # sum across columns
print(a_reshaped.sum())  # sum entire array
```

```
[ 3  7 11]
21
```

[12]:
```
# Get the minimum horizontally across columns (axis 1)
a_reshaped.max(axis=1)
```

[12]:
```
array([2, 4, 6])
```

[13]:
```
# Boolean operations
a_reshaped > 1
```

[13]:
```
array([[False,  True],
       [ True,  True],
       [ True,  True]])
```

[14]:
```
# Has many capabilities; for example, the code below creates a random linear␣
 ↪system and solves it

# Standard Gaussian distributed 10-by-10 matrix (A) and 10-element vector (b)
A = np.random.randn(10, 10)
b = np.random.randn(10)

# Solve A*x == b
x = np.linalg.solve(A, b)

print(x)
```

```
[ 1.32471607 -2.34230632 -0.12422118 -1.11951729 -1.01797555 -1.2221038
 -0.86566133  0.20357261 -0.61063287 -0.05756561]
```

[15]:
```
# Verify that x is an (approximate) solution. Note @ operator for matrix␣
 ↪multiplication; np.dot works too.
A@x - b
```

[15]:
```
array([-2.77555756e-16,  0.00000000e+00,  2.22044605e-16, -7.77156117e-16,
        4.44089210e-16, -7.21644966e-16, -6.38378239e-16,  2.22044605e-16,
       -2.22044605e-16,  3.88578059e-16])
```

For more examples, work through the NumPy quickstart

3.8.1 Pandas

- A library to work with tabular data
- Comparable to data frames in R

Pros:

- You can name columns and refence by labels instead of numeric indices like in NumPy arrays
- This also makes performing group operations easier and more readable

Cons:

- Designed for tabular data, not for dimensions > 2. We will cover xarray for N-dimensional data later.

3.8.2 Looking at real in-situ data: AERONET

- Aerosols are particles suspended in the atmosphere, including dust, sea salt, volcanic ash, smoke, and pollution.
- Aerosol Optical Depth (AOD) is a unitless measure of the amount of aerosols in the atmosphere.
- AERONET (AErosol RObotic NETwork) stations provide in-situ AOD observations.

[16]:
```
import pandas as pd
```

[17]:
```
filename = 'data/aeronet_locations_v3.txt'
station_list = pd.read_csv(filename, skiprows=1)
```

Note: If you are getting errors!

Check the path in the home tab in your browser, it must be relative to the where Jupyter was initially launched.

[18]:
```
# Show column names - kind of ugly
station_list.columns
```

[18]:
```
Index(['Site_Name', 'Longitude(decimal_degrees)', 'Latitude(decimal_degrees)',
       'Elevation(meters)'],
      dtype='object')
```

[19]:
```
# Shorter column names
station_list.columns = ['site', 'lon', 'lat', 'elev']

# List the first few rows
station_list[0:5]
```

[19]:

	site	lon	lat	elev
0	Cuiaba	-56.070214	-15.555244	234.0
1	Alta_Floresta	-56.104453	-9.871339	277.0
2	Jamari	-63.068552	-9.199070	129.0
3	Tucson	-110.953003	32.233002	779.0
4	GSFC	-76.839833	38.992500	87.0

Exercise 2: Import an ASCII file

- From the data folder, import "20180801_20180831_PNNL_lev15.csv" using the pandas read_csv command. Assign it to a variable.
  - HINT: You might need to check out the file path with respect to your notebook location.
- What are the column names?

3.8.3 Basic plotting with Matplotlib

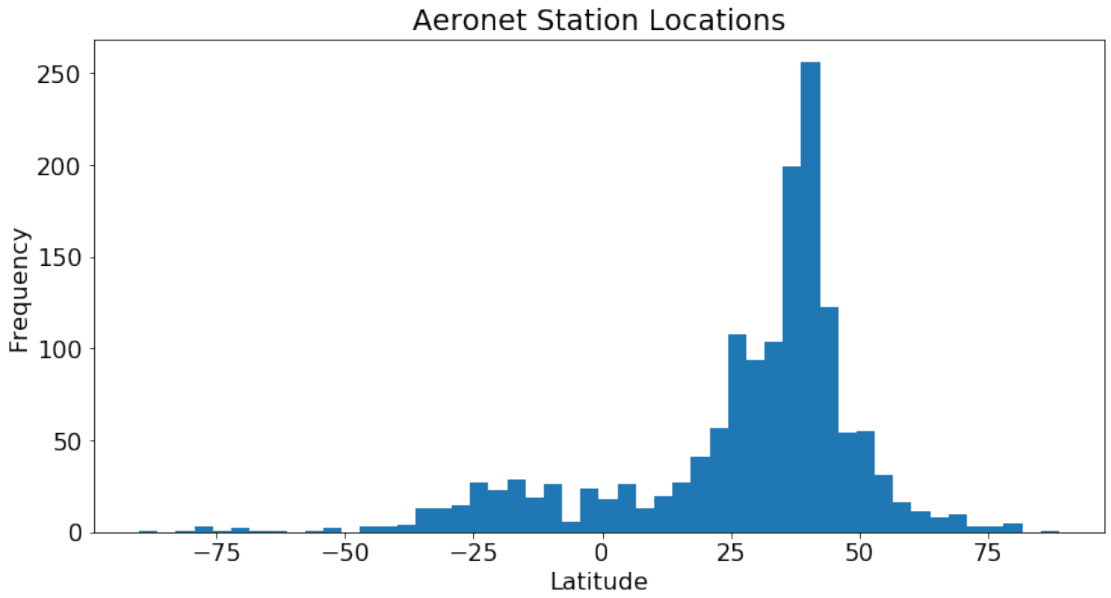
Common (simple) tasks in Earth science…

- Histograms
- Time series
- Taking averages, computing the bias

[20]:
```
# Matplotlib
import matplotlib.pyplot as plt
```

[21]:
```
# Options to increase figure size
plt.rcParams['figure.figsize'] = [12, 6]
plt.rcParams.update({'font.size': 16})
```

[22]:
```
# Create a histogram with 50 bins
plt.hist(station_list["lat"], bins=50)
plt.title("Aeronet Station Locations")
plt.ylabel("Frequency")
plt.xlabel("Latitude")

plt.show()
```



Exercise 3
Create a histogram

- Import the aeronet station list and locations from the file aeronet_locations_v3.txt using the pandas read_csv command.
- Plot a histogram of the longitude distribution of stations using the plt.hist(???) command

Time series data Need to tell Python the date strings in the file are dates/times.

[23]:
```
ground_station_PNNL = pd.read_csv('data/20180801_20180831_PNNL_lev15.csv')
```

[24]:
```
# Examine the first few lines...
ground_station_PNNL[0:2]
```

[24]:

	Date(dd:mm:yyyy)	Time(hh:mm:ss)	Day_of_Year	Day_of_Year(Fraction)	\
0	01:08:2018	13:38:54	213	213.568681	
1	01:08:2018	13:43:01	213	213.571539	

	AOD_1640nm	AOD_1020nm	AOD_870nm	AOD_865nm	AOD_779nm	AOD_675nm	\
0	0.021377	0.045989	0.059832	-999.0	-999.0	0.091885	
1	0.019886	0.041031	0.053130	-999.0	-999.0	0.080608	

```
                   ...                   Exact_Wavelengths_of_AOD(um)_380nm  \
0                  ...                                              0.3796
1                  ...                                              0.3796

   Exact_Wavelengths_of_AOD(um)_340nm  Exact_Wavelengths_of_PW(um)_935nm  \
0                              0.3406                              0.936
1                              0.3406                              0.936

   Exact_Wavelengths_of_AOD(um)_681nm  Exact_Wavelengths_of_AOD(um)_709nm  \
0                              -999.0                              -999.0
1                              -999.0                              -999.0

   Exact_Wavelengths_of_AOD(um)_Empty  Exact_Wavelengths_of_AOD(um)_Empty.1  \
0                              -999.0                                -999.0
1                              -999.0                                -999.0

   Exact_Wavelengths_of_AOD(um)_Empty.2  Exact_Wavelengths_of_AOD(um)_Empty.3  \
0                                -999.0                                -999.0
1                                -999.0                                -999.0

   Exact_Wavelengths_of_AOD(um)_Empty.4
0                                -999.0
1                                -999.0

[2 rows x 113 columns]
```

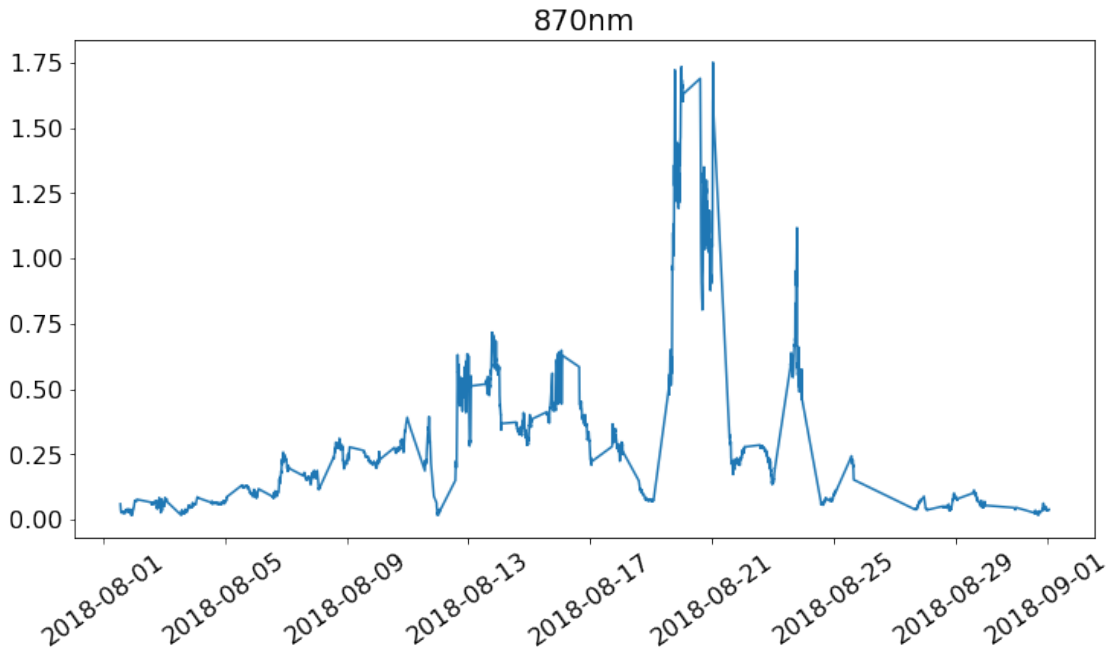
[25]:
```
# Convert the string dates and times to date format (note the line␣
 ↪continuation)
ground_station_PNNL['datetime'] = (ground_station_PNNL['Date(dd:mm:yyyy)'] + ' '
                                   + ground_station_PNNL['Time(hh:mm:ss)'])

# Explicitly declare the format (for the full list, see http://strftime.org)
fmt = '%d:%m:%Y %H:%M:%S'
ground_station_PNNL['datetime'] = pd.
 ↪to_datetime(ground_station_PNNL['datetime'], format=fmt)
```

Plotting a single variable

- Add the data using plt.plot(dataframe['X Column Name'], dataframe['Y Column Name'])

[26]:
```
plt.plot(ground_station_PNNL['datetime'], ground_station_PNNL['AOD_870nm'])
plt.xticks(rotation=35)
plt.title('870nm')
```

[26]:
```
Text(0.5, 1.0, '870nm')
```



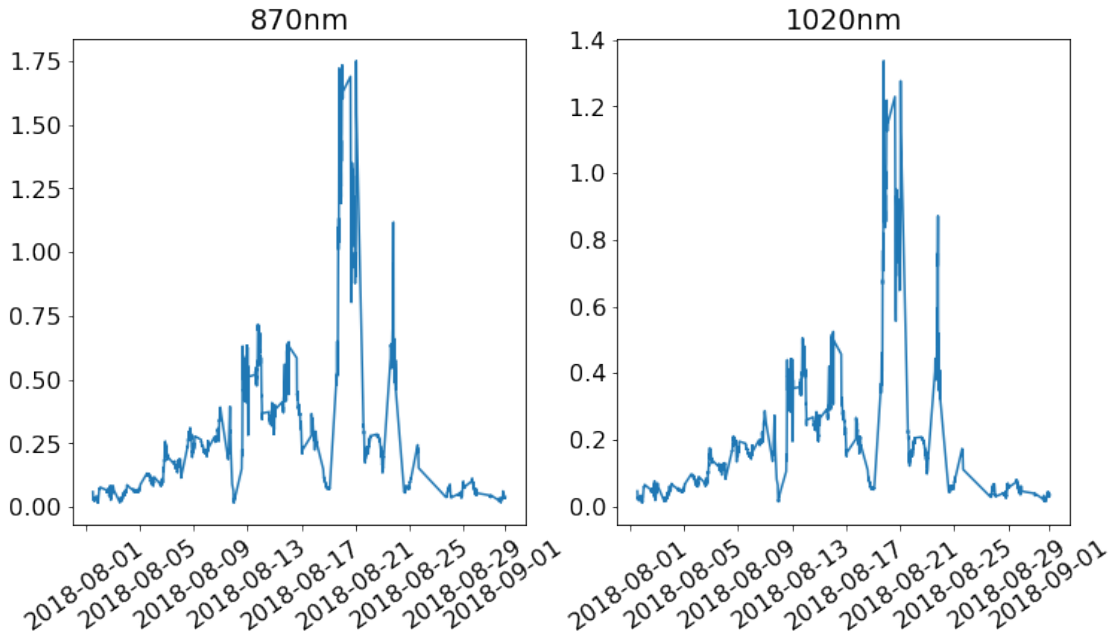
Plotting two side-by-side plots

- Before writing your plt.plot code, add: plt.subplot(**row number**, **column number**, **position of the plot**)
- Call plt.plot() and fill in the x and y variables for the first line/data series on the plot
- Call plt.subplot again, increment the position of plot number
- Add in any aesthetics, such as rotating the axes, labels, and adding a legend
- Show the plot using plt.show()

Example: * plt.subplot(2, 1, 1) places the plot in the first position of a two row, one column stack of plots. * plt.subplot(1, 2, 2) places the plot in the second position of a two column, one row of plots next to each other.

[27]:
```
# Call subplot - in column one, write the AOD_870nm plot
plt.subplot(1, 2, 1)

plt.plot(ground_station_PNNL['datetime'], ground_station_PNNL['AOD_870nm'])
plt.xticks(rotation=35)
plt.title('870nm')

# Call subplot - in column two, write the AOD_1020nm plot
plt.subplot(1, 2, 2)
plt.plot(ground_station_PNNL['datetime'], ground_station_PNNL['AOD_1020nm'])
plt.xticks(rotation=35)
```

```
plt.title('1020nm')

plt.show()
```

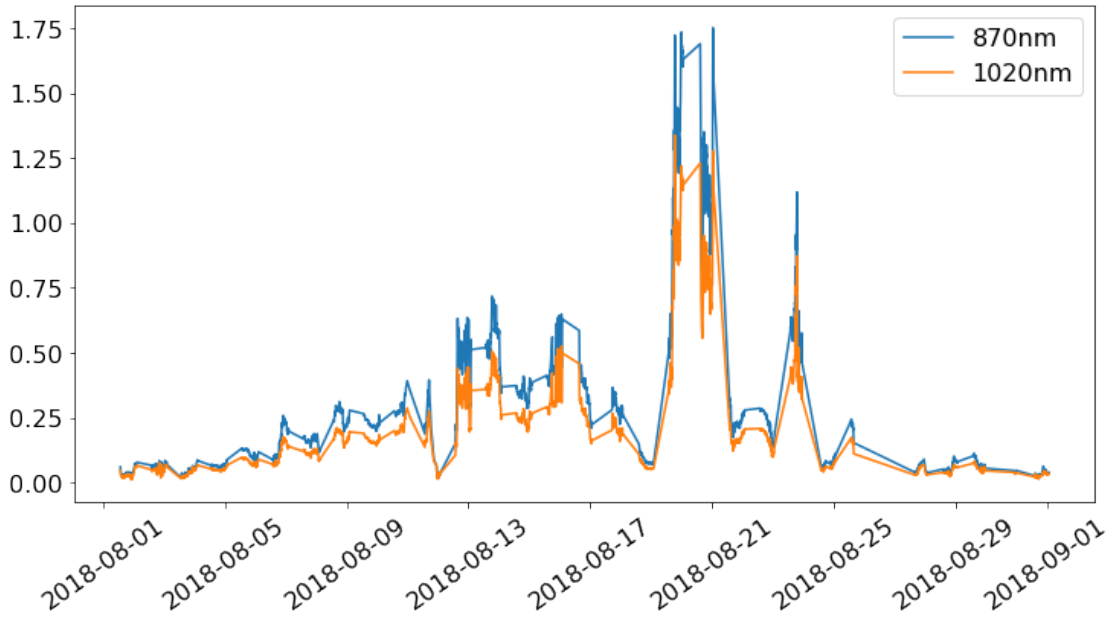


Adding data to an existing plot

- Call plt.plot() and fill in the x and y variables for EACH line/data series on the plot
- Add in any aesthetics, such as rotating the axes, labels, and adding a legend
- Show the plot using plt.show()

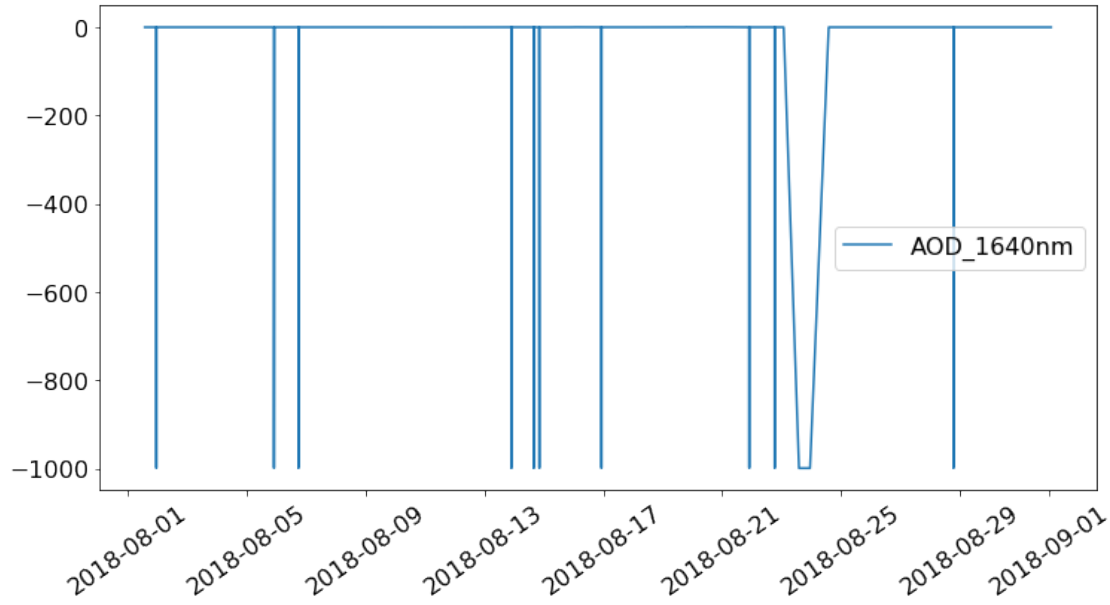
Note: If you want to change from a line to a point, you can add a '.' argument to plot.plot() or use plt.scatter() instead.

[28]:
```
plt.plot(ground_station_PNNL['datetime'], ground_station_PNNL['AOD_870nm'])
plt.plot(ground_station_PNNL['datetime'], ground_station_PNNL['AOD_1020nm'])
plt.xticks(rotation=35)
plt.legend(['870nm', '1020nm'])
plt.show()
```

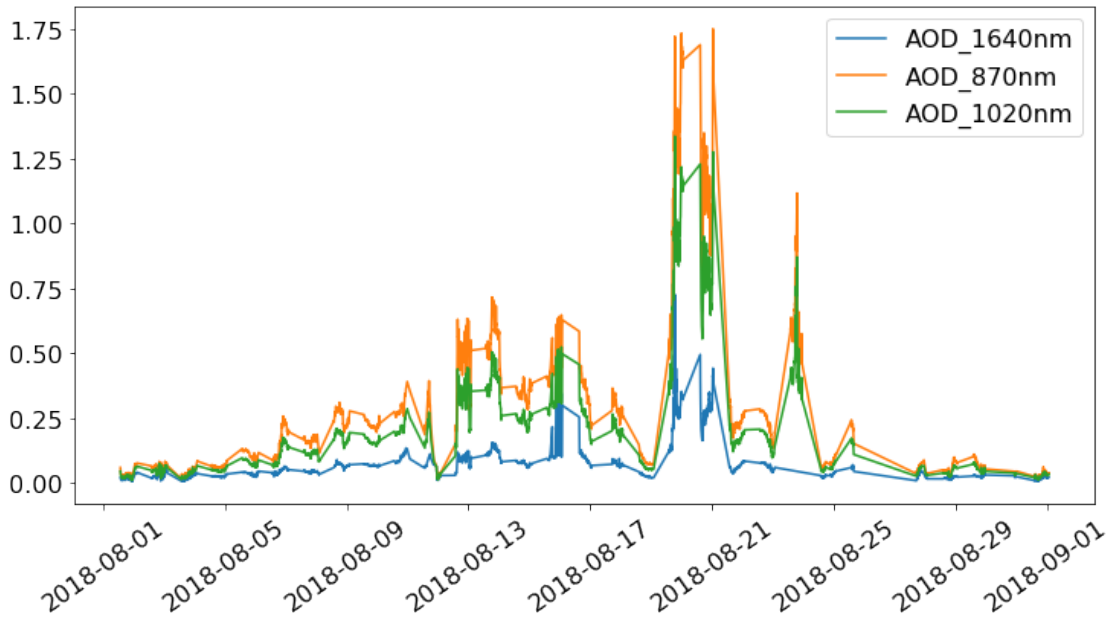


Filtering data Sometimes there are values we don't want in our plots, e.g. missing data values of -999.0

[29]:
```
# The following plot is not very useful because the missing values are being␣
↪added
plt.plot(ground_station_PNNL['datetime'], ground_station_PNNL['AOD_1640nm'])
plt.xticks(rotation=35)
plt.legend()
plt.show()
```



[30]:
```
# Filter out missing values by only using ground_station_PNNL['AOD_1640nm'] >= 0
valid_mask = ground_station_PNNL['AOD_1640nm'] >= 0

# Then plot the filtered values...
plt.plot(ground_station_PNNL[valid_mask]['datetime'],␣
 ↪ground_station_PNNL[valid_mask]['AOD_1640nm'])

# Option: Add in the other AOD lines
plt.plot(ground_station_PNNL['datetime'], ground_station_PNNL['AOD_870nm'])
plt.plot(ground_station_PNNL['datetime'], ground_station_PNNL['AOD_1020nm'])

plt.legend()
plt.xticks(rotation=35)

plt.show()
```

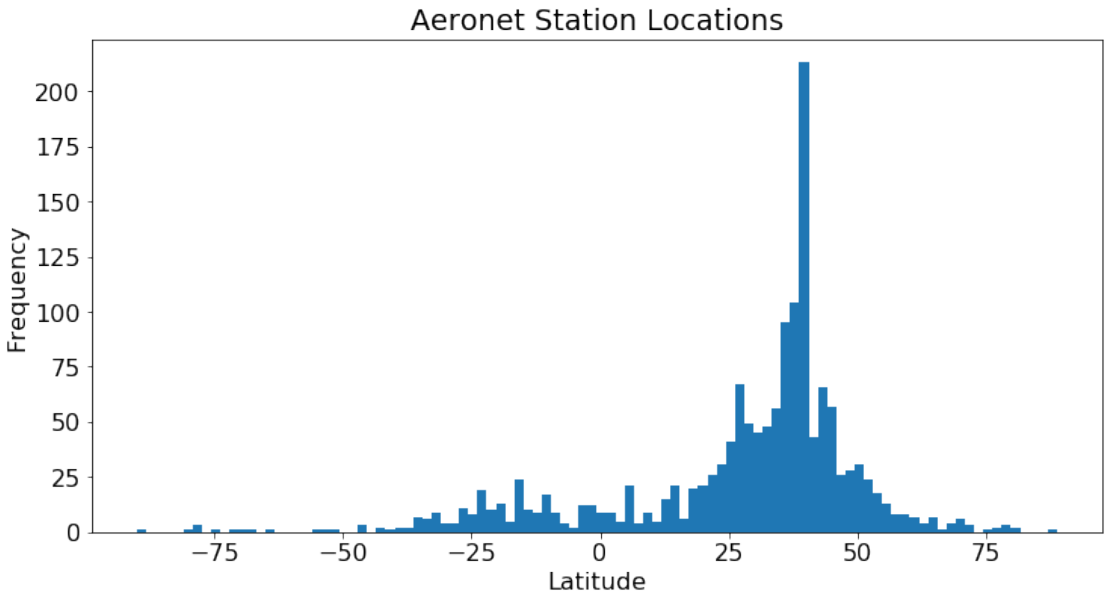


3.8.4 Saving images

- plt.savefig('pick_a_filename')
- Save as a vector format, such as PDF, SVG, or EPS, when possible. These files have "infinite resolution". See the AGU guide, for example.

[31]:
```
plt.hist(station_list["lat"], bins=100)
plt.title("Aeronet Station Locations")
```

```
plt.ylabel("Frequency")
plt.xlabel("Latitude")

plt.savefig("histogram.pdf")
```



3.9 Resources

- Some (free!) ways to learn:
  - CS Dojo Youtube series for absolute beginners
  - Automate boring stuff
  - Codeacademy
  - Local Meetups
  - Start a Python Club!