Module 10 – Reading and Writing Files

Instructions

1) If not already open, from the Anaconda Navigator Home screen, launch Sypder

2) In Spyder, go to "File", then "Open", then navigate to the location where you saved: "10_ReadingWritingFiles.py", and select it to open the script file.

3) Introduction

Now that we've covered the basics of the data types, let's turn our attention to something that I'm certain is/was high on your wish list coming into this workshop: learning how to import data and work with it in your research or research team's workflow.

This is also the module where you will start working with the datasets that I provided to you. These are: 1) the NV_Cities.txt file, 2) the NV_Cities.csv file, and 3) the Washoe_County.csv file. I strongly recommend you have all three saved in one directory in your working environment.

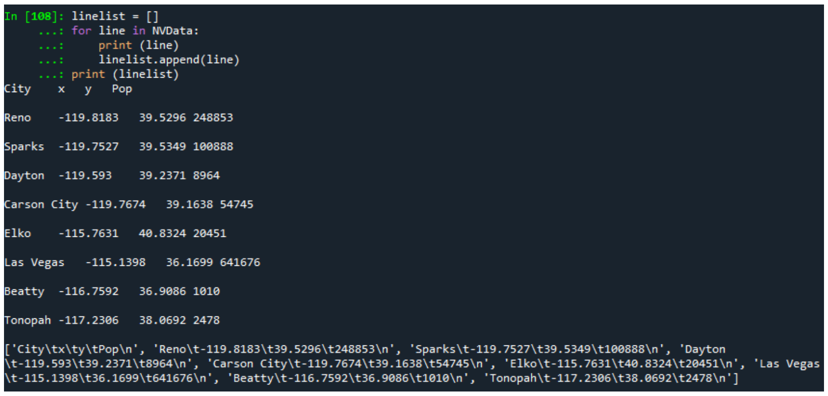
In this module, we'll use a couple basic techniques for reading/writing files that come along with the base Python installation. Then, we'll turn our attention to the **pandas** module, which comes along with your download of the Anaconda Distribution, and use it to read data and make some histograms. **<u>**Important: this is the module that contains the deliverable for the workshop**</u>**

At the end, you'll get a chance to read some data in from a .csv, use these data to make a histogram, and modify that histogram. You'll upload your modified histogram for the credit for the course.

4) Reading and Writing Files in Python - Basic

Before starting this module, make a note of the location where your three files are saved (NV_Cities.txt, NV_Cities.csv, and Washoe_County.csv). **Run lines 18-20 together**, and modify the files' location in Line 20 to reflect your environment. If you enter my code as-is, you will see an error message. After running Line 20, **run Line 21**. Line 21 opens this file, returning the file as an object, now associated with the variable "NVData." If this line ran without error, you are now in the right working directory, and the text file has been opened.

To read the contents of the file, **run lines 25-29 together**. Here, I initiate an empty list (line 25), appending each line read in the file to the list (linelist). You should see an output like this:

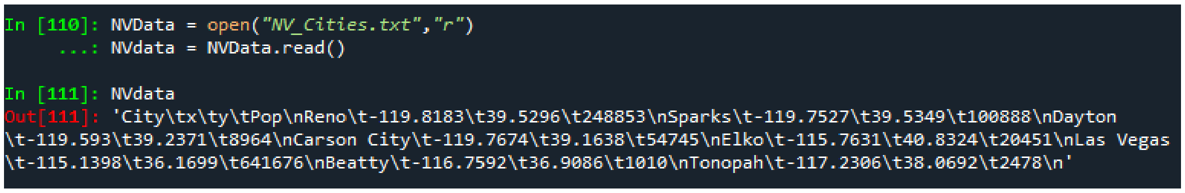


```
In [108]: linelist = []
     ...: for line in NVData:
     ...:     print (line)
     ...:     linelist.append(line)
     ...: print (linelist)
City    x   y   Pop

Reno    -119.8183   39.5296 248853

Sparks  -119.7527   39.5349 100888

Dayton  -119.593    39.2371 8964

Carson City -119.7674   39.1638 54745

Elko    -115.7631   40.8324 20451

Las Vegas   -115.1398   36.1699 641676

Beatty  -116.7592   36.9086 1010

Tonopah -117.2306   38.0692 2478

['City\tx\ty\tPop\n', 'Reno\t-119.8183\t39.5296\t248853\n', 'Sparks\t-119.7527\t39.5349\t100888\n', 'Dayton
\t-119.593\t39.2371\t8964\n', 'Carson City\t-119.7674\t39.1638\t54745\n', 'Elko\t-115.7631\t40.8324\t20451\n', 'Las Vegas
\t-115.1398\t36.1699\t641676\n', 'Beatty\t-116.7592\t36.9086\t1010\n', 'Tonopah\t-117.2306\t38.0692\t2478\n']
```

The print statement within the definite loop is simply printing out each line that is read. We can see that we've got 8 city names and a file header, for a total of 9 lines. Stored with these city names are **x**, **y**, and **Pop** values (that refer to longitude, latitude, and population values for each city). Then, we see the list **linelist** after each line has been appended to it.

See any familiar characters within each string object? Note the **\t** and **\n** characters, which we saw way back in the string data module. We can use those to get this list full of unwieldly string elements into more manageable formats. Still, this block of code is most helpful for just reading what is in the file. Before we move on, let's start implementing good practice. Line 30 closes the file – get in the habit of including that line of code early and often. **Run Line 30**.

Next, **run lines 33 and 34 together**. Now we have used the .read() command. Enter NVdata into the console, and now look at the output



```
In [110]: NVData = open("NV_Cities.txt","r")
     ...: NVdata = NVData.read()

In [111]: NVdata
Out[111]: 'City\tx\ty\tPop\nReno\t-119.8183\t39.5296\t248853\nSparks\t-119.7527\t39.5349\t100888\nDayton
\t-119.593\t39.2371\t8964\nCarson City\t-119.7674\t39.1638\t54745\nElko\t-115.7631\t40.8324\t20451\nLas Vegas
\t-115.1398\t36.1699\t641676\nBeatty\t-116.7592\t36.9086\t1010\nTonopah\t-117.2306\t38.0692\t2478\n'
```

This time, there was no need to read each line in and append it to an empty list. The read method will store everything in this .txt file in one long string, with each new line indicated by the **\n** character.

Run line 36, which lets us use that to our advantage. When you **run Line 38** after the split, and access index position 1, you'll see the output, which include the line holding Reno's data. Then **run line 39** to close to the file. **Run lines 42-48 together** to show an example of how to conditionally read in lines, which might help if you're only interested in certain subsets that meet certain conditions. In this example, we use the **continue** function to pass over the first line (since it starts with City), while the rest of the lines are evaluated under the 'else' condition that prints each line.

Finally, **run Lines 52-61 together** and let's review. First, we read through each line, using the .strip command on Line 55 (this time with an '\n' in the argument), which removes the new line character from the string. Then we split by the tab command (line 56) to place each row into its own element in a list. Since the city name is always the first element in each row, we identify it using its index position [0] on Line 57, then append this to our empty list on line 58. Lastly, on Line 59, we make use of an allowable operation with list data. The **pop()** function removes an item from the list at – in this case – the specified index position (the first one), which was the word 'City'. Line 60 prints the names of all the cities in the list, and then we close the file on Line 61.

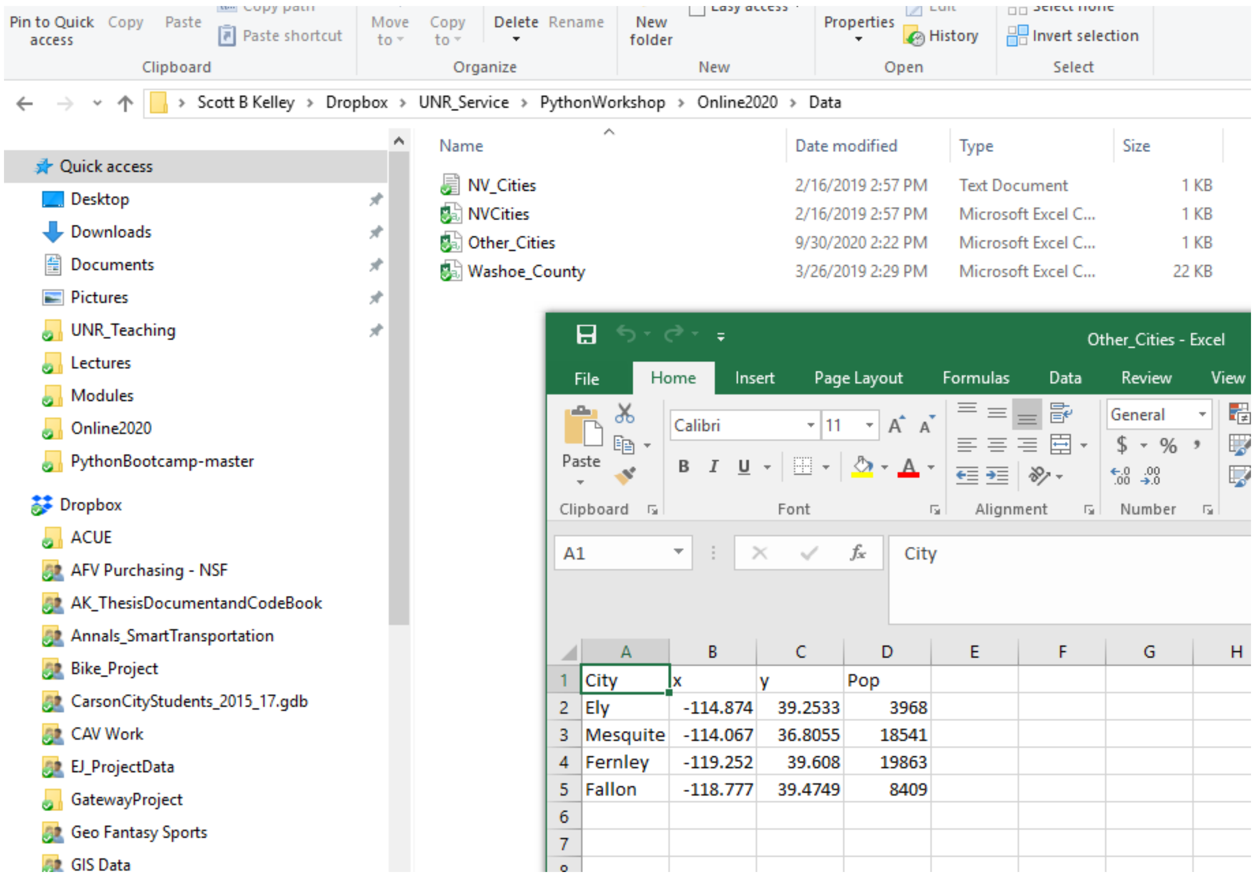
5) Reading .csv files - Basic

Python does have the ability to read csv files. Importing the csv module is one way to do it, and this functionality has been included in the basic Python installation for awhile. I will be blunt about this – **pandas** (which we'll cover next) makes things far easier than this, but in the off-chance something were to happen to it, it's nice to have a backup. But because pandas is a better way to proceed, I won't go into too much detail here.

To use the csv module, import it (**Run line 65**), and, similar to the definite loops above with the text file, I make use of lists and list operations to get the data formatted how I want. **Run Lines 66-74 together** and let's review. This time, I read the NVCities.csv file (Line 67), though (not the .txt file), and create a reader (Line 68). I set a counter (Line 69), then set a definite loop that iterates through each line of the reader, and after skipping over the header (Line 71), I read the first element of each line and append it to the empty list on Line 66. Then I step to the next line (Line 73). This now stores each city name in a list.

Take a look though, at Lines 77-83. Here, I'm writing data to a new .csv file, which will be saved in the working directory set way back on Line 20. We give the file a name and indicate that we'll be writing data to this file (Line 77). Next, I add a writer object, and indicate that a comma will be used to separate values (Line 78). Then, I simply write 5 new rows into this formerly blank table. **Run Lines 77-83 together**. When done, navigate to your working directory.

You should see a file called "Other_Cities.csv", and if you were to open it in a program like Excel, this is what you would see. You just wrote a new .csv file to your working directory!



Name	Date modified	Type	Size
NV_Cities	2/16/2019 2:57 PM	Text Document	1 KB
NVCities	2/16/2019 2:57 PM	Microsoft Excel C...	1 KB
Other_Cities	9/30/2020 2:22 PM	Microsoft Excel C...	1 KB
Washoe_County	3/26/2019 2:29 PM	Microsoft Excel C...	22 KB

	A	B	C	D
1	City	x	y	Pop
2	Ely	-114.874	39.2533	3968
3	Mesquite	-114.067	36.8055	18541
4	Fernley	-119.252	39.608	19863
5	Fallon	-118.777	39.4749	8409

6) Working with pandas

The commented section of Lines 89-94 spell this out pretty well, so I will spare you the repetition. The highlights are: pandas is an acronym for (Python Data Analysis Library), more here: https://pandas.pydata.org/

This truly makes things much easier to read and write files than some of the methods I showed above. The advantage of those earlier methods is that they use functions and modules that come along with a Python base installation. However, pandas comes along with the Anaconda Distribution download.

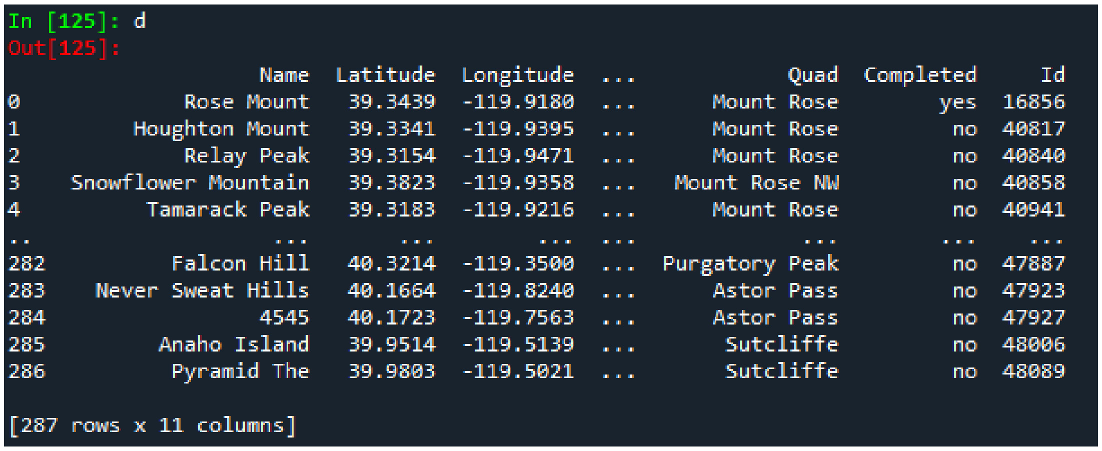
Line 88 imports pandas (and does so with the shorthand pd in the namespace, which will be useful below).

Line 87 imports part of the matplotlib module, which we will cover in more detail in the next module. For now, **run Lines 87 and 88 together**.

Once done, view line 96. A variable **d** stores the output of accessing the read_csv command associated with the pandas module, pointed at the “Washoe_County.csv” file saved in your working directory. **Run Line 96.**

Next, **run and view Lines 97 and 98**. These set variables for two of the columns in the .csv file (“Elevation” and “Prominence”). Notice all I had to do to access these was include the **d** variable ahead of brackets that contained the columns’ names in the data. **Much** easier than some of the methods above!

Before proceeding, enter **d** into the console input, which should bring about something like this:



```
In [125]: d
Out[125]:
                  Name  Latitude  Longitude  ...          Quad  Completed     Id
0            Rose Mount   39.3439  -119.9180  ...    Mount Rose        yes  16856
1        Houghton Mount   39.3341  -119.9395  ...    Mount Rose         no  40817
2            Relay Peak   39.3154  -119.9471  ...    Mount Rose         no  40840
3     Snowflower Mountain 39.3823  -119.9358  ...  Mount Rose NW        no  40858
4         Tamarack Peak   39.3183  -119.9216  ...    Mount Rose         no  40941
..                  ...       ...        ...  ...           ...        ...    ...
282         Falcon Hill   40.3214  -119.3500  ...  Purgatory Peak       no  47887
283     Never Sweat Hills 40.1664  -119.8240  ...    Astor Pass         no  47923
284                4545   40.1723  -119.7563  ...    Astor Pass         no  47927
285        Anaho Island   39.9514  -119.5139  ...     Sutcliffe         no  48006
286         Pyramid The   39.9803  -119.5021  ...     Sutcliffe         no  48089

[287 rows x 11 columns]
```

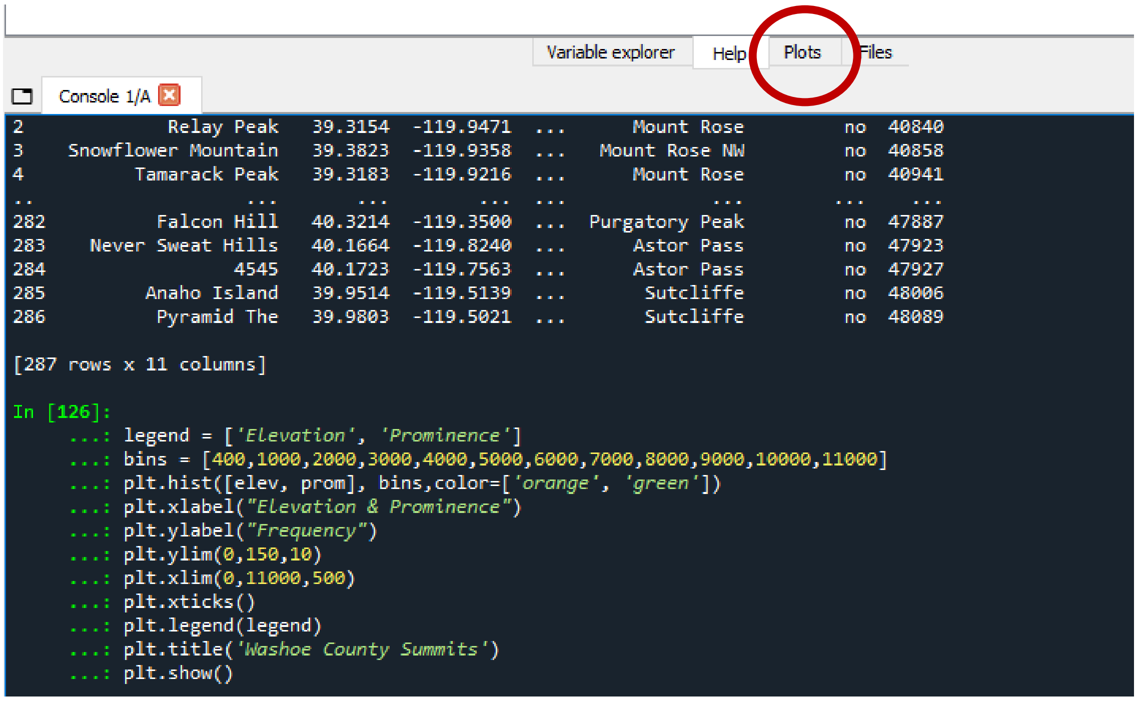
Line 96 alone would have produced that output! But I did want to show you that can set variable names using the column names themselves (Elevation and Prominence are not visible in this screenshot, but they are in the dataset).

7) Your Deliverable for the Course

After running lines 87-88, and then 96-98 (if you haven’t yet), review (but don’t run yet), Lines 101-111. While there are quite a few lines of code here, you’ll see they are each quite simple. If you’ve worked in other platforms making plots, charts, and graphs, this probably feels like familiar turf. Each of these lines does something to contribute to a histogram that is the output of this block of code. The histogram is created using the .csv file (Washoe_County.csv), and read in by pandas.

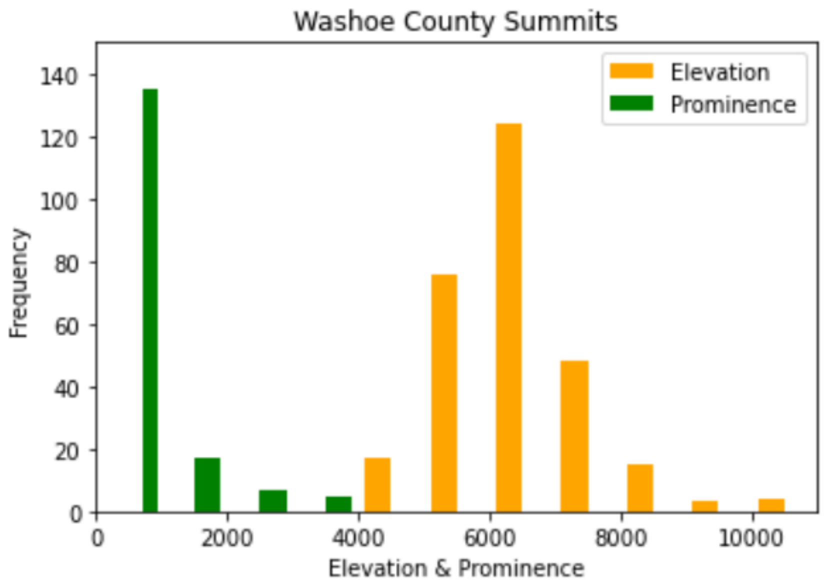
Each of the lines of code should be fairly self-explanatory in what they do (as in, Line 104 produces the plot’s x-label, 105 the y-label, etc.). After reviewing, run lines **101-111** together.

This part may feel a bit like R Studio. You might get a notice about plots not showing up in the console output, and that's ok. We don't need them to show up there. To see output, look above the console, where you should see four tabs, one of which is "Plots." Click on it.



```
2          Relay Peak  39.3154  -119.9471  ...    Mount Rose        no  40840
3   Snowflower Mountain  39.3823  -119.9358  ...  Mount Rose NW     no  40858
4        Tamarack Peak  39.3183  -119.9216  ...    Mount Rose        no  40941
..                 ...      ...        ...  ...           ...       ...    ...
282        Falcon Hill  40.3214  -119.3500  ...  Purgatory Peak    no  47887
283   Never Sweat Hills  40.1664  -119.8240  ...    Astor Pass       no  47923
284               4545  40.1723  -119.7563  ...    Astor Pass       no  47927
285       Anaho Island  39.9514  -119.5139  ...     Sutcliffe       no  48006
286        Pyramid The  39.9803  -119.5021  ...     Sutcliffe       no  48089

[287 rows x 11 columns]

In [126]:
     ...: legend = ['Elevation', 'Prominence']
     ...: bins = [400,1000,2000,3000,4000,5000,6000,7000,8000,9000,10000,11000]
     ...: plt.hist([elev, prom], bins,color=['orange', 'green'])
     ...: plt.xlabel("Elevation & Prominence")
     ...: plt.ylabel("Frequency")
     ...: plt.ylim(0,150,10)
     ...: plt.xlim(0,11000,500)
     ...: plt.xticks()
     ...: plt.legend(legend)
     ...: plt.title('Washoe County Summits')
     ...: plt.show()
```

The histogram should now be visible, and if code that I provided to you is run without modifications, you should see something closely resembling this:



We'll get more into the nuts and bolts of matplotlib in the next module, but for now, what I want you to for your deliverable is simply this.

Change three (3) things about the plot by changing the code (anywhere between lines 97-111), and upload two things to the Assignment submission for this workshop:

1) An image (or even a screenshot) of the modified histogram

2) A short explanation of the three things you changed, and how you did it. You can either show me the code or just talk me through the changes in narrative form, so long as the submitted image of the modified histogram reflects them.

Changes can include **any** of the elements set by the code that occur between Lines 97-111. Anything is fair game, so long as some of those lines are modified by you to produce a histogram that looks different than mine. Please use the same source/input data (Washoe_County.csv), though. Let's be honest: there is much to be improved about this histogram, so please do!

This completes the structured part of Module 10. Feel free to experiment some more with setting variables and values, working with print statements, or anything else you are curious about.