**GEOG/EOS 230**

Tutorial 1: Introduction to data file pre-assessment and Google Collaboratory.

In this tutorial we have the following objectives:

* Have a first look at Google Collaboratory
* Download and install Notepad++ for examining ascii files
* Examine ASCII files using Notepad++ to start thinking about ingest
* Read in some files using Python
* Learn about the concept of objects, functions, attributes, methods, and arguments
* Learn about python programming structures (loops, conditionals, functions and blocks)
* Introduce the concepts of loops, conditionals, and code blocks.
* Provide hands-on experience with basic data ingest from a CSV file.
* Practice exploratory analysis approaches

**Google Collaboratory**

Google Collaboratory, Google Colab, or just colab, provides an online environment in which computer code development can occur. By “development” is meant the process of writing, running, trouble-shooting and refining computer code until it is working the way you want. This type of environment is also called an “IDE”, or “Integrated Development Environment”. This is different from the old days, so to speak, where the process of writing code and running code were two different tasks (note that it is still the case for some languages – compiling is a separate task). That is, one can write code for any language in any text editor, such as Notepad++, but you are not able to run it or troubleshoot it, beyond the editor possibly providing you with help to correct syntax. Further, Colab is an example of a “notebook” style of IDE. These are quite popular because along with the code they can create finished documents by combining nicely formatted text and inline plots and tables. For this reason you can often find tutorials available in a notebook format; they are also useful for preparing reports. Colab uses the notebook structure favored by Jupyter notebooks – Jupyter is probably the leading notebook format out there (it is actually an older format – IPython – that is stored in an ASCII format termed *JSON*, an important and widespread file format which we’ll look at later).

***In this tutorial*** we will look at how to access colab (not difficult - <https://colab.research.google.com/> needs a google account to access) and how to set up your environment, which mostly means telling colab where to find your input files and where to send your output files and save your code.

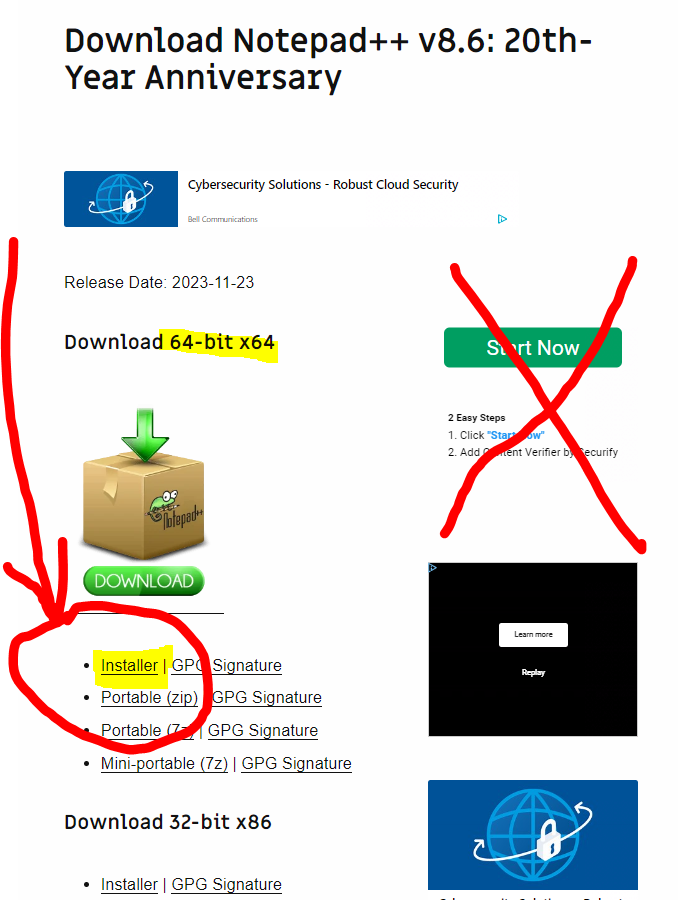
**Notepad++**

You can install Notepad++ on your own machine. It is lightweight and easy to install. Here is the site with a recent version (Nov 2023): <https://notepad-plus-plus.org/downloads/v8.6/>

Be careful: the page has some trick advertising downloads. Click on the installer for 64-bit machines. See below for what to click. When you look at the website and scroll down, note that there are installers for different types of computer “architectures” – 32-bit x86 and ARM64. Recall our quick look at a bit of x86 Assembly Language code from the other day. x64 uses the same basic instructions as x86, but ARM64 is an entirely different setup. So, you need to be aware of what type of machine you have when you are installing software. I looked at ARM64 and found a nice writeup of what we discussed the other day:

<https://mariokartwii.com/armv8/ch1.html>

Have a look at chapters 2, 3, and 4 of this tutorial as well. They are short reads but have some nice info.

Notepad++ installer – always best to click on the actual installer link. Many software download sites have fake ad things that look like installer buttons, so be extremely careful.

***In this tutorial*** we will use Notepad++ to open ASCII files that we want to read in to provide us with information about the file structure that will guide how we should approach the reading process. If you don’t have a machine that can install Notepad++ let me know and we’ll see if we can find something online. I looked and, while there are many, I couldn’t find any that have a hidden character toggle. If anyone knows of one, please speak up!

**File Ingest – what to look for**

As discussed the other day, when we bring an ASCII file into Notepad++ we are looking for a number of things:

* Is it actually an ASCII file…
* Basic structure of the file:
  + Information about variables – a header present?
    - If not, where is the metadata file?
  + Is there one observation per variable per row? Or are there multiple obs per row for a given variable?
* Separators
* Missing value encoding
* Flags
* Presence of spurious characters

Is it actually an ASCII file This will be readily apparent when you open the file! If it’s not ASCII then it is one of two main forms of binary (“binary-fied” data or one of several types of self-describing structured file).

Basic structure of the file – header? As we saw the other day, a header is just that – information about the variables and data in the file that resides in the first few lines of data, termed the “header”. A header can be any number of lines, and a header can be as little as one line that provides direct information about the variable (its name), two lines (name and units), or many lines that provide more information about where/how the data were gathered. Such information is useful of course, but is normally discarded during the actual reading process. If no header where is the metadata file? This is clearly important if you don’t know what the data are! Sometimes you do know and it doesn’t matter as much. Generally, though, it is poor practice to prepare datasets that don’t have any metadata information associated with them. Others who rely on you for data will not be pleased, and even your future self will thank you (this goes for commenting your code as well – it is obvious in the moment, but return to it in 6 month, or as your career goes on, 20 years later! I guarantee it will no longer be quite so obvious).

Is there one observation per variable per row, or multiple obs per row? We’ll have a look at this one, but it is what it says. You can imagine data from a weather station – each row is a single record from the station, taken on the hour, and contains one observation from each instrument. The next row is repeated with obs for the next hour. That can be contrasted with some formats that have annual values arranged by decade. These are harder to read in.

Separator We discussed this a bit the other day. Is there a separator between individual values, and what is it? If there is no separator, that is termed a “column delimited” file. It’s considered archaic, but you will definitely still see it around.

Missing value encoding Many datasets, especially time-series data, can have missing data. It is very important to provide an explicit means of identifying such values. For example, you can’t just use 0 because, for many variables, this is a valid value that says “was observed but was zero”, rather than “was not observed”. The encoding for this can take a wide range of forms and is another important reason to consult metadata for the file.

Flags A flag is a code that can accompany a data value. Typical codes used by Environment Canada include M for Missing, E for Estimated, and T for Trace. There can be other codes when an operator has edited the file and such a code accompanies a data value (and gets noted in metadata). Conditions when an operator modifies data include to fill in missing values with estimates or to remove erroneous values. While we can try to look for missing values during the Notepad++ stage, you might have large files that do not allow you to easily scan through the file, e.g. 30 years of hourly data has more than half a million obs – a missing value might occur at ob #100,000. That’s what the Initial Data Assessment in python is for!

Presence of spurious characters Sometimes you can end up with a file that for whatever reason carries traces of some other file structure or format codes. A good example is having odd html coding characters inserted into a file. These will mess up a data read, throwing strange errors.

Metadata. The information or description of data is called metadata. Often, it is in the header, but it can be provided as separate data file to explain other data.

***In this tutorial*** we will explore a range of files in Notepad++, paying attention to the above items.

**Read in some files using Python**

We will walk through this in tutorial focussing on the following commands:

**import pandas as pd**  - this activates a library called pandas and gives it a short-form handle “pd”

**read\_csv()** – one of several readers that pandas has. Recall csv stands for comma-separated values, which is a standard data file format. Note this is a *function*, denoted by the (). It has many possible *arguments* a user can select but only one is required, which is where to find the data file. The rest of the arguments are optional (and are assigned *default values*, something very important to keep in mind about functions).

***In this tutorial*** we will read in a couple of ascii files into python, noting how to handle different format and structures.

**Learn about the concept of objects, functions, attributes, methods, and arguments**

Python is an object-oriented scripting language. “Object-oriented” means it is built around the concept that everything is an object with its own methods and attributes, and a “scripting language” means that it runs inside an *interpreter*, rather than being *compiled*.

The idea of methods and attributes for an object may be thought of by using an analogy to a house. A house would be an object. A **method**, which is just a function, provides tools that perform operations on that particular type of object. So a method for a house object could be **house.close\_door()** Arguments for the close\_door() method might include door\_Id and slam. They would be assigned default values like door\_Id=”Front” and slam=False (slam would be a “Boolean” variable, taking only true or false). Thus without any user modification to the arguments, **house.close\_door()** would refer to closing the front door in a calm manner. If you want to change that, you have to specifically refer to the argument: **house.close\_door(door\_Id=”R\_side”)** ; so this would close the side door calmly. You cannot run **close\_door()** on anything but an object of type house. Note that “house” can be any name and it is assigned the object type of house. For example you could have a house object called “Green\_Gables”, on which the above method would work. An **attribute** does not do anything but report on information associated with the object. As such, it has no (). Take careful note of this distinction between method and attribute. Attributes for a house object could include levels, roof\_type, and basement. Entering Green\_Gables.roof\_type simply returns a value for that attribute, and typically you assign it to a variable (otherwise it just appears in the output). Attributes can return any type of data: string, numeric, or Boolean. house.levels will return a numeric as an integer (1, 2, 3, etc levels); house.roof\_type will return a string (asphalt\_shingle, metal, etc) and house.basement will return a Boolean value, True or False.

A *function* is a very small program within a program, which means a collection of lines of code that are grouped together and given a name for ease of reference. You can create your own functions, and in fact, you will find as you go on that it can be hard to imagine a program that does not have a function you have written. As we mentioned, a library is a place where many functions are stored, that typically work on a general theme, for example, the MetPy library that houses a large collection of functions for producing specialized weather plots: <https://unidata.github.io/MetPy/latest/examples/index.html>

***In this tutorial*** we will learn about these features of the Python language as we commence starting to work with the language.

**Python structures**

There are about a thousand tutorials on these things, so I won’t repeat here. Go to these sites from datacamp and w3schools:

**For loop**: <https://www.datacamp.com/tutorial/for-loops-in-python> (**Stop** at “Iterating over a sequence with Lambda function”)

**Conditionals**: <https://www.datacamp.com/tutorial/elif-statements-python> (Note the function % - this is called the modulus division operator. It returns the remainder of a division, so 4%2 actually returns 0

**Functions**: <https://www.w3schools.com/python/python_functions.asp> (Don’t worry about recursion at the very end)

Ask me if you have questions.

**To submit (100 marks total):**

**Please Work on without using generative AI.**

**Week 1: File Ingestion (40 marks)**

1. Access the folder “Files\_to\_try\_in\_Notepad” and open each of the 15 files. Provide answer as a table (A Sample Table is in Appendix). For each, Provide the following:

Describe the file using the following identifiers (1 mark per file):

* 1. binary or ascii
  2. separated values or not separated values
  3. Metadata included yes or no

1. Would pandas read\_csv work to read in this file? (1 mark per file)
   1. Yes or no
   2. One sentence why or why not

**\*\*MAKE SURE THE FILES ARE IN ALPHABETICAL ORDER\*\***

Access the folder “Files\_to\_read\_Python”:

1. Write a program to read in the files. For each file show your program. (10 marks).

**Week** 2: **Basic Python Structures (10 marks)**

1. Write a Python program that uses a `for` loop to print the numbers from 1 to 10 (2 marks).
2. Implement a function called `is\_even` that takes an integer as an argument and returns `True` if it's an even number and `False` otherwise. Test the function with both even and odd numbers (5 marks).
3. Create a list of integers `[3, 7, 12, 5, 20]`. Use a `for` loop to iterate through the list and print each element multiplied by 2 (3 marks).

**Conditional Statements (14 marks)**

1. Write a program that asks the user to input a number. Use an `if-else` statement to check if the number is positive, negative, or zero, and print the result (print the number and a statement saying if pos, neg, or zero) (4 marks).
2. Read the data set “student\_grades.csv”. Implement a function called `grade\_classifier` that takes a student's percentage as input and returns their grade: 'A' for 90- 100, 'B' for 80-89, 'C' for 70-79, and 'F' for anything below 70. (10 marks).

**Data Ingest (36 marks)**

1. Download the provided CSV file `weather\_data.csv` containing weather data. Isolate the DATE, TAVG, and PRCP columns. Read the file and convert the DATE to a proper datetime variable. Print the first 5 rows of data (10 marks).
2. Write a function called `average\_temperature` that takes the temperature column from the CSV file and calculates the average temperature. Print the result (12 marks).
3. Implement a program that takes PRCP column counts the number of days with precipitation greater than quantile 75. Print the count (14 marks).

**Bonus (10 marks)**

1. Extend the program in C.3 to also calculate and print the percentage of days represented by the values exceeding q75. How can you assess whether your result is reasonable?

Submission:

Submit a Python script file containing your code for each exercise along with a summary of your approach and any challenges faced.

Note:

* Ensure your code is well-commented and follows good coding practices.
* Use meaningful variable and function names.
* Test your code thoroughly with different inputs.
* **Appendix**

Table 1: Data Ingestion Answer Table

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Filename | Binary or ascii | Separated values | Metadata included | Pandas\_ csv read | Why? |
| 1100030.ascii |  |  |  |  |  |
| 2018081900.nc |  |  |  |  |  |
| 08LE021\_Daily\_Flow\_ts.csv |  |  |  |  |  |
| counts\_2021\_9.csv |  |  |  |  |  |
| datafile.asc |  |  |  |  |  |
| event\_data\_1993.txt |  |  |  |  |  |
| event\_data\_structure.txtx |  |  |  |  |  |
| meiv2.data |  |  |  |  |  |
| Oliver\_vineyard\_station.kmz |  |  |  |  |  |
| Ottawa\_data.pkl |  |  |  |  |  |
| Ottawa\_weather.py |  |  |  |  |  |
| package.json |  |  |  |  |  |
| tasmax\_day\_BCCAQv2+AN USPLIN300\_HadGEM2-AO  \_historical+rcp26\_r1i1p1\_19 500101-21001231.nc.ascii |  |  |  |  |  |
| test\_data\_2.csv |  |  |  |  |  |
| test\_plot.html |  |  |  |  |  |