Lab #3 – Json plotting

Goals of the lab

In this lab, we'll

- Learn a few new Python features – opening files, reading Json-formatted data from files, and working with dictionaries.
- Reinforce our plotting skills.

Overview of the lab

The Tufts Coronavirus-dashboard website shows various views; the last seven days vs. cumulative, student vs. faculty/staff, etc. However, it does *not* allow you to see trends over time. In this lab, we'll fix that!

Professor Bell has put together a web scraper that goes to the dashboard, pulls in that day's data and saves it in a file. The scraper runs again every day. We will read all these files (one file for each day's data), assemble the information and plot statistics over time.

If you haven't already, you should look at the Tufts Coronavirus dashboard at https://coronavirus.tufts.edu/testing-metrics. Familiarize yourself with the data, especially the *Cumulative Results* section.

Metacode

Here's a high-level version of the top-level code flow:

```
(stu_cum, fac_cum) = read_files ()
plot (the cumulative data) with the title "cumulative"

# Turn the cumulative data into new-cases-per-day data
stu_new = []      # everything in here also gets done for faculty!
for each pair of days (0 & 1, 1 & 2, 2 & 3, etc):
    subtract the cumulatives to get the new cases for that day
    append that day's new cases to the stu_new list
plot (the new-cases-per-day data) with the title "per-day"
```

Specifically, we first read the entire set of files to get the cumulative number of infections, and plot it. We then massage the cumulative infection count to get new-cases-per-day counts and plot those.

While most of the lines above are reasonably close to Python, the *read_files*() function is quite vague. So here's metacode for the *read_files*() function.

```
initialize month and date to the first day that we have data for
initialize stu_cum and fac_cum to empty lists
while (they still represent a legal month/date):
      read the data file for that day
      grab the relevant data, append it to stu_cum and fac_cum lists
      increment the date (correctly rolling to February if needed)
return the two lists
```

That's the high-level *metacode*, which tells you "roughly" how to code everything. Now for some detail!

The main functions

The metacode mentions two functions, *read_files*() and *read_json_file*(), for you to implement and use. Here is more detail on them:

- *read_files* (). This function reads the .json files for all dates between Jan. 14th and Feb. 28th inclusive (which is 46 days of data). It does this by calling the function *read_json_file* (*filename*) 46 times, each time giving it the name of the file containing that day's dashboard snapshot. Each call to *read_json_file* (*filename*) gets the data from the file *filename* and returns the cumulative student and faculty infection count from that day's dashboard. *Read_files*() accumulates this data over the entire time period and returns two 46-element lists *stu_cum* and *fac_cum*. E.g., *stu_cum* holds 46 numbers, where each number is the cumulative number of students testing positive as of that day. *Fac_cum* is the same, but for faculty/staff.
- *read_json_file* (*filename*). It uses *json.load* (see below) to read a json file. It then grabs two numbers from the file – the cumulative number of students and of faculty who have tested positive so far – and returns them. This is the function that *read_files*() uses to do much of its work.

The files

We give you 46 files – stats_2021_1_14.json, stats_2021_1_15.json, etc.; all the way through stats_2021_2_28.json. You can get them in any of several ways:

- If you're logged into the Halligan cluster, they're available in the directory /es/2/public_html/data/tufts/
- The files are also available to download over the web; e.g., you can find stats_2021_1_14.json http://www.ece.tufts.edu/es/2/data/tufts/
- That same web location also contains the file *dashboard.zip*, which is a Windows compressed folder. You can download it to a Windows laptop and then use it, without needing to download 46 individual files one by one. However, once you download it to your laptop, you will have to extract it. I.e., right-click on the *dashboard.zip* icon, select **Extract all**, and pick the name of the extracted folder – that will be the one that you actually use.

Note that the data files for the last few days will appear as soon as they are available; the Feb 28th data will appear on Monday, March 1st.

Format of each data file

Each data file has text that looks like the following:


```
{"date": "January 10, 2021", "7day": {"total": 6976, "negative": 6941,
"positive": 35, "processingtime": 16.93, "boston-faculty-pos": 3,
"boston-students-pos": 20, "boston-faculty-neg": 676, "boston-students-
neg": 1667, "grafton-faculty-pos": 1, "grafton-students-pos": 2,
"grafton-faculty-neg": 444, "grafton-students-neg": 413, "medford-
faculty-pos": 3, "medford-students-pos": 6, "medford-faculty-neg": 711,
"medford-students-neg": 688, "other-faculty-pos": 0, "other-other-pos":
0, "other-faculty-neg": 61, "other-other-neg": 24}, "cumulative":
{"total": 271329, "negative": 271003, "positive": 324,
"processingtime": 16.87, "boston-faculty-pos": 25, "boston-students-
pos": 72, "boston-faculty-neg": 892, "boston-students-neg": 2094,
"grafton-faculty-pos": 18, "grafton-students-pos": 10, "grafton-
faculty-neg": 559, "grafton-students-neg": 471, "medford-faculty-pos":
43, "medford-students-pos": 148, "medford-faculty-neg": 1514, "medford-
students-neg": 6597, "online-student-pos": 0, "online-student-neg": 1,
"other-faculty-pos": 6, "other-other-pos": 2, "other-faculty-neg": 106,
"other-other-neg": 97}}
```

This is the *Json* format, and is a fairly standard way of describing data structure via text. In our case, it describes a Python dictionary with the syntax { key1:value1, key2:value2, …}. The extra trick is that some of the values are themselves full dictionaries! The full Json format is easy to find on the web if you're interested, but we'll only be using this small piece.

Reading one file

Now you know where to find the files and what is in each file. The function *read_json_file*() gets the name of a single file (which, again, holds the data for a snapshot of the dashboard on one single day), and must grab the data from the file and return two numbers – the cumulative infected count for students, and for faculty/staff. Here's a bit more detail on how to do that.

The easiest way to read a file is by using code similar to

```
import json
file = open ("02_10_2021.json")
obj = json.load (file)
```

(Note that, depending on how you access the data files, you probably will have to use the full pathname of where you've placed them (e.g., for me, *C:\users\joelg\desktop\dashboard_data\stats_2021_01_14.json*).

With this done, your *obj* will then contain a dictionary of the data for (in this case) Feb 10. The dictionary will have just three keys: *date, 7day*, and *cumulative*. The *date* key should have a string (e.g., "January 10, 2021") as its value. The *7day* and *cumulative* keys each have an entire dictionary as their value!

Drilling down to that second level, these two dictionaries have identical formats. For example, they each have the key *medford-students-pos*, with a value such as 710. In the *cumulative* dictionary, this is the cumulative number of students who have ever had a positive test. Similarly, *medford-faculty-pos* is the number of faculty/staff who have ever had a positive test.

Your function *read_json_file*() should grab the relevant data (i.e., the cumulative counts for Medford students and for Medford faculty/staff) from the dictionary, and return that data. Remember that the "numbers" in the dictionary are actually strings, and you must still conver them to integers. The Python code **int** ("13") converts from the string "13" to the integer 13.

*Figuring out the file names in read_files***()**

The files are named by date. So, e.g., the file with data for Feb 10, 2021 would have the name 2021_2_10.json. You will use the data collected from Jan 14, 2021 through Feb 28, 2021 inclusively.

This means that you must put together the file names to give to *open*(). How do you do that? You can take advantage of your knowledge of loops and of *if* statements. You might keep one variable *month* (which would start at 1 for January) and another variable *date* (starting at 14 for Jan. 14th). Then just keep incrementing *date*, but use an *if* statement to make everything roll over correctly from January to February.

Whichever way you choose, you still have the problem of converting integers (e.g., *month*=2 and *date*=12) to the equivalent file-name string (in this case, the string

"stats_2021_2_14.json"). You can do this by first converting the integers to strings with *str*() and then using string operations, or by using Python's *str.format*().

Going from cumulative cases to new cases by day in your top-level code

Assume that you've merged all of your input files and created a list of cumulative student-case counts with 46 entries (one each for the 46 days).

Next, you can get the number of new cases on any given day by simple subtraction – e.g., the number of new cases on Feb 10 is just the difference between the cumulative case count on Feb 10 vs. that on Feb 9. Of course, the very first day will not have a sensible new-case amount – so you should not use that. I.e., your first day with per-day information will be the *second* day.

What to plot

- Plot two graphs – one for students and one for faculty/staff – in the same figure.
- Do two figures – one for the cumulative data and one for the new-cases-per-day data.
- The *x* axis of your graphs can just be numbers, starting at 0, to indicate the number of days since the start of the data.
- The *y* axis of your graphs is the number of people infected on that day (either cumulative or just the new infections as the case may be).

Challenge problem

The *x* axis of your graph would be more informative if it showed actual dates rather than just numbers. Matplotlib has a *plot_date*() function that does this. For extra credit, you can use this to create a nicer graph.

1. You can use the Python package *datetime* to automate that task. So, e.g., *datetime.date* (2021, 1, 5) creates an object that is the date January 5, 2021. You can increment a date with *d = d + datetime.timedelta* (*days*=1). You can compare dates with code such as **if** (*d <= datetime.date* (2021, 3, 10)) to check if a date is earlier or equal to March 10. And finally, you can access *d.month* and *d.date* to pull the month and date from a *date* object.

Questions

1. Real-world data often has anomolies. You will notice an interesting one in your graphs on roughly day 28 (which is February 11). Can you explain what is so strange about this? (10 points)
2. Can you come up with two possible explanations for the unusual data? (5 points)
3. Presenting the data as shown would probably look quite odd to people. What might be a reasonable (or unreasonable!) way to massage the data so that the data looks less odd? (10 points)

What to turn in

- Your program, *lab3_read_json.py*
- A .pdf with your two graphs and the answers to the questions.

Grading

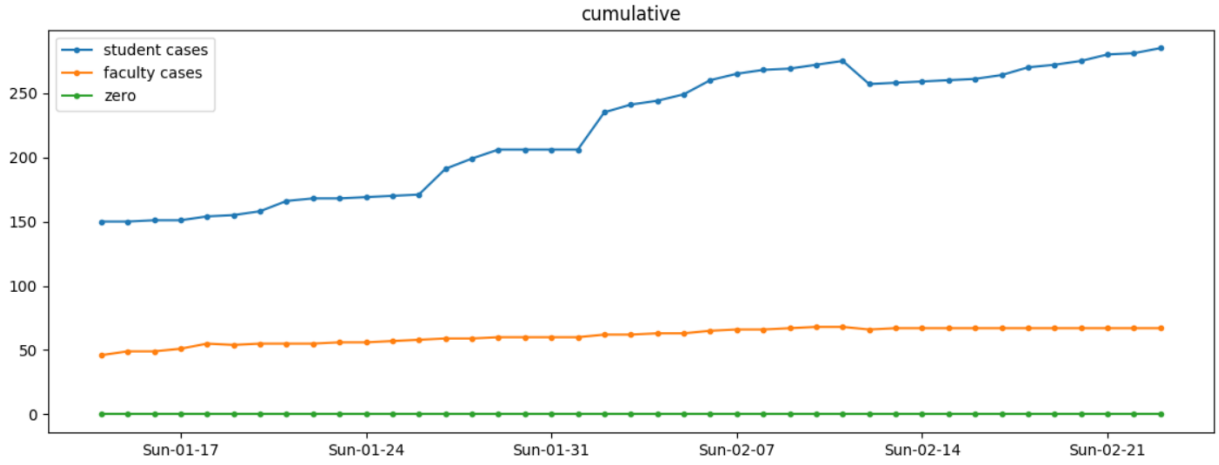
- Code & plot correctness: 65 pts
- Code clarity: 10 pts

- Questions: 25 pts total

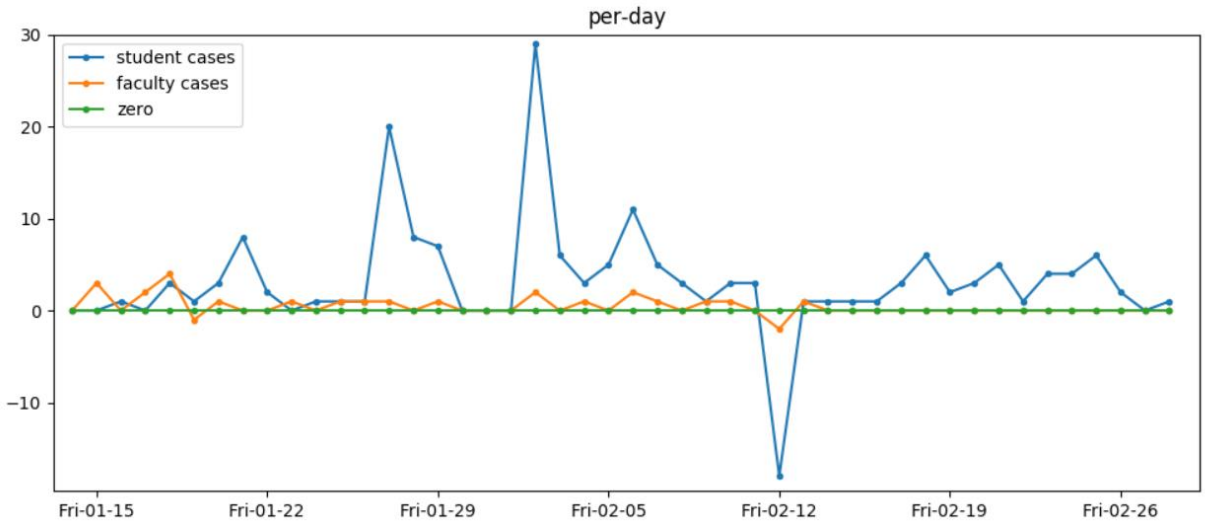
Sample plots

Here is what the plots should look like. However, note that this is the "extra challenge" version that has dates rather than numbers for the *x* axis (also, yours probably won't have all the dots).

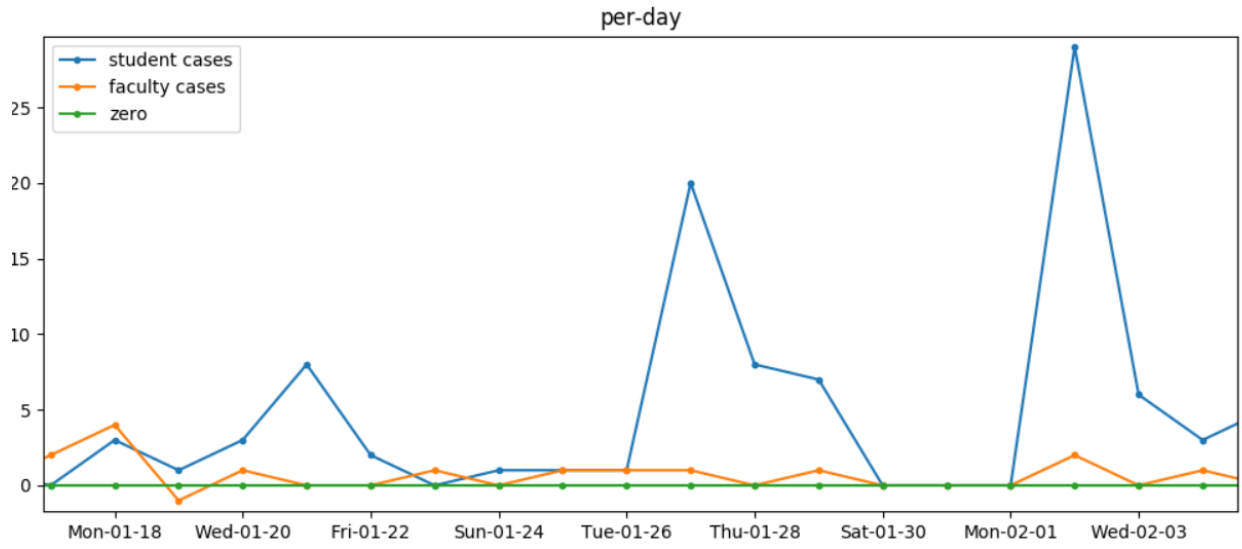
Also, I included an extra line just showing where "zero" is; feel free to do that or not to.



Here is a snapshot of the new-cases-by-day version:



And here's a closeup of the three big spikes:


per-day
student cases
faculty cases
zero
25
20
15
10
5
0
Mon-01-18
Wed-01-20
Fri-01-22
Sun-01-24
Tue-01-26
Thu-01-28
Sat-01-30
Mon-02-01
Wed-02-03