University of Texas Health Science Center at Houston
School of Biomedical Informatics
BMI 5353 Biomedical Informatics Data Analysis

Professor: Kirk Roberts (kirk.roberts@uth.tmc.edu)

Project 2

API Data

This project is designed to go a bit deeper with data access by focusing on API data (specifically the API for Fast Healthcare Interoperability Resources, FHIR). There will be some descriptive and a bit of exploratory data analysis as well (largely descriptive analysis of subsets of the data).

We will be using a free, synthetic FHIR server supported by MITRE. Normally there are security protocols with FHIR (i.e., authentication), but the MITRE FHIR server doesn't require authentication and it isn't important for this lesson anyway (frankly, it can be a hassle so I'm happy it is "insecure"). The FHIR server has 1 million simulated patients, and is located at: https://syntheticmass.mitre.org/fhir/metadata

Feel free to check out the server via HTTP, it's a helpful thing for debugging. You'll also need to learn a decent amount about FHIR for this project. The guest lecture by Dr. Rogith should help, but that won't be all you'll need to learn. "*Figuring out stuff on your own*" is very much part of the data analysis process, so my role as a professor is to give you the high-level theory and point you in the right direction. This is a very good project to interact with your fellow students on the project discussion forum.

You won't need it (I never used it), but I'll point out that it's actually possible to download the code for the server: https://github.com/synthetichealth/syntheticmass

And, they kindly provide you with some documentation as a starting point for learning how to use FHIR: https://github.com/synthetichealth/syntheticmass/blob/master/docs/using-fhir.md

The documentation all assumes you know how to use REST, but that is covered in the *Python Data Analysis* textbook (Chapter 5).

The code setup is similar to Project 1. You will be provided with a **`project2_5353.py`** file that you fill in. I strongly suggest you only edit between the "`Begin CODE`" and "`End CODE`" comments, with the exception of new functions that you create, since this program will be graded with an auto-grader. Actually, I strongly suggest you write a lot of the code in separate functions, as you'll want to re-use a lot of it for other problems.

It might also be a good idea to cache the calls to the FHIR server. I'm sure MITRE would appreciate this, but it will also make your code run faster. Just make sure if you write anything to disk you write it in a way that won't break on a different server (writing to the current directory is fine).

If You Need Help

Projects are designed to get you to struggle through things a bit. That's kind of the point. If every time you don't know what to do your first thought is "*Professor Roberts really did a terrible job organizing this assignment*", then you're kind of missing the point. Working things through is how we grow, data analysis is no exception. Now, there may very well be cases where I have indeed done a terrible job, and I will do my best to own up to those cases, but I still promise you those are a small subset of the cases that students think it so. As such, this section is here to help YOU help yourself. So when there are times that you get stuck, this is the list of steps you should follow. Notice that the *final* step involves e-mailing me and/or the TA, not the first step.

1. **Check Yourself.** Stop and make sure you're doing what you actually thought you were. As with 95% of the issues that people have with computers, it is most likely your fault. So backtrack a bit, double-check you're doing exactly what you intended to, debug your code a bit, and if still isn't working then seek external help. *You should spend at least several minutes this before moving on to the next step* – this will ultimately save you time, as the following steps incrementally involve more time.
2. **Consult the Textbook.** I assigned it for a reason. *You should spend at least a minute seeing if the textbook addresses your issue before moving on to the next step.*
3. **Consult the Documentation.** Each recommended Python library has substantial online documentation, and a significant community that uses it. Chances are if your issue is related to a particular Python library, the documentation might clear up your misunderstanding. *You should spend several minutes trying this if your issue involves a particular library.*
4. **Consult Google.** At this point, probably tens (or hundreds) of millions of people have used Python. Many of these people know then language better than I do. Some of those people happen to use the Internet, so you might want to consult them first. Look at Google (or StackOverflow) for answers to your questions. If your first search doesn't immediately turn up answers, try a different search. As an information retrieval expert myself, I can assure you that reformulating a query can give you drastically different results. So don't just type "numpy broke median" into Google and declare the answer to your question isn't on the Internet if it doesn't show up in the first 5 results. Remember, no one on the Internet is doing this exact project, but your issue is likely more general. *You should spend 5-10 minutes trying this before moving on to the next step.*
5. **Consult the Discussion Forum.** I'll set up a discussion in Canvas for each project. Post questions here so that other students can answer them. *If there are project-specific questions that Google will never find the answer to, then you can jump straight here after Step 1.* But do try to generalize your issue a bit to see if one of the previous steps can help. Give your fellow students time to answer, and remember: if you only visit the forum when *you* have a problem, your fellow students will be unlikely to check it frequently too. I shouldn't need to state this, but I will anyway to be explicit: don't post any actual answers on the forum. *You should try this and wait 24-48 hours before moving to the next step.*
6. **E-mail the TA and Professor.** Our addresses are at the top of the document. *You must demonstrate in the message that you've gone through steps 1-4 prior to e-mailing us.* If you don't, your question will very likely just be ignored.

You might think I've set it up this way because I'm lazy, I don't want to be bothered, and I'm not particularly interested in your problems. That *may be true*, but that doesn't mean it's wrong. When you leave this class and do data mining in the real world, you're not going to be able to rely on me to debug your data analysis issues. You're going to have to figure it out yourself. And the first four steps above roughly approximate what you should do in such a case.

Problems [Total: 100 points]

A. [0 points] **`get_patients(pt_filter)`**
I've implemented this function for you, but I suggest studying it. Returns a list of all the patients in the server (as a JSON object) based on some filter object, **`pt_filter`**, which contains two methods: **`id()`** and **`include(patient)`**. If **`include`** returns `True`, the patient should be included, otherwise the patient should be excluded. This is the function pretty much all of the following code should call on some level or another. Perhaps there's a faster way to implement something than first retrieving all the patients, but in those cases the filter should still be applied. In case it's not obvious already, the auto-grader will use a different filter than the default one provided in the python file.

B. [0 points] **`get_conditions(patient_id)`**
I've implemented this function for you. Returns a list of all the conditions (as a JSON object) for the given **`patient_id`**.

C. [0 points] **`get_observations(patient_id)`**
I've implemented this function for you. Returns a list of all the conditions (as a JSON object) for the given **`patient_id`**.

D. [0 points] **`get_medications(patient_id)`**
I've implemented this function for you. Returns a list of all the conditions (as a JSON object) for the given **`patient_id`**.

1. [10 points] **`num_patients(pt_filter)`**
Return a 2-tuple with (i) the total number of patients on the server, and (ii) the total number of *unique* surnames.

2. [10 points] **`patient_stats(pt_filter)`**
Return a dict of dicts with the distributions (probabilities, not frequencies) of (i) gender, (ii) maritalStatus, (iii) race, (iv) ethnicity, (v) birth year rounded down to 10-year increments, e.g., 1968 → 1960, and (vi) the percentage of patients with and without a listed address. For the keys in each dict, use whatever values are provided by FHIR, except for (vi) in which case you should use the keys `yes_address` and `no_address`. Use `UNK` for any missing values.

3. [15 points] **`diabetes_quality_measure(pt_filter)`**
Return a 3-tuple with (i) the total number of patients with diabetes (use SNOMED code 44054006), (ii) the total number of diabetes patients that have at least one hemoglobin a1c test (use LOINC code 4548-4), and (iii) the total number of diabetes patients that have at least one hemoglobin a1c test greater than 6.0. This is a crude, but somewhat realistic quality measure that could be used by a payer, but also is a nice data quality check.

4. [10 points] **`common_condition_pairs(pt_filter)`**
   Return a list of 2-tuples with the ten most frequent pairs of co-occurring *active* conditions (i.e., active right now, not active together at some past date). The order of the conditions in the 2-tuple should be based on python's built-in sort for strings. In normal practice, one would want to use the codes, but for our sanity's sake we'll use the display names for the conditions. Note: the conditions should be unique, so discard cases where the same condition occurs multiple times in a patient for this and future problems.

5. [10 points] **`common_medication_pairs(pt_filter)`**
   Return a list of 2-tuples with the ten most frequent pairs of co-occurring *active* medications. Please follow the relevant instructions for the previous problem as well regarding sorting and codes. RxNorm also includes other information in the string, so it's fine to include this as well (so different dosages/brands will be unique medications as far as we're concerned). Note that due to what the MITRE server supports, we're only using the MedicationRequest resource, which is a bit different than actual medications taken (e.g., MedicationAdministration), but it's fine for this purpose.

6. [10 points] **`conditions_by_age(pt_filter)`**
   Return a 2-tuple with the ten most common *active*, non-inflammation conditions for patients (i) 50 years of age or older, (ii) under 15 years of age. Assume the current date is January 31, 2018 (i.e., so the first group must be born on or before January 31, 1968, while the second group must be born on or after February 1, 2003). Ignore patients without birthdates. To keep things simple, assume non-inflammation conditions are those that do not end in *-itis*.

7. [10 points] **`medications_by_gender(pt_filter)`**
   Return a 2-tuple with the ten most common active medications for (i) men and (ii) women.

8. [25 points] **`bp_stats(pt_filter)`**
   Stratify patients into those who (i) have normal blood pressure (LOINC code 55284-4) for 90+% of their readings (for our purposes here, we'll say this means their BP readings are within [90, 140] systolic and [60, 90] diastolic), (ii) have more than 10% abnormal BP readings, and (iii) have no BP readings. Return a list of three dicts, one of each for the three groups above, with the descriptive statistics for number of condition SNOMED codes. The keys to the dict should be `min`, `max`, `median`, `mean`, and `stddev`. It should be obvious what these are.