Joshua Rozenberg - Nayya - Software Engineering Assessment - 11/17/2020

Thanks for this opportunity, it's been a fun project and I'm looking forward to getting my hands on some complex real world problems if I am selected for the role.

I've broken up the challenge into 3 parts and defined some assumptions, approach, code, testing, and notes for each. In the interest of time I've simply noted some things that we would want to do in a real world scenario but are left out in this assessment. I've also tried to keep code to a single file per question for ease of use but in the real world each of these tools would obviously have much more infrastructure.

Part 1: Data cleaning

Problem:
Clean the carefirst_bluechoice_doctors.txt and remove all doctors that are inactive. Inactive could mean that the doctor is either not in the NPI Registry or has been deactivated (refer to readme). Please provide a cleaned file in a machine readable data format of your choice (e.g .csv, json, etc) and also the supporting code used to link and clean the datasets in a programming language of your choice.

Assumptions/Notes:
For efficiency and reliability I would want to store the NPI registry in our own database rather than a .csv and update it monthly when a new one is published. I wouldn't rely on their csv to be completely clean either and it would allow you to maintain some additional info about doctors who aren't necessarily on the CareFirst list currently but may have been in the past or have some other relation historically that may be valuable. I cleaned this in a relatively manual way based on the file's issues but if a lot of similar data files were to be cleaned I would invest the time to write a proper parser that can be reused and automated.

- NPI data is assumed correct, duplicate IDs or basic corruption are ignored.
- If there are two NPIs, either being valid would make the provider valid
- The whole cleaning process is bottlenecked by the loading of that huge NPI file initially. It would be beneficial to get or maintain a smaller, more relevant dataset. It currently takes 1.5min to run and only takes a couple seconds with a more localized dataset.

Approach:

- Define a function that takes in a filename for the doctors list, writes out a csv file to the current working directory, and returns void.
- Read in the NPI registry IDs to a dictionary and filter by states to reduce the overall size.
- Read in the initial doctors.tx, check that they are valid, and write out each valid record to a new csv file.

Testing/QC:

- Add some unit tests for this function.
- Would likely want to verify the NPI data before use.

Error handling:

- On read or write failure, throw an exception to the caller with a specific error message.
- On a line read failure, skip the line and print the error, but continue .

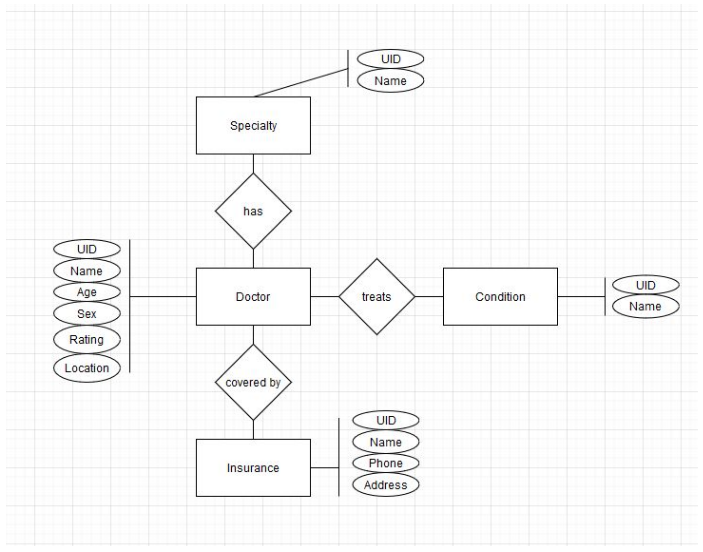
<u>Part 2: API</u>

Problem:
CareFirst is looking to build a service to serve as the API for a “Find a Doctor” search tool. This API should be consumable by a front-end engineering team. Please define the requirements for this API. What are the various endpoints and parameters? How would you design the database?

Assumptions/Notes:
The proper way to design this database is to have separate tables for all the many to many related entities. However, depending on how many “many” is we could possibly get away with trimming down the number of tables and using something like a json field or enum. For instance we don’t really need 2 extra tables if there are 5 known specialties. If there are unknown or dynamic specialties, or there are many options, then we would want those separate relational tables.
If the responses are large (ie. 1000’s of doctors) pagination, caching, chunking, etc. can be implemented to improve performance.

Approach/Requirements:
- Create a doctor-centered database with multiple API endpoints.
- Create an endpoint for users to receive a list of doctors for a set of parameters. POST method at ‘api/doctors’ with a response of type application/json containing a list of the doctors that match the given parameters.
- Create an endpoint to receive a doctor record for a given id. GET method at ‘api/doctor:id’ with a response of type application/json containing a record of the doctor that matches the given id.
- Create an endpoint to edit a doctor record for a given id. PUT method at ‘api/doctor:id’ with a response of type application/json containing the updated record of the doctor that matches the given id.
- Create an endpoint to add a doctor record for a given object. POST method at ‘api/doctor’ with a response of type application/json containing a record of the doctor that was added.
- Create an endpoint to delete a doctor record for a given id. DELETE method at ‘api/doctor:id’ with a response of type application/json containing a confirmation of the doctor that was deleted.

Database ERD:



Testing/QC:
- At the start of this project setup a basic API testing framework that covers each endpoint.
- Location data in the database needs to be guaranteed valid on each change.

Error handling:

- On API failure, send the correct status code and return an error message in the response body.

<u>Part 3: Geodata</u>

Problem:
Additionally, they've asked us to evaluate their current network. Long term they'd like to know which areas are over/under served based on the local demographics, but in the near term - we need to show them we have the data science skills to partner with them. Write a script or function that takes a radius r (in miles) and returns the centroid (latitude/longitude to 3 decimal places) ofthe highest density of in-network active providers in the BlueChoice HMO network. E.g. if the radius is 5, return the latitude and longitude of the center of the most dense 5 miles of providers.Please provide the supporting code in your favorite language.

Assumptions/Notes:
I went with Python and used the Geopandas library for basic geography functionality. I haven't used this library before and it's pretty powerful. However, there are a couple of other good options that have complex spatial analysis tools and functions like this built in. For instance, QGIS has a command line API where you can load spatial data and basically the sky's the limit on analysis. I would also want to store any spatial data in some type of spatial database for efficiency. POSTGIS would be a great option and allows you to do quite a bit of spatial analysis directly in SQL.

- Spatial parts of the data are assumed valid and have already been QC'd and cleaned.
- No duplicate geoms.
- Radius range 1 - 100.
- For large datasets this also will likely require optimization or another approach, and there are a lot of ways to optimize the geographic part of the search using spatial indexing, etc.
- Requires Geopanda library and it's dependencies.

Approach:

- Search for the densest cluster of nodes using basic spatial distance and transformations.
- Make a list of clusters to store the results.
- For each point/node:
  - Make a list to store the cluster.
  - While all points are within the centroid area buffered by r.
    - Record the cluster into results.
    - Add the next closest node to the cluster.
- Take the result cluster with the max nodes as the answer.
- Get its centroid and return a rounded coord.

Testing/QC:

- Add some unit tests for this function.
- Spatial data validity needs to be maintained.

Error handling:

- If we are given an incorrect radius we throw an exception to the caller.
- If there is an error with the spatial data we will also simply throw an exception.