Final Project Report

Changes Made and Functionalities Added/Removed

- Although a feature was removed during stage 2, the direction of the project remains unchanged, the application is still a disease predictor and drug recommendation system with diseases and drug information displayed.
- The functionalities related to doctor type and doctor price suggestions were removed during stage 2 in favor of the current version, which satisfies the stage 2 requirements
- Drug review information was added
- The feature involving checking the most occurring symptoms in the user's location was added
- The layout of the interface was changed from the one in the proposal in favor of simplicity in implementation

Achievements and Failures

- The application has a functioning database that does exactly what was proposed in stage 2 and 3, involving extracting relevant information from the disease and drugs-related tables.
- The application's ml model was a random forest model, which does not output multiple predicted diseases based on one symptom entry, so we were unable to display what other likely diseases the patient might have other than the one suggested disease.
- The way the application takes location data from the user is awkward, they have to input their latitude and longitude themselves, which makes it an impractical feature to use.
- Most drugs in the database did not have side-effect entries since the information was obtained from another data source which has low entry overlap, reducing the usefulness of the drug information display feature.

Schema and Data Source Changes

- All data were obtained from Kaggle
- The schema was decided in stage 2, after the feature changes were decided, and remained the same in the final implementation.
- The dataset for diseases and symptoms was changed in favor of more disease information.
- One of the datasets for medicine was changed in favor of more medical information
- Every dataset for doctor recommendation functionality was removed
- The drugs review dataset was added to use with the drugs dataset

Changes to the ER Diagram and Table Implementation

- There were no changes made to the ER diagram and table implementation after they were designed in stage 2.
- To aid the conversion from binary symptoms entries of the Symptoms table to string symptom names, we could add a table with columns of (symptom_id, symptom_name) where each entry corresponds to a single symptom of a single entry in the Symptoms table, thus having a many-to-one relationship with the Symptoms table.

How the Advanced DB Complement the Application

- The advanced database implementation was most of the application itself since the front end's main job is to display data queried by the database, with only ML disease prediction functionality that did not use the database. Almost every functionality was implemented in MySQL, with Python used to connect frontend and backend.

Each Team Member's Technical Challenges

- Ian
    - Most of my work was centered around writing SQL queries and then translating them into Python for our backend. I also set up the initial connection to the MySQL database which took a bit of googling to get correct as I had never done this before. I had to do a hefty amount of debugging when it came to implementing the queries we planned in Stage 3 as each query returned data in different ways so I had a different method of returning data in a way that was readable for the front end. For the six queries I wrote each had null checks and often some sort of loop to organize the important parts of each query returned. A lot of time was then spent on testing each query extensively with malformed inputs, proper inputs, etc. to ensure our application wouldn't crash or error.
- Max
    - My hardest challenge was when I was trying to implement a function that gets the user's location and finds the top five common symptoms in the area. The function got a time frame, user location, and duration as input. Since the function involved unique data such as geolocation and time, I was having problems with handling such data types and performing operations with them. For instance, the user location was represented as a data type called Point. It was a tuple of two coordinates, the longitude and latitude, and the unit for distance wasn't clarified. Therefore, we had to choose a way to select an area that was within a certain distance from a point. We decided to use the Euclidean distance formula to calculate the area. The distance that the user inputs must be in metric units such

as km or ms so we had to calculate the point's location and the limit of areas that is within the distance.

    - Also, I had trouble setting up MySQL workbench since it was my first time using a database. Learning SQL this semester, and using it to perform advanced tasks took a lot of thinking to decompose the function into smaller steps. Debugging it was also hard and this was done by creating a few dummy data to see if the function was working.
- Nobt
    - Planning the functionality of the application and making sure it fits the stage requirement was very difficult. The implementation requirement of each stage often dictates the design of the application itself, and changes had to be made to fit those requirements. By the time we reached stage 4, the design of our application was far too removed from stage 4 CRUD requirements and I couldn't figure out how to coherently incorporate it into our application design.
    - Working with data from different datasets from Kaggle was quite challenging, as oftentimes the dataset will not have a lot of overlap in entries or the string identifying the entry is different between each dataset, even though they correspond to the same thing ie. same disease with a different name. I had to manually look through the data to see the overlaps and differences between each dataset and process them accordingly, because if those data don't combine well, too many entries will be pruned out to adhere to the constraints of the database, resulting in reduced usefulness of the program. This can be to an extreme point where at one point, the only drugs in our database were the ones for acne.
- Shaurya
    - The front end was the point where every team member's contributions and separate implementations were molded together into a coherent product that integrated and communicated with each other. The first part was setting up all the functions that would implement all the queries connecting the back end and front end. To this end, there was no other way but to manually code up every function, followed by 2 pages for every feature. An input page and an output page, each with its own HTML formatting and input-output functions and separate backend function calls. This was very volume-heavy and repetitive. However, after seeing how everything began meshing together, I felt it was worth it! Then came the App Routing page, the heart of the front end which would translate clicks from the user to actions in the backend and switch from one page to the next, setting up hyperlinks and funneling inputs and outputs between the front and backend. The final touch was implementing the navigation bar, connecting to the database, and setting up the "go back" functionality that allowed the user to go back to the previous page using the arrows in the top left corner of the browser. This would allow them to take the inputs of one feature and place them into another (eg: input symptoms to get a disease, input disease to get drugs, input drugs to get substitute drugs, ratings, reviews, etc). It definitely took a while but once the core functionality was down it was more about fixing up repetitive elements. However, by far the hardest part of setting up the front end was getting the special data

types of SQL to be parsed from front-end user inputs into passable parameters for the SQL queries in the back end. I STILL can't figure out how to convert a POINT data type (which takes in latitude and longitudes) from user inputs of coordinates. This was an unexpected hiccup and heavily crippled our project as it was tied to user IDs, which would have been our method of implementing the updates, delete, and create queries on the user data. We couldn't do this because we just couldn't get in user coordinates without crashing the SQL engine. This was very frustrating, especially with a lack of documentation on how to achieve this.

    - Another technical challenge was parsing through the multiple project datasets and creating new ones with all the features we wanted in them. This required a lot of data processing in Pandas because we needed to mix and combine different datasets in order to fuel the features we wanted to implement. There were many on Kaggle but rarely did they overlap enough to be used on their own.

Future Improvements (other than the interface):

- CUD part of CRUD:
    - Once we get the user inputs working for our location page we could add a userID output that allows the user to come back and remove themselves from the dataset so that the location can be shown as “getting better”.
    - Updating user location for more symptoms or changes in location.
    - All of this can be done by giving the user a user ID which we already had.
- Expanding the dataset for more values by finding and refining more datasets.
- Make the location feature take in their location automatically without having to manually insert the coordinates, so they only have to submit their symptoms.
- Make a comprehensive page that gives a summary of all our separate features rather than have the user go through them one at a time.
- Allowing the admin to add, delete, and update diseases, drugs, and symptom database tables.

Final Division of Labor

Given that the result of the project was not fully complete, the management of the teamwork was not entirely well done. We managed teamwork in a more divide-and-conquer approach without proper identification of what should be done first before other parts could proceed, which led to tight scheduling near the end of the project. There was also uneven work distribution among team members.

- Shaurya
    - All of the front end, 3 back end functions, 2 advanced Queries, and Drugs table. ER/UML Diagram creation with Nobt, reviewing project proposals with Nobt,

creating and uploading work on GitHub after stage 2, and making subsequent releases from GitHub. Writing/ reviewing final report.

- Nobt
    - Responsible for deciding the direction of the project ie. drafting the project proposal, making design changes for each stage, drawing the schema's UML diagram, and designing what each query should do and which ones will be advanced before distributing the implementation of the queries between each member. Writing/ reviewing final report.
- Ian
    - Wrote 2 advanced SQL queries and 2 regular SQL queries, translated 7 planned queries into Python, and set up a helper function for connection to our GCP thus integrating the backend to the frontend. Did extensive stress testing of our queries to ensure correctness.
- Max
    - Wrote 1 advanced SQL query and 2 regular SQL queries, and wrote the logic to get the user input to perform the area search for common symptoms. Extensively searched to debug our top symptoms in an area query. Made the SQL queries into Python functions so that the result could be retrieved by the front-end.