ECE 356 Project - Group 34

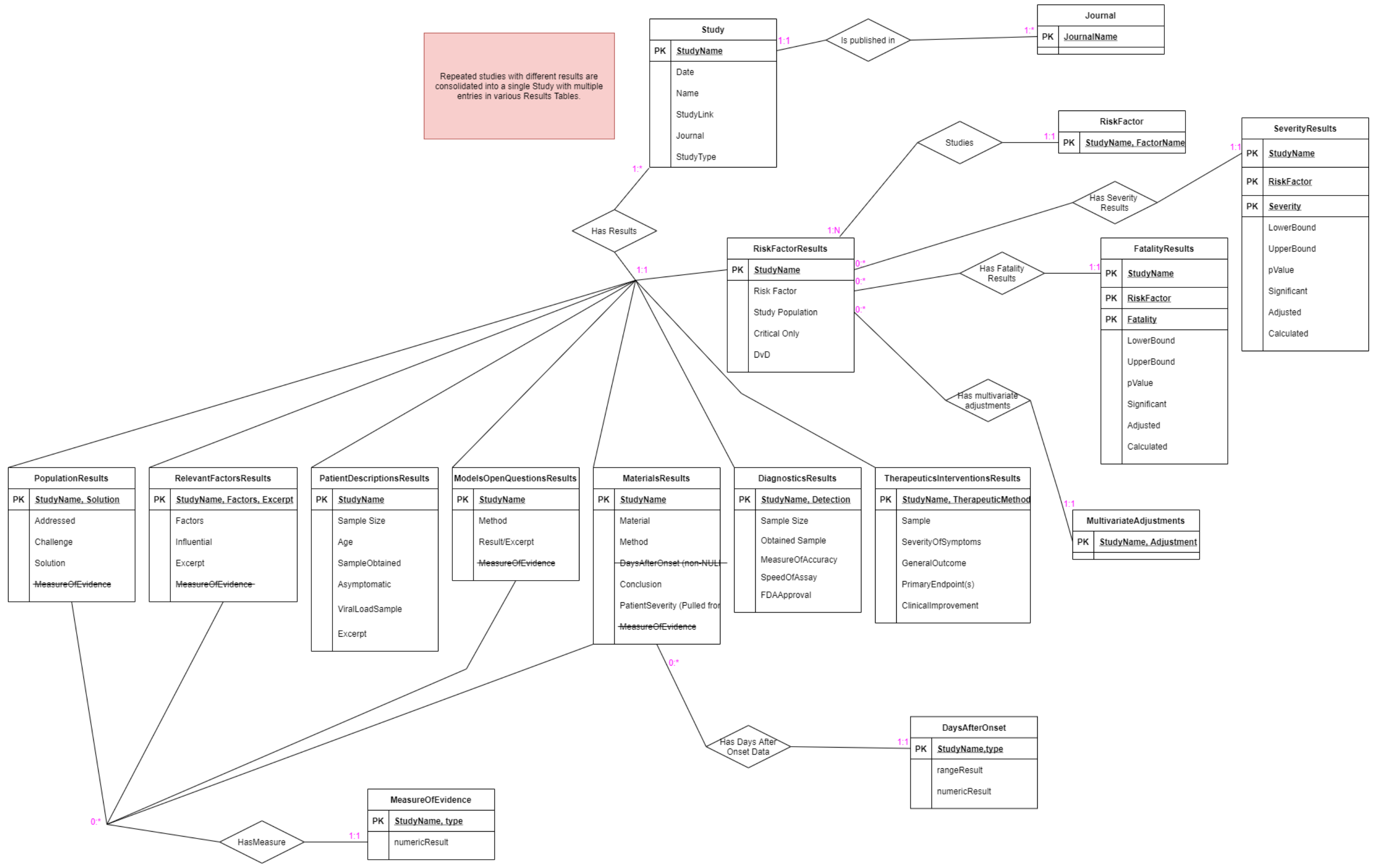
Report

Ian Steneker - 2070869
Abdulmateen Shaikh - 20726710
Hariharan Karthikeyan - 20734710

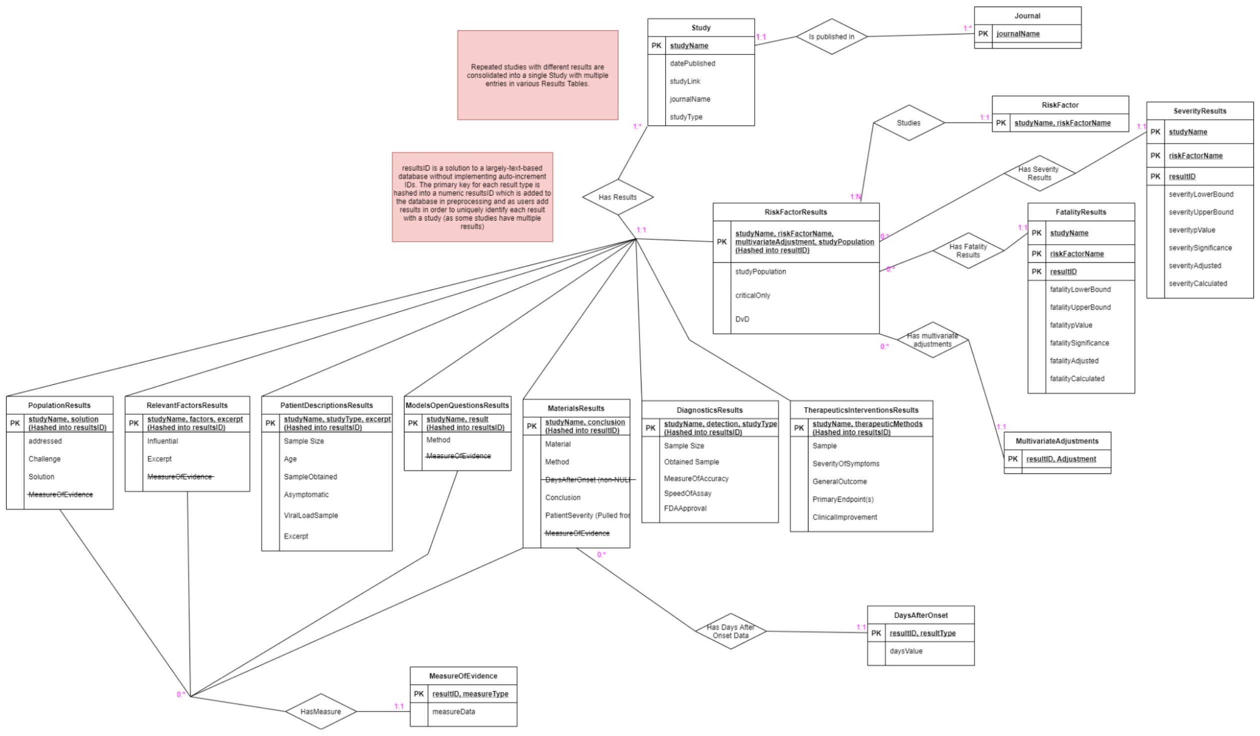
Database Design

Entity-Relationship Details

When initially designing our entity-relationship diagram, we decided to do get it as close to an effective relational schema as possible in order to avoid having to do extra work later on in conversion. This was primarily done by identifying multivariable attributes such as MeasureOfEvidence, DaysAfterOnset, and MultiVariateAdjustment. As stored in the provided dataset, these were all delimited lists using various delimiters and formatting which we planned to rectify in preprocessing. As such, our preliminary ER Diagram was the following.



This included all selected primary keys which were selected as a function of examining the dataset and finding attribute collections which could be used to identify the results for a given study. One might also notice that Each of the result entity sets are related to the Study set. This is because after identifying that each study type had a shared collection of attributes which helped to identify the study, but had different types of data attributes, that a top-down hierarchical relationship was discovered and so the 8 different result type entity sets were born, each related to Study. After using this preliminary ER diagram to develop a relational schema and write the preprocessing and SQL creation code, a second ER diagram was created to more accurately reflect the final results of our project. This more accurately reflects a translation from our relational schema back to an ER diagram.



One can see that not much has changed in the structure of the entity-relationship diagram, but the introduction of resultID being hashed from primary keys resolved some rather large primary keys.

Preprocessing

The dataset received from Kaggle was inconsistent and would have been extremely difficult to load into any kind of relational database without a not-insignificant amount of preprocessing. This preprocessing was done with python and the data analysis tool pandas in the *preprocessing.py* script. The script is well-documented and can be inspected freely but a breakdown of the steps taken include

1. Delete all CSVs each study type which do not align with the table definitions specified in *Kaggle>target_tables>0_table_formats_and_column_definitions>list_of_tables_and_table_definitions* for each table type (1-8)
2. Move a singular table which was in the wrong study type
3. For all tables in *5_materials*, split out PatientSeverity from the composite MeasureOfEvidence attribute and add it to the Conclusion attribute for generation of a primary key later.
4. For all tables in *2_relevant_factors*, clean up column names and drop all non influential rows which have no excerpt as they contain no meaningful data.
5. For all tables in *3_patient_descriptions*, clean up the Asymptomatic columns due to different names, and convert them all to a probability score between 0 and 1 (0 being no, 1 being yes)

6. For each study type, rename all columns that are slightly misnamed so they can be merged into a single table for each study type. Additionally, fix any bad date formats. Then for each study type, concatenate those tables into a single table for loading into the database, and generate a resultID for each study result based on a hash of its primary key elements. This allows us to use a non-auto-increment primary key without relying on large text segments for said primary key which could hurt the performance of the database.
7. For each study result type with a multivalued MeasureOfEvidence attribute, split that attribute into a separate table via a series of delimiters, each with its own type, measure, associated studyName, and resultID
8. For each risk factor with a multivalued MultivariateAdjustment attribute, split that attribute into a separate table via a series of delimiters, each with its own Adjustment, resultID, and associated StudyName.
9. There are some duplicated studies in the aggregated *5_materials* table which all link back to the same study. We can delete all but the first instance of this study, and break the multivalued DaysAfterOnset attribute using the same function as was used for for the multivalued MeasureOfEvidence attribute.
10. The cleaned and updated tables are then published to CSVs, ready to be loaded into a database with some further processing once there to clean them up and establish the appropriate constraints.

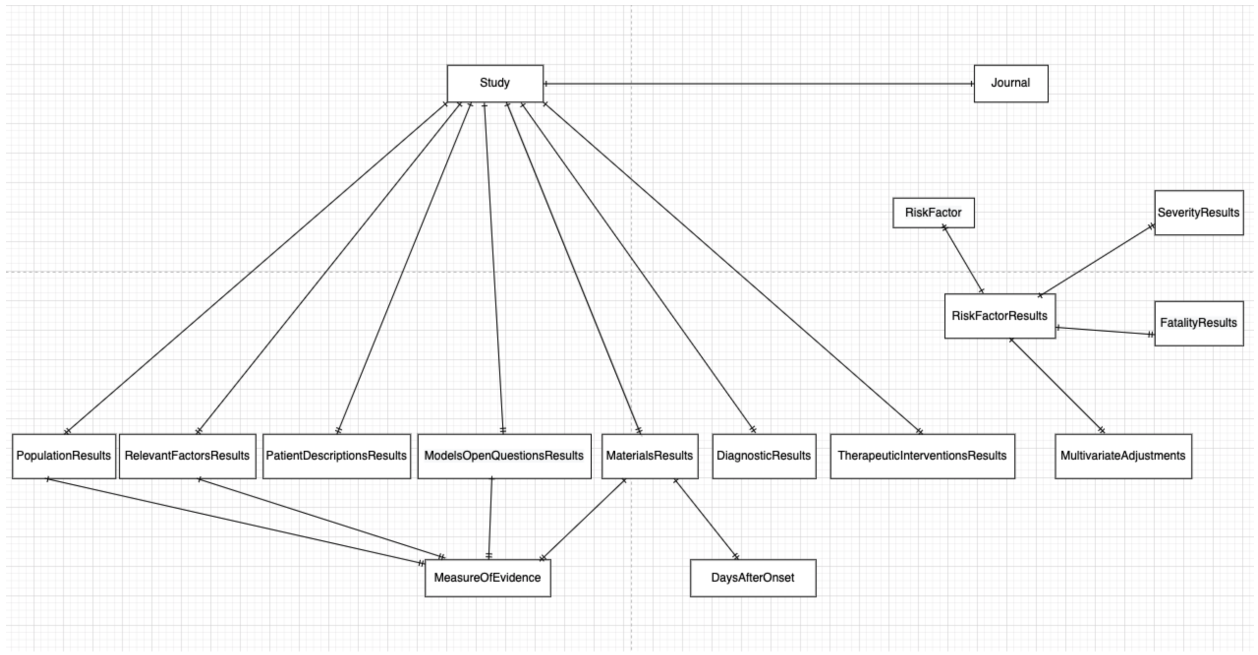
Relational Design Choices

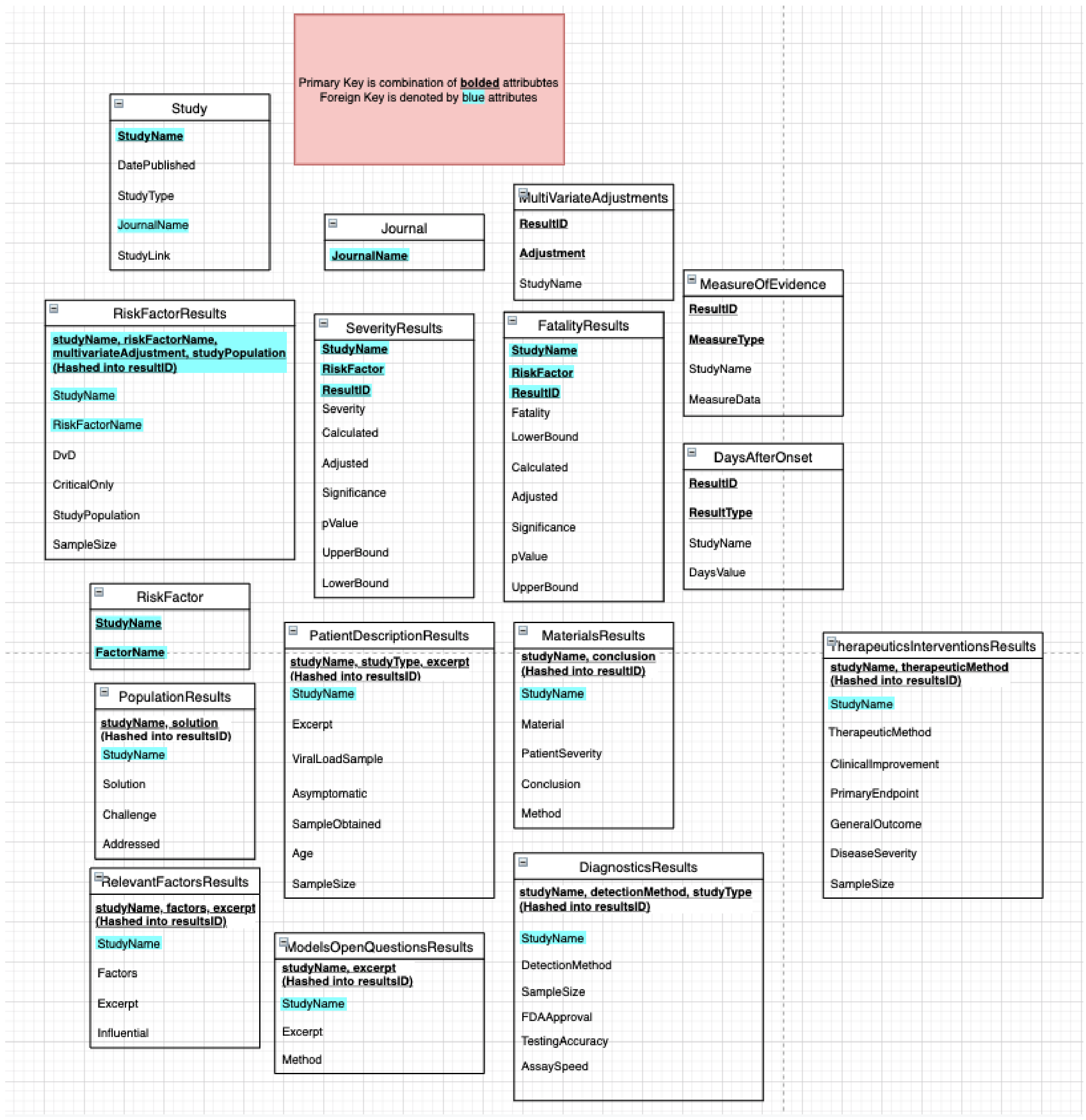
For our relational design, we initially started converting our entities into tables as specified by the initial ER model diagram. We specified key constraints including primary keys for each table and represented the relationships in the ER model diagram by creating corresponding foreign keys. However, we soon realized that our chosen primary keys didn’t accurately represent the data and were rather large. Therefore, we soon switched to hashing our primary keys to generate a resultID which then became the primary key for all of our results tables.

For the table creation and data loading process in MySQL, we first created all the results tables with the hashed key resultID as primary key. The data type and size for each of the columns was chosen after performing an analysis on the raw data. All these tables were loaded from the preprocessed csv’s we generated using the ‘load data infile’ command. The three small tables: MultivariateAdjustments, MeasuresOfEvidence, and DaysAfterOnset were loaded in a similar fashion and appropriate primary keys were chosen for these tables after analyzing the data and determining unique columns.

Once the RiskFactorResults table was populated, we populated RiskFactor, SeverityResults, and FatalityResults by running an ‘insert into … select … from RiskFactorResults’ command. And finally when all the other tables were generated and populated, we ran a similar ‘insert into Study select … from ...’ for each of the other results tables to produce our Study table. After this process completed, we realized there were several duplicate studies. In order to remove these duplicates, we created a temporary auto-increment primary key called studyID. We then got rid of duplicate studies by comparing against this studyID. Finally, we dropped the studyID column once duplicates were removed and added the appropriate primary key back to the Study table (studyName). The last table to be loaded was the Journal table where we ran the ‘insert into Journal select … from Study’ command.

Since all the relevant tables in our database were now correctly loaded, we dropped irrelevant columns from each of the results tables and then added foreign keys to enforce the appropriate relationships. For example, each of our results tables had a foreign key on studyName references Study(studyName). As the final step of the process, we added ‘not null’ checks for the relevant attributes and dealt with NULL values. We also added checks to ensure data validity and integrity constraints for consistency as well as error reduction. In order to improve query efficiency, we added important indexes that resulted in faster execution time. This was done by testing several queries using the ‘explain’ and ‘explain analyze’ commands learned in the lectures. Our final table specification (showing one-to-one and one-to-many relationships between tables) and relational schema diagram is as shown below:





Client Application

Intended Use

As a command-line program, our program is intended to be used by researchers and scientists who, while not data scientists themselves, are sophisticated enough with computers to access a command-line interface and maybe desire to run some queries they've written themselves or have been given by colleagues. It is intended to be an easily-searchable database of studies for

COVID-19 such that studies can easily be classified and searched for by risk factor, or their intended purpose of analysis, or by a substring of the title. Being entirely text-based, this functionality is extremely fast and allows researchers to rapidly aggregate studies they are interested in where they can follow the links associated with the studies to examine them for more information regarding their area of interest

Interface & Menu Structure

The client application is a command line program built using Python that connects to a MySQL database. As this program runs in the command line, it does not have a graphical user interface and instead users will see instructions and available choices to them with a number associated with each one. To select a choice, they can choose to either type the number or the name of the option. Any data from the database is shown in a table format using the help of a python library called PrettyTable. The program starts with a main menu with 8 options a user can choose. These options are 'show', 'add', 'edit', 'delete', 'direct SQL mode', 'export', 'help' and 'quit'.

'Show', 'add', 'edit' and 'delete' are equivalent to the CRUD operations on the database. When any of these options are selected the user is asked to specify a category to perform this operation on. The categories are 'study', 'journal' and 'result'. After selecting the category the user will be asked to enter required and optional parameters. This will need to be entered one by one and in the case of optional parameters, leaving a parameter blank is allowed. In the COVID-19 dataset, the study names and journal names are very long and it would not be practical for a user to enter the entire name. So, for anytime a study name or journal needs to be entered, a user can type only part of the name and they will see a list of possible studies/journals that match what they entered. They can then enter the number associated with the study/journal they want to choose. 'Show' takes into account joining the appropriate tables when needed. 'Add' and 'edit' take into account foreign key relationships and will automatically add data to the appropriate tables. 'Delete' also takes into account foreign keys and also cascades deletes to ensure no FK constraints are broken. For example, if a user deletes a study, the application will delete all results associated with that study before deleting the study itself.

The other options in the menu include extra features that users have access to that will be useful for them. 'Direct SQL mode' is for advanced users who have a basic understanding of SQL. This option allows the user to write whatever SQL queries they would like to run against the database. They are restricted however from deleting rows or truncating tables. 'Export' allows users to export whatever data they see from the 'show' function to a text file. This function works similar to the tee function in MySQL, where a user can start exporting by typing "export start" and it will continue to add to the file until they type "export end".

Finally, 'help' is a function that displays instructions to the user on how to use these various functions and 'quit'' allows them to quit from the application.

Tests & Testing Methodology

There are 4 main tests that are related to each of the CRUD operations, and 1 test related to the direct SQL mode and 1 test related to the export function.

Test Case #	Description	Steps	Expected Result
1	Tests SELECT and joins	1) Go to show -> results -> PopulationResults 2) Enter studyName = "Pop"and choose option 1	- 1 row with columns from PopulationResults table and MeasuresOfEvidence table
2	Tests INSERT and FK	1) Go to add -> results -> RiskFactorResults 2) Enter studyName = "Risk" and choose option 1 3) Set adjustment = "test" 4) Go to show -> results -> RiskFactorResults 5) Enter studyName = "Risk" and choose option 1	- 1 row with the adjustment value = "test"
3	Test UPDATE and FK	1) Go to edit -> results -> RiskFactorResults 2) Enter studyName = "Risk" and choose option 1 3) Set adjustment = "changed adjustment" 4) Go to show -> results -> RiskFactorResults 5) Enter studyName = "Risk" and choose option 1	- 1 row with the adjustment value = "changed adjustment"
4	Test DELETE and FKs	1) Go to delete -> study 2) Enter studyName = "Risk" and choose option 1 3) Go to show -> results -> RiskFactorResults 4) Enter studyName = "Risk"	- no study name with that name available
5	Test Direct SQL mode	1) Select "Direct SQL Mode 2) Enter "DELETE FROM PopulationResults"	- see error message "Error: Invalid command"

6	Test Export Function	1) Type "export start" 2) Go to show -> results -> PopulationResults 3) Enter studyName = "Risk" and choose option 1 4) Go to show -> results -> RiskFactorResults 5) Enter studyName = "Risk" and choose option 1 6) Type "export end"	- text file with timestamp is created in directory - file has 1 row from PopulationResults table and 1 row from RiskFactorResults

Data Mining

Question to answer

A very useful question we wanted to answer using our dataset is secondary risk factors besides age that pose a significant risk of increased severity of COVID-19, or increased fatality risk. This question can be answered using FatalityResults and SeverityResults which are derived from studies with RiskFactorResults by classifying severity and fatality odd ratios by their associated risk factors.

Techniques & Technologies Used

The primary techniques used were association and classification. The risk factor results were inherently broken up into classes by their associated risk factor when they were loaded into the database. As part of data mining these tables are exported to CSVs. The tables are then loaded into the data mining tool Orange3 and processed to remove their type of odd ratio (crude or adjusted) since the odd ratio format is inconsequential with regards to aggregating the data and extracting trends. The tables are then classified according to their RiskFactorName, and then four distributions are taken for analysis, two for severity and two for fatality. One of these two for both measures is the unaggregated results for the distribution of frequency of risk factors across various fatality and severity results, whilst the other represents the median results for the distribution of frequency of risk factors across fatality and severity.

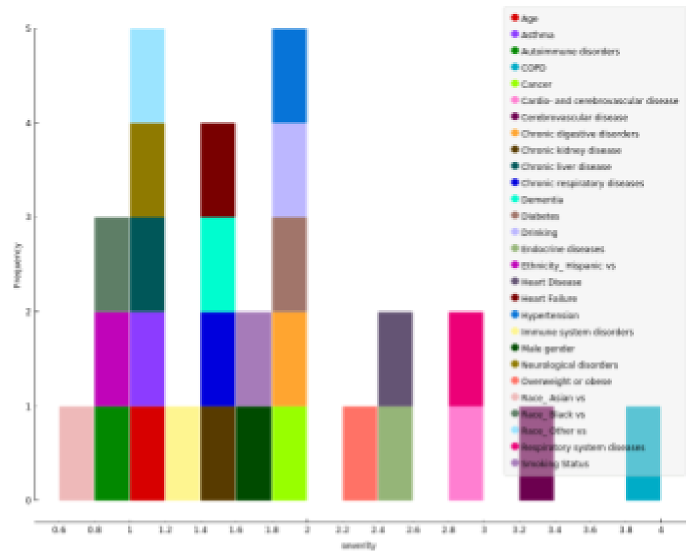
Results

The results of data mining are quite interesting for planning vaccine rollouts. Besides Age, there are a number of risk factors that could increase the severity of covid symptoms or increase the risk of fatality for those diagnosed with COVID-19 which can be identified through classification of risk factor by fatality/severity odds ratio.

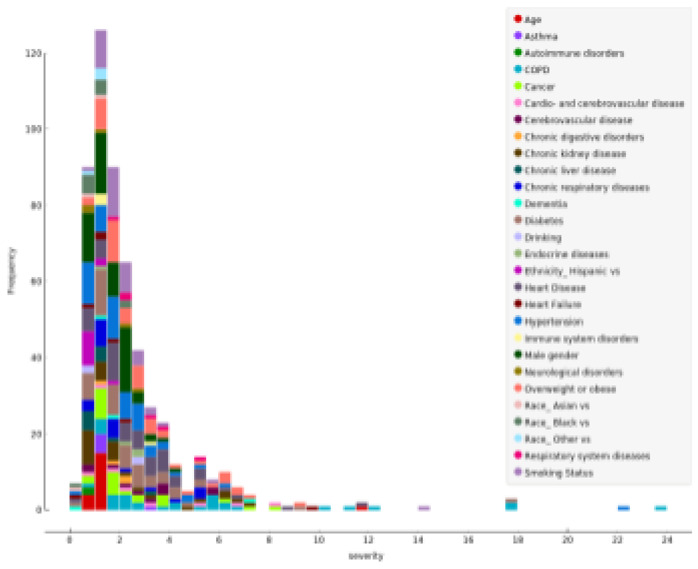
Please note that any risk or odds ratio over 1 represents that a risk factor represents a greater danger to the affected population than a healthy control group, but only the most interesting risk factors will be discussed in the accompany results paragraphs.

Risk Factors affecting severity of symptoms

One can see that from the studies performed on risk factors in our dataset, COPD (Chronic obstructive pulmonary disease) is the primary risk factor that increases the severity of COVID symptoms, while asthma, cardiovascular, and pre-existing respiratory diseases are also major concerns. This makes sense given the destructive effect COVID-19 has on the lungs. Surprisingly, age does not seem to be a major risk factor for COVID-19 symptom severity, with a median odds ratio between 1 and 1.2, however those with advanced age are more likely to have pre-existing risk factors such as COPD and other increased symptom severity risk factors, which may have been excluded from the studies focused on age.



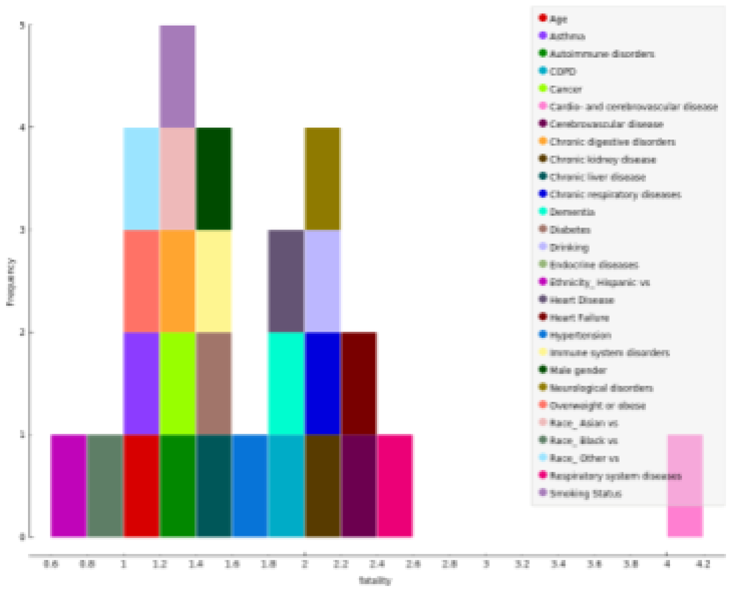
The aggregated (median) severity of symptoms distribution classified by risk factor



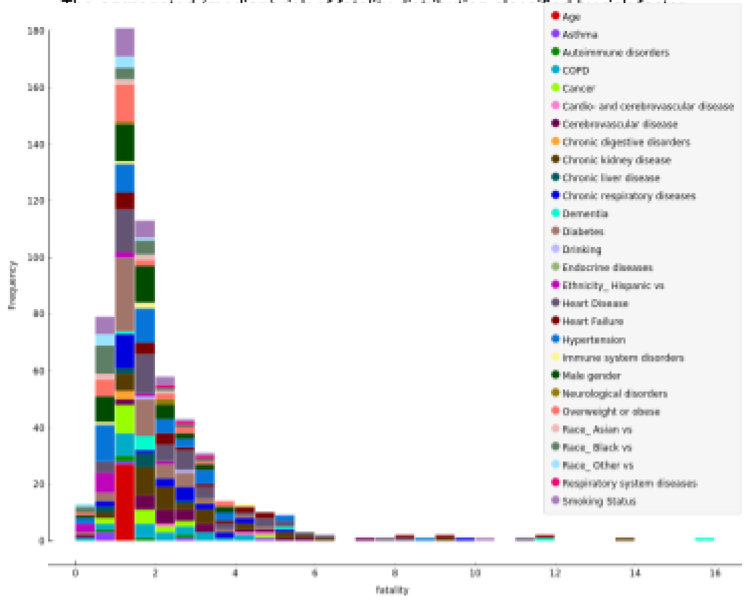
The unaggregated distribution of severity of symptoms classified by risk factor

Risk Factors affecting fatality

One can see from the distributions for fatality that while a symptom may indicate increased symptom severity, that same symptom may not increase risk of fatality to the same extent. COPD, while still a danger with a median odds ratio between 1.8 and 2, is much less of a fatality risk compared to cardiovascular, cerebrovascular, and respiratory diseases. However, comparing the unaggregated results with the aggregated results does somewhat show that cardio and cerebrovascular diseases may be somewhat of an outlier in the aggregated results due to a lower number of studies concerning themself with that particular risk factor. Interestingly, in the unaggregated results, both dementia and heart failure have a significant range of recorded fatality ratios. This again may have an indirect correlation with age and other pre-existing risk factors if studies were not conducted in isolation with respect to risk factors, thus polluting the data.



The aggregated (median) risk of fatality distribution classified by risk factor



The unaggregated risk of fatality distribution classified by risk factor

Validation

Unfortunately, our dataset is not large enough to perform accurate validation with a separate training set. However, many resources exist which have identified the more important risk factors for COVID-19. The CDC has a list of conditions[1] which may require those afflicted who contract COVID-19 to seek hospitalization, many of which align with the risk factors described here. Additionally, the Mayo Clinic has a more restrictive list[2] which lists age, lung problems (including asthma), heart disease, diabetes, and more as at-risk populations for more severe symptoms require more intense care. Many of these external validation results may be based on the same or similar studies that are part of our dataset, but can still be used for validation due to the added analysis of subject matter experts.

References

[1]
https://www.cdc.gov/coronavirus/2019-ncov/need-extra-precautions/people-with-medical-conditions.html
[2]
https://www.mayoclinic.org/diseases-conditions/coronavirus/in-depth/coronavirus-who-is-at-risk/art-20483301