Project Summary

Additional Feature

The additional feature measures the factor of the willingness of local people to be fully vaccinated against COVID. We assume that the fully-vaccinated number is affected by two factors: a.) the economic status of a person, in our case represented by the livable area per capita; b.) the willingness to be fully vaccinated. Specifically, the factor is calculated based on the total fully-vaccinated number (covid_data file) per capita(population.txt) being divided by the total livable area(properties.csv) per capita via Zip Code. By dividing the livable area per capita, we rule out the economic factor (the objective factor), thus leaving only the subjective factor, which is the willingness of local people to be fully vaccinated.

The factor is used against an assumed standard factor. The higher the output over the standard, the more willingness of the local people in that Zip Code to be fully vaccinated against COVID. The same mathematical formula is applied to calculate the standard factor. We calculate the constant standard factor to be 5.56, whereas the accuracy of the local factor is guaranteed by comparing outputs against the same output calculated by using excel formula.

Use of Data Structures

We have used the following three data structures throughout the program design.

1.Tree(TreeMap). For instance, we use TreeMap to calculate the value of vaccination per capita. We use TreeMap, instead of HashMap, specifically for the purpose to print out the Zip Code in ascending numerical order (please refer to vaccinationsProcessor class).

2. LinkedList. We use LinkedList to extrapolate data points respectively from three data sets for the ease of cross-referencing (SQL). E.g., all data sets have the variable of zip code. We use this dynamic linear structure because it can be expanded or shrunk dynamically based on size, thus it is more convenient for storing the data to be used. Our decision to use LinkedList over ArrayList is because we

are dealing with some large lists of data (particularly from properties.csv) which require a great amount of insertion. LinkedList works better since it doesn't have to double in size when the max is reached. Also, our implementation do not use index searching to get element from these particular lists, so the advantage of ArrayList is not evident here.

The three reference-data include covid data, property data, and population data. These data points refer to date, zip code, partial vaccination, full vaccination, population size, market value, and livable area.

3. ArrayList. We use ArrayList to store the index number of the required data when parsing the csv file. Please refer to the helper function getIndex() method in the following class: CSVCovidReader and JSONCovidReader. We store those target numbers in the ArrayList and get each of them by their index. We choose ArrayList over other List structure because searching and retrieving data by index is most convenient for ArrayList, as it requires only O(1) complexity of time.

4. Array. We use array to parse a string of data. Take populationreader.java class for instance. We use array to hold data points of zip code and population size after using the string split method. The reason for us to choose this data structure is two-fold: 1. It is easy to manipulate data using array, as the string is fixed size; 2. We have used the same method to read and parse flu tweets in our previous assignment. Please refer to the following classes for detail: propertiesreader, populationreader, jsonreader, and csvreader.