Process Book

Background and Motivation

We will be creating an interactive visualization of food inspection data from the Salt Lake County Health Department. This visualization will be helpful as it informs possible restaurant patrons about the overall cleanliness of establishments at which they may dine. We have also found the information in these data humorous, in a sardonic way. For example, we saw a sushi burrito restaurant that received over 70 health code violations in the past four years. Additionally, one of our group members is particularly interested in urban geography, and this project has an urban geological component from the spatial context of the establishment locations. Furthermore, we determined that this dataset was accessible to us because the Salt Lake City Health Department provides thorough definitions for the attributes of these data.

Project Objectives

The primary questions that we are trying to answer are as follows:

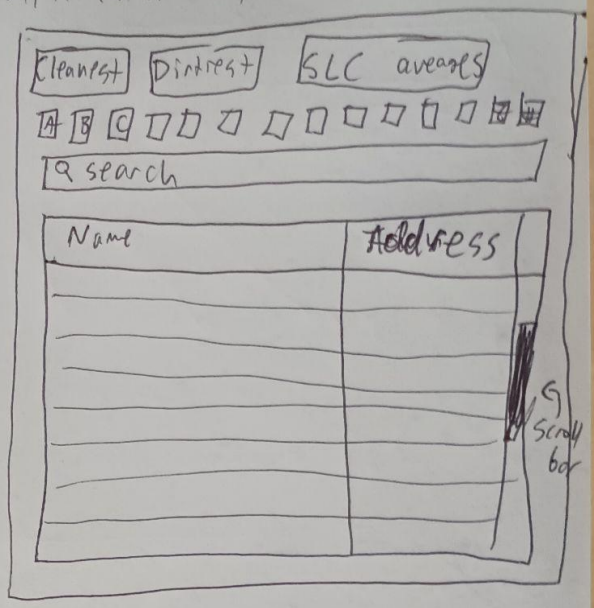
Which local restaurants and restaurant chains in Salt Lake County are the cleanest and least clean? What is the distribution of code violations for a given restaurant? How has restaurant compliance with the health code changed over time?

We aim to provide a visualization to help potential restaurant patrons make informed decisions before they dine. The benefits to this are immediate: such a visualization will help customers make informed decisions about where to dine in terms of any restaurant's sanitation. Additionally, one can see whether a restaurant has improved or degraded in health code compliance over time.

Visualization Design.

Found below, the first two photos are Nathan's prototype (Figure 1), the next two are Eliza's prototype (Figure 2), and the next two are Arleth's prototype (Figure 3). The final two photos are our final designs (Figure 4). Figure 4(a) corresponds to the must-have features we thought to implement first, and Figure 4(b) corresponds to the optional features we agreed upon. Our prototypes were based on the features we decided upon, and we conceived of our designs based on these features. The agreed-upon features to consider were a menu containing the establishments, visualizations containing aggregate statistics of the inspection data, and visualizations of individual establishment statistics.

Minimal Implementation



Common codes for Sushi Burrito:

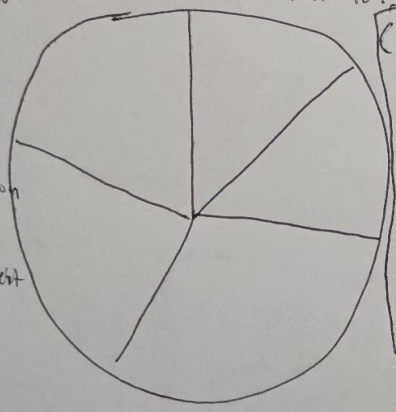
4.1 Worker certification

4.2 Management

...

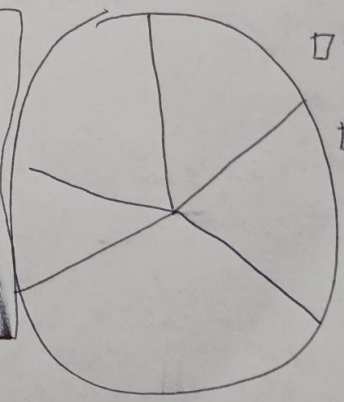
...

other



Click on a slice on the left pie chart to change the right pie chart. This text would not be in the display

4.2 Management & Personnel



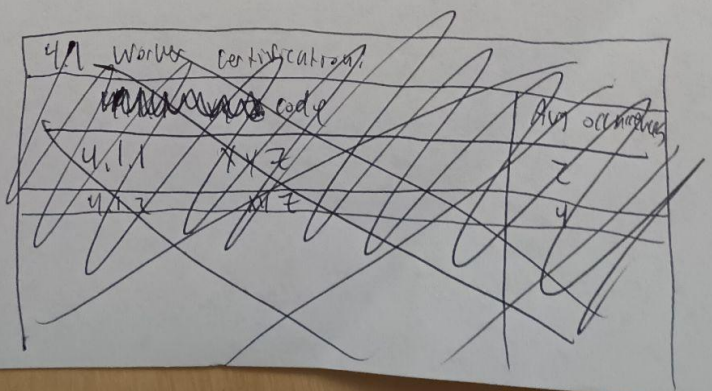
4.2.15 Tobacco

4.2.12 hand washing

...

...

...



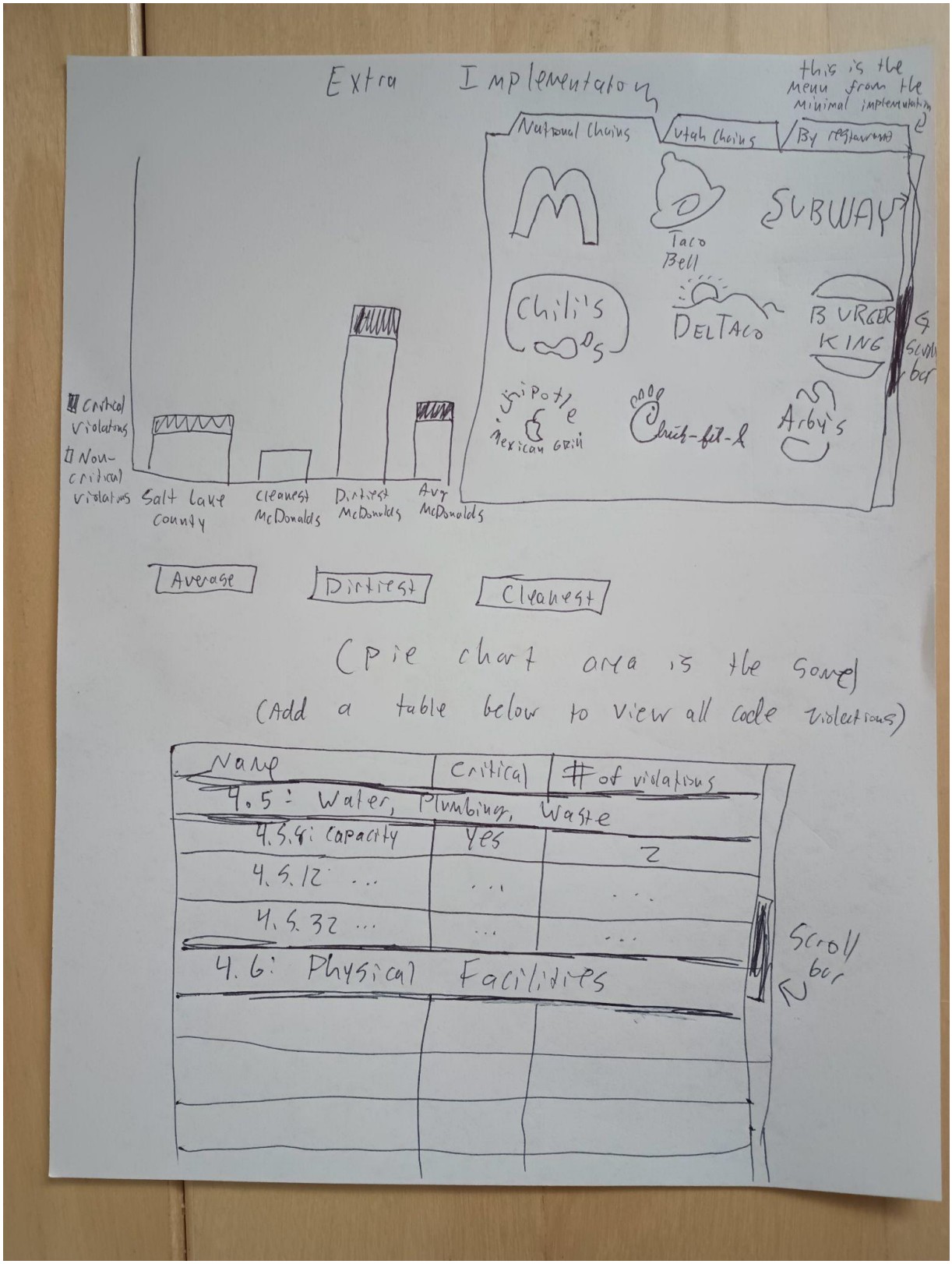
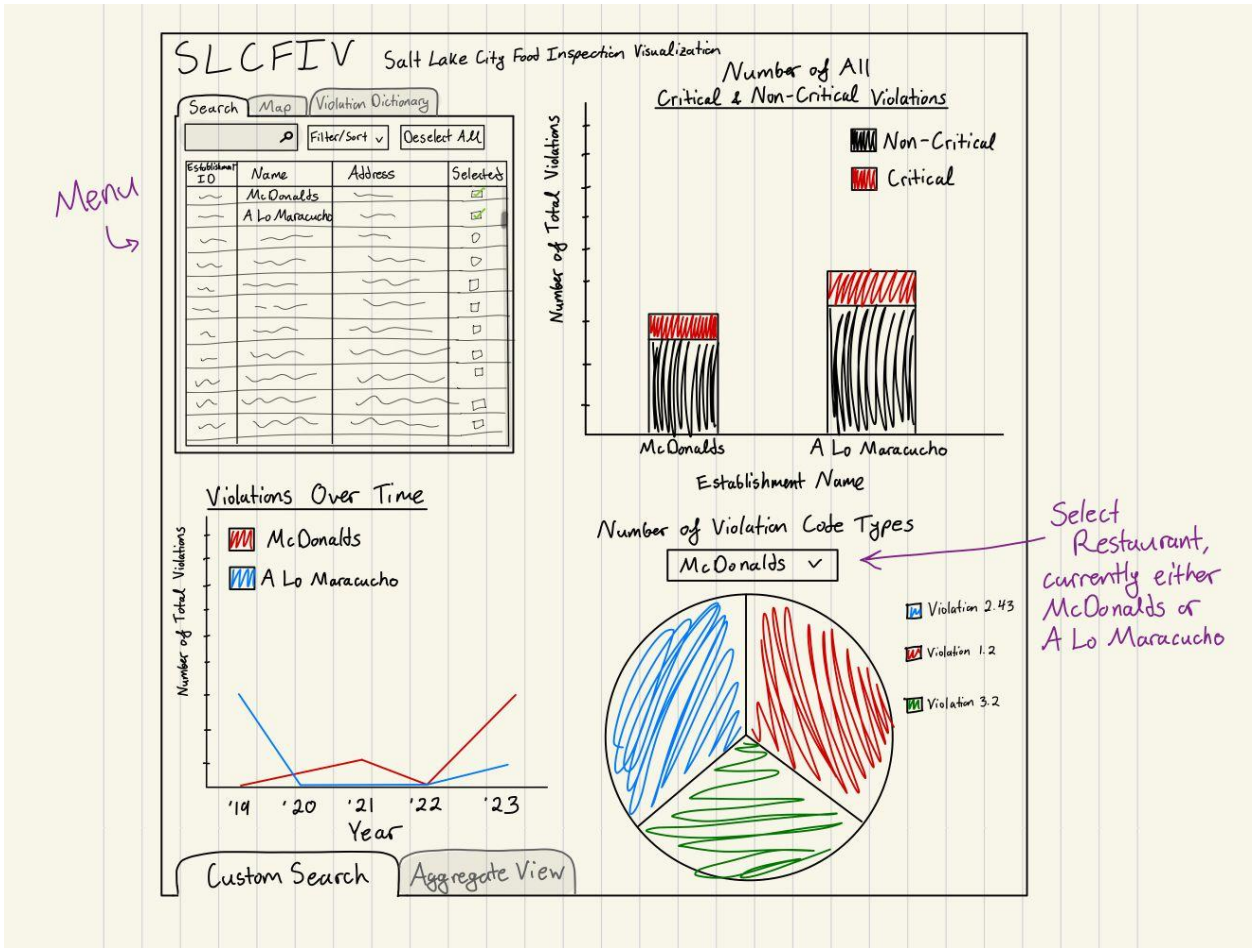


Figure 1: Nathan's Prototype 1 & 2

Figure 2: Eliza's Prototype


SLCFIV
Salt Lake City Food Inspection Visualization
Search
Map
Violation Dictionary
Filter/Sort
Deselect All
Establishment ID
Name
Address
Selected
McDonalds
A Lo Maracucho
Menu
Number of All Critical & Non-Critical Violations
Non-Critical
Critical
Number of Total Violations
McDonalds
A Lo Maracucho
Establishment Name
Violations Over Time
McDonalds
A Lo Maracucho
Number of Total Violations
'19
'20
'21
'22
'23
Year
Number of Violation Code Types
McDonalds
Violation 2.43
Violation 1.2
Violation 3.2
Select Restaurant, currently either McDonalds or A Lo Maracucho
Custom Search
Aggregate View

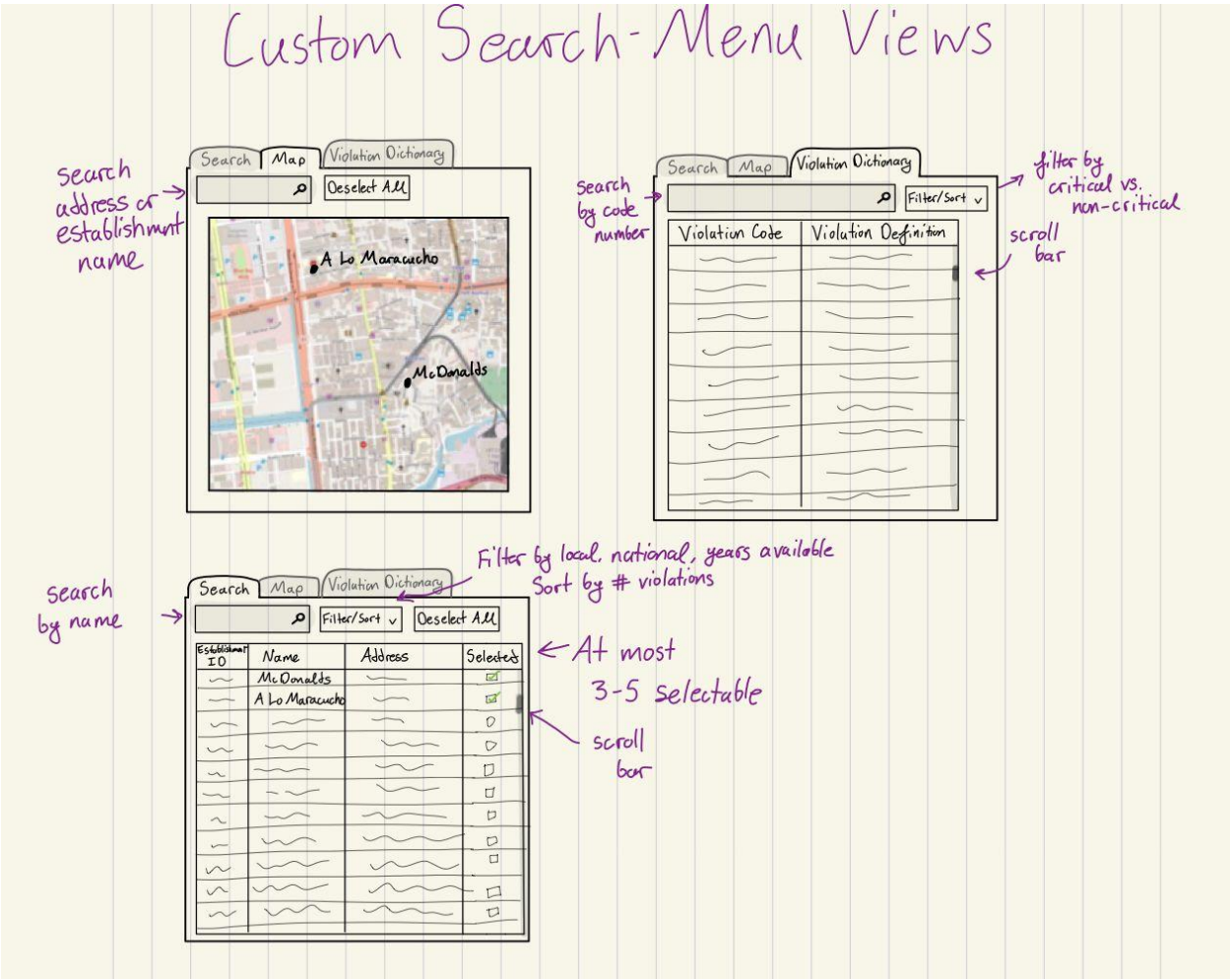
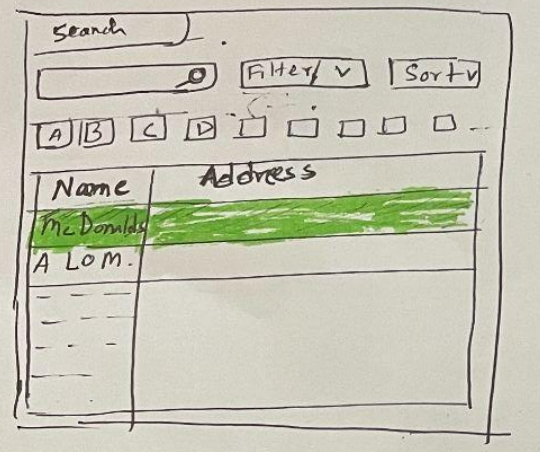
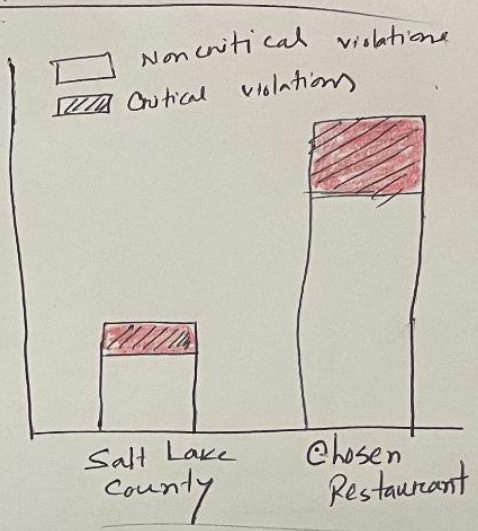


Figure 3: Arleth's Prototype 1 & 2

Salt Lake City Food Inspection Visualizer

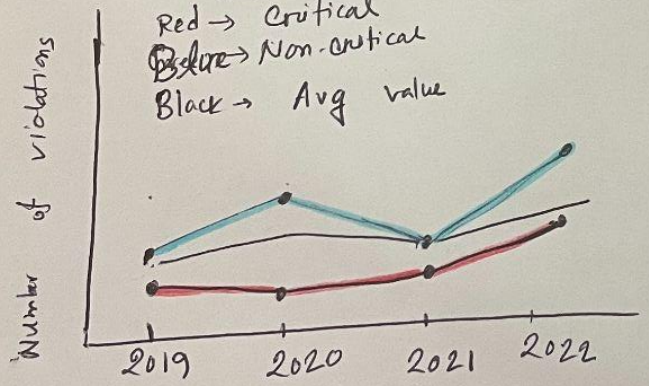
SLCFIV





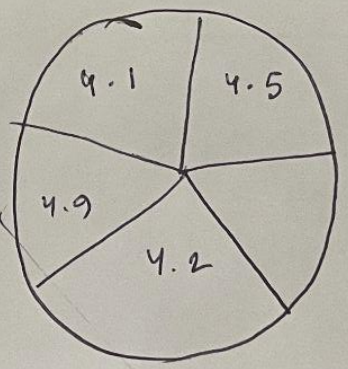
Code violation progression

Red → Critical
Black → Non-critical
Black → Avg value



Code violation types

Chosen Restaurant



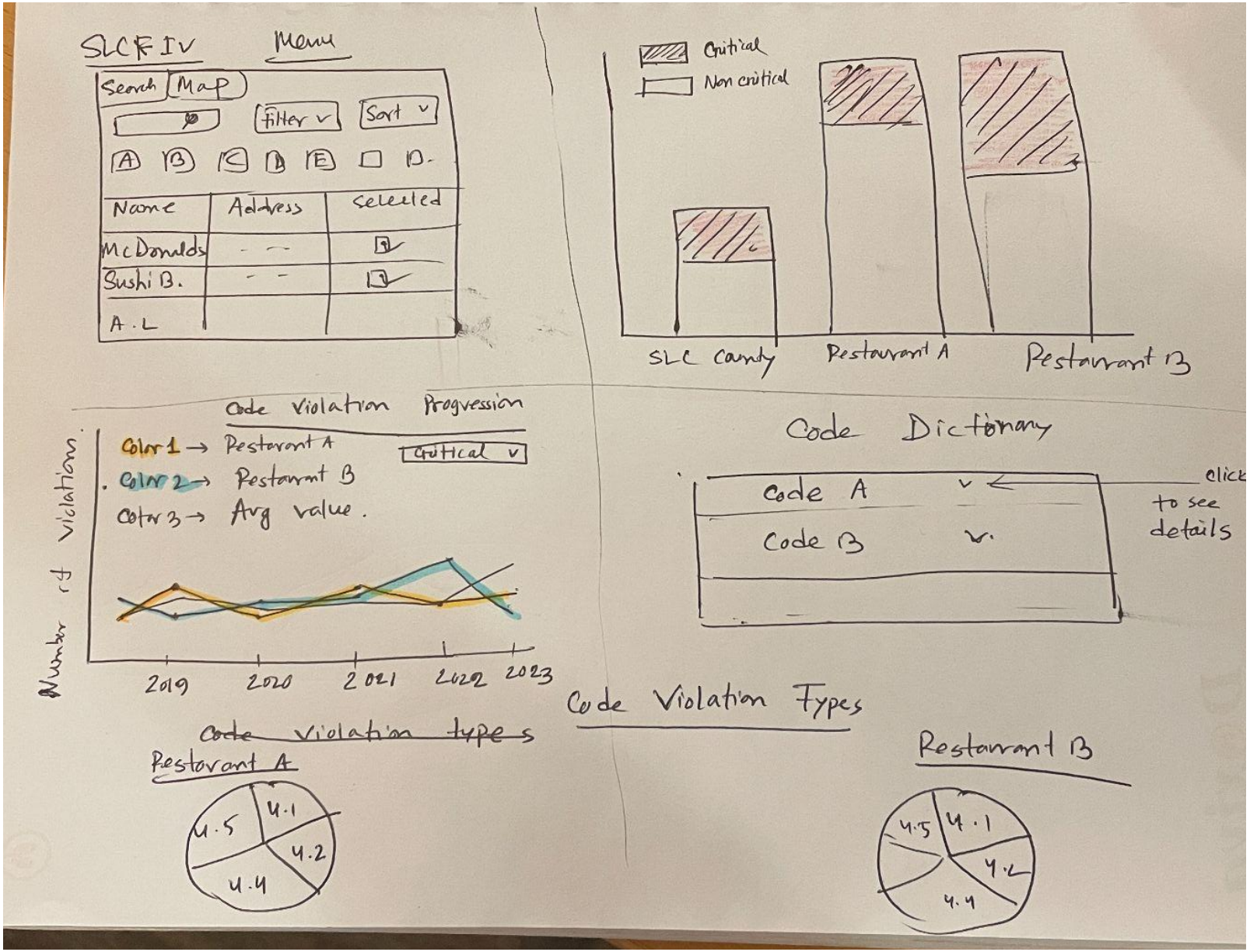


Figure 4: (a) The Must-Have Features, (b) The Optional Features

Data Processing

We first updated the original CSV that we were given to include the town of each restaurant since this was not included. Instead, the CSV had only the address initially.

For most restaurants, this was a simple 2 step process.

1. First, we used an API offered by the Utah government, which allows you to send the address and county of a parcel, which will send you information about that parcel. Unfortunately, for Salt Lake County, the town was always listed as Salt Lake City. However, it also included the latitude and longitude, which we used in step 2.
2. We then used the OpenStreetMap reverse geocoding API. This API allows you to send a latitude and longitude, and it returns information about that point, including the town.

There were many addresses for which the above process did not work. Unfortunately, addresses on numbered roads (e.g., 1300 W) did not include the direction of the road. For example, 1300 E and 1300 W would be listed as 1300. Such cases created ambiguity for addresses, preventing us from looking them up with the above process. To get around this, we first wrote a script that uses geographically informed checks to identify which town an address is in (e.g., north of 2100 S is in Salt Lake City) or what the direction of the road is (e.g., there are no N roads south of 1300, so any address of the form XYZ E 1800 must be on 1800 S). These checks were able to mostly whittle down these ambiguous addresses, although some needed to be manually verified. Some of these restaurants turned out to be food trucks that operated in Salt Lake County, but whose mailing address was outside of Salt Lake County. Because our visualization focuses primarily on Salt Lake County, we thought that including addresses from outside of the county might be confusing, so the address was instead listed as “Food Truck - Out of County” and the town was listed as Salt Lake City.

The data includes health inspection reports of anywhere that serves food. Primarily, this includes restaurants, but it also includes schools, some grocery stores, prisons, retirement facilities, and other facilities. We decided to limit the visualization to establishments that anybody from the general public can attend. Thus, when doing this search, rows containing the following strings in their establishment name were skipped: “ARAMARK,” “CAMP,” “CARE,” “CENTER,” “CHILD,” “CORRECTION,” “DETENTION,” “ELEMENTARY,” “FOOD BANK,” “HEALTH,” “JAIL,” “KIDS,” “LIVING,” “RECOVERY,” “SCHOOL”. This is because businesses with these words in their names are almost certainly not public establishments that anybody can go to.

This CSV was then processed in our Jupyter Notebook using *Pandas* and *NumPy*.

First, we handled data cleaning and derivation in these steps:

1. Add columns defining a violation as a “Critical 1” or “Critical 2” item. Definitions for “Critical 1” and “Critical 2” are on page 4 of the Food Sanitation PDF.

2. Remove the numerical identifiers on restaurants, e.g., Subway #12345. We inspected all the unique names that include the character “#.” We decided that each establishment is identifiable without it via its address. So, we removed the “#” and all characters following it from each name.
3. Next, we deleted all rows that contained the strings used to identify non-public locations in the town search in order to limit our data to only locations that are publicly accessible.
4. During our data cleaning, we realized it would be helpful to note when a restaurant received a violation under 7.4.4 Emergency Enforcement in its own column.
5. We removed rows with the following violations for the corresponding reasons:
    a. "R392-510UICAA” violation code as this is regarding the Utah Clean Air Act, which is not mentioned in the Food Health Regulation PDF.
    b. “R392-100: 5-202.11*” violation code, which is not mentioned in the Food Health Regulation PDF.
    c. All violation codes begin with “0”, as these are not rules broken by the establishment but rather notifications for events such as “Cleaning Education Provided” or shadowing.
6. We noticed that violations under 7.4.2 are Variances, which are not rule violations but rather allowances for modifications or waivers to establishments of specific rules. We decided to label such violations accordingly with a “Variance” column.
7. After all this cleaning, we then removed asterisks from the violation codes.
8. Finally, for establishment names and street names, we made all two and three letter words lowercase, as well as the word “a”. All other words had their first letter capitalized and all other words lowercase. This is because the data was given to us in all capital letters.

After this processing, we converted the data we pre-processed as *CSV* into the *JSON* format. It helps to read data easily for creating the menu, tables of the restaurants, and all the necessary charts.

Menu Creation

To get the data ready for the menu, we created a javascript file that iterates through all of the data and assigns it to a relevant class hierarchy. This hierarchy has three classes and can be seen demonstrated in figure 5. The Restaurant class represents a single restaurant. Each restaurant has a collection of instances of the Inspection class. This Inspection class represents a single time a health inspector inspected that restaurant. Each Inspection contains a collection of instances of the Violation class. The Violation class represents a single violation found during a health inspection. All of the restaurants are stored as a list of javascript objects. We wrote some helper functions that allow us to sort this list by various criteria, as well as to filter it down according to search terms. One can sort and filter by name, address, and town. Additionally, one can sort by various metrics pertaining to violations.

Figure 5: The Three Class Hierarchy

We represented the table itself with a JQuery Datatable. Each row holds the name, address, and town of each restaurant, which are the criteria that we decided it would be appropriate to sort and filter by. Above this, we created a small interface, shown in figure 6, that allows the user to sort and filter as needed.

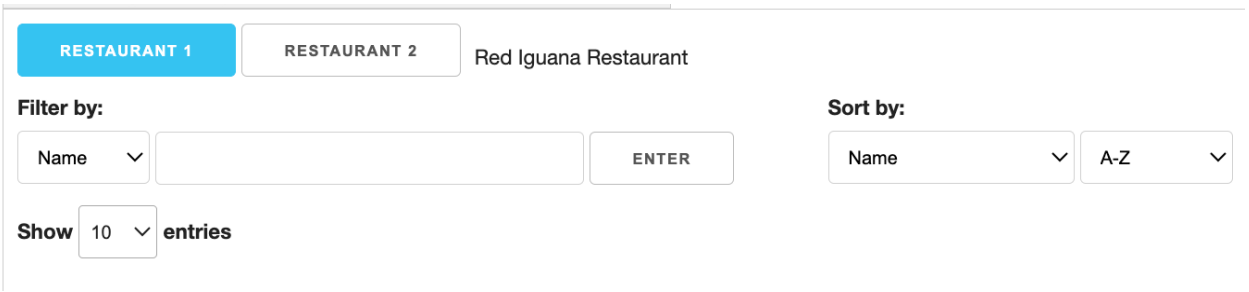


Figure 6: The Selection Interface

We also added the functionality to select a second restaurant so that the user can compare the results of the two restaurants. We decided to limit the selection to two, as otherwise we fear that some of the charts, particularly the bubble chart area, would get cluttered.

Initially, the table was a scrolling list, although the menu could be quite large depending on the search terms. It would also be annoying if the list was so short that one would scroll quickly to the end and thus scroll the entire webpage. We decided that it would be best to make the menu display a series of pages, rather than one large list. By default, the table holds 10 records on each page, and the user can also choose 10/20/50/100 records per page. Pagination has some buttons to go to the NEXT of the PREVIOUS page and some page numbers as buttons.

Line Graph Creation

To make the line graph, the first thing that needed to be accomplished was to calculate the average number of violations per inspection for Utah County for each year. This was not difficult

to accomplish. After this, the line chart was made using standard procedures in D3. It was not initially clear how to plot the Salt Lake County average each year as a single vertex on a line chart. This is because the rightmost endpoint of the x-axis of the line chart is the first day of the first year outside of the date range being represented. To that end, it did not make sense to put a data point there. We ultimately decided to place each data point at the midpoint of each calendar year, which it represented.

Initially, the legend for the chart was at the bottom. However, because the line chart is at the bottom of the display, we worried that the user may not scroll down far enough to see it. Thus, we placed the legend at the top-left. We also have put a dropdown to choose what kind of violation the user intends to display the line charts. The dropdown has three options: “All Violations”, “Non-Critical Violations”, and “Critical Violations”.

Later, we decided to add a functionality where when one hovers over the points on the line chart of the restaurants, it displays the date and the violation count for that date in the tooltip. For the Salt Lake County Average, the tooltip shows the year and average violation count for that year.

Finally, we decided to make the Salt Lake County Average line dashed in order to make it stand out from the actual restaurants.

Figure 7 shows the final result.

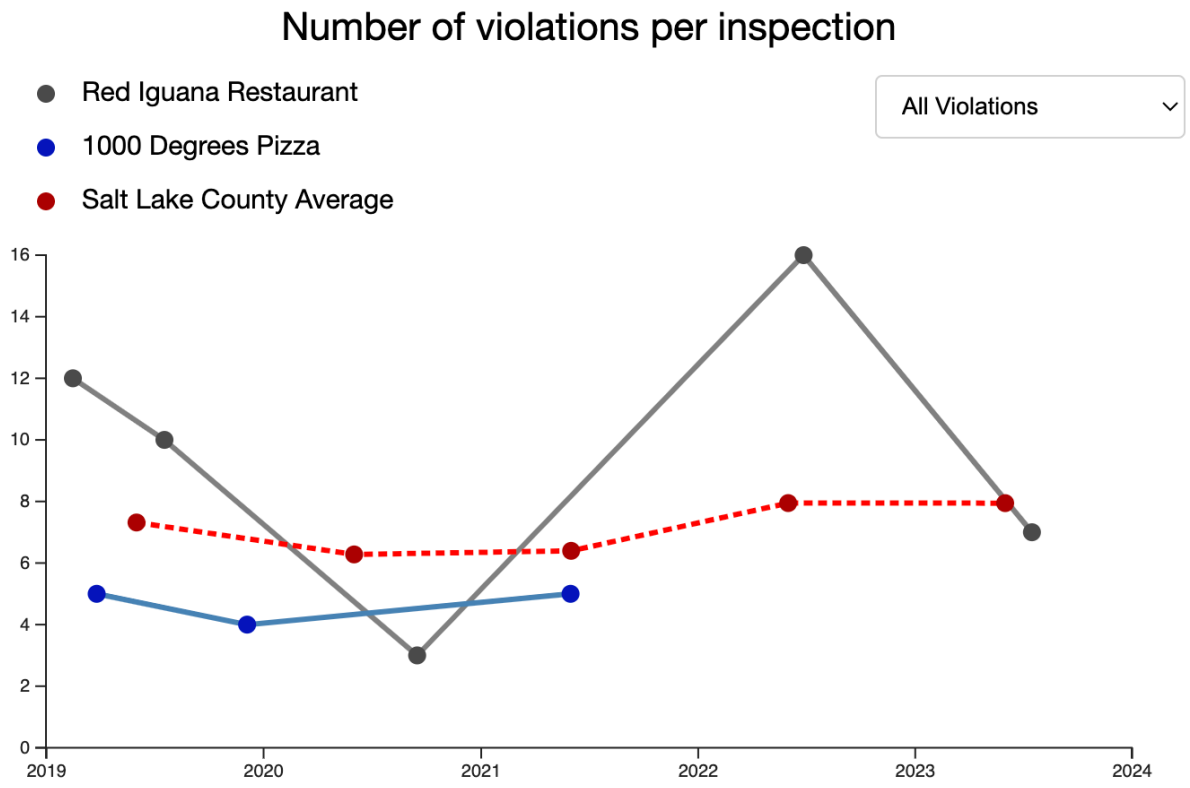


Figure 7: Final iteration of the line chart

Bar Chart Creation

The bar chart initially involved creating two stacked bars. Each bar was a stack of the average number of non-critical violations per health inspection, and the average number of critical violations per health inspection. One bar represented the average for Salt Lake County, and the other for the restaurant selected by the user. The Salt Lake County averages are precomputed on the JSON data. The total violations for a selected restaurant are obtained from our JSON data structures as well. We then later added the functionality to view a third bar corresponding to the second restaurant that the user selected if they eventually chose to do so.

Initially, we had the bar labels displayed horizontally below the bars. However, some restaurant names are very long, and the names started to intersect one another. We tried various angles to display the names and eventually decided to display the restaurant names at a 15-degree angle.

Also, we decided to add a functionality where when one hovers over a bar, it displays the bar value in the tooltip.

Figure 8 shows the final iteration of the bar chart

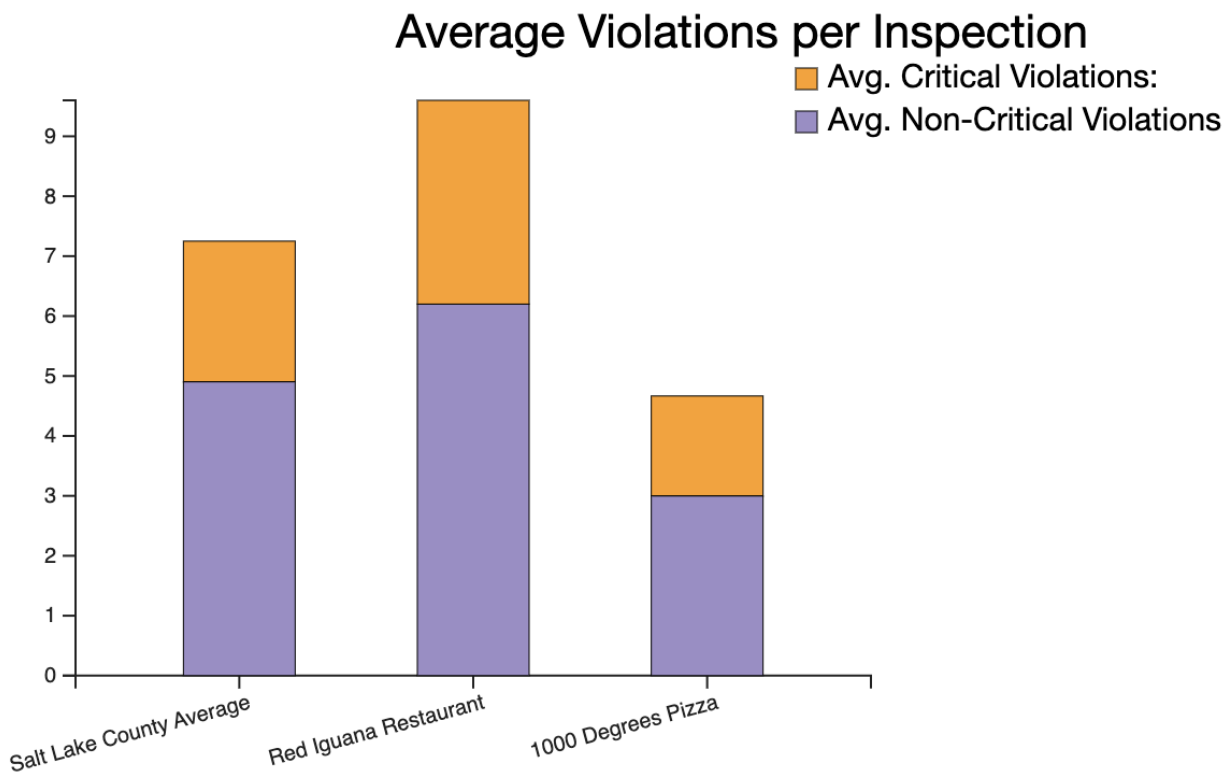


Figure 8: Final iteration of the bar chart

Visualizing Code Violations

Initially, we had planned to visualize the code violations for the chosen/selected restaurant in a pie chart. While proceeding with the implementation, we printed the violation codes that occurred for random restaurants and found that, on average, they are usually 6/7 for each restaurant. Some codes occur twice or thrice, but mostly, they occur once within the total number of inspections they went through. We decided that a pie chart is, therefore, not the best way to visualize the selection. While specific codes occurred only once most of the time, each code belonged to a code family, and there were frequently several different codes in a single-family observed in a single restaurant. Thus, there is a hierarchical nature to violations that we want to preserve. In order to show this sort of hierarchical nature, we considered two different possibilities. First, the Treemap / D3 | Observable. Second, Bubble chart / D3 | Observable.

After deeper consideration, we chose to use the bubble chart. The bubble chart showed one bubble for each code that a restaurant had violated, where the area of the bubble was proportional to the number of times that the violation occurred. Each bubble is colored by what violation family it is in. There are legends having these color groups and descriptions of each grouping (violation code family). Initially, the bubble chart showed all of the different codes in no

particular arrangement. We later decided that it would be better if the violation family grouped the bubbles, so we adjusted it in that way. Also, each bubble initially listed the exact violation code that it represented in the bubble. In order to fit in these codes, sometimes the text needed to get very small. Additionally, the violation codes themselves are not necessarily meaningful to the viewer. Thus, we decided to drop the violation code from the individual bubbles and instead add it to the tooltip text. We also initially had each bubble display how many times the violation corresponding to that bubble occurred. While we decided to keep this on each bubble, there were times when the font needed to be extremely small in order to fit it on the bubble. In these cases, we decided to drop the number from those bubbles, and again added this information to the tooltip. The tooltip text for each bubble now contains the exact violation code and its corresponding description. To make it clearer what bubble the user was hovering over, we added a border to the specific bubble that the user was interacting with.

We considered adding a second bubble chart in a new row for the second restaurant that was selected; however, we decided that adding a new row for a second restaurant that was selected would disrupt the flow of the user only some of the time. Thus, we decided to split the area in half and show two smaller bubble charts in the same area. This was made further possible by our decision to drop text on the bubbles when they get too small. To make the bubble charts distinguishable to the user, we have added a vertical dividend and added the names of the restaurants at the bottom of each bubble chart.

Figure 9 shows the final iteration of the bubble chart.

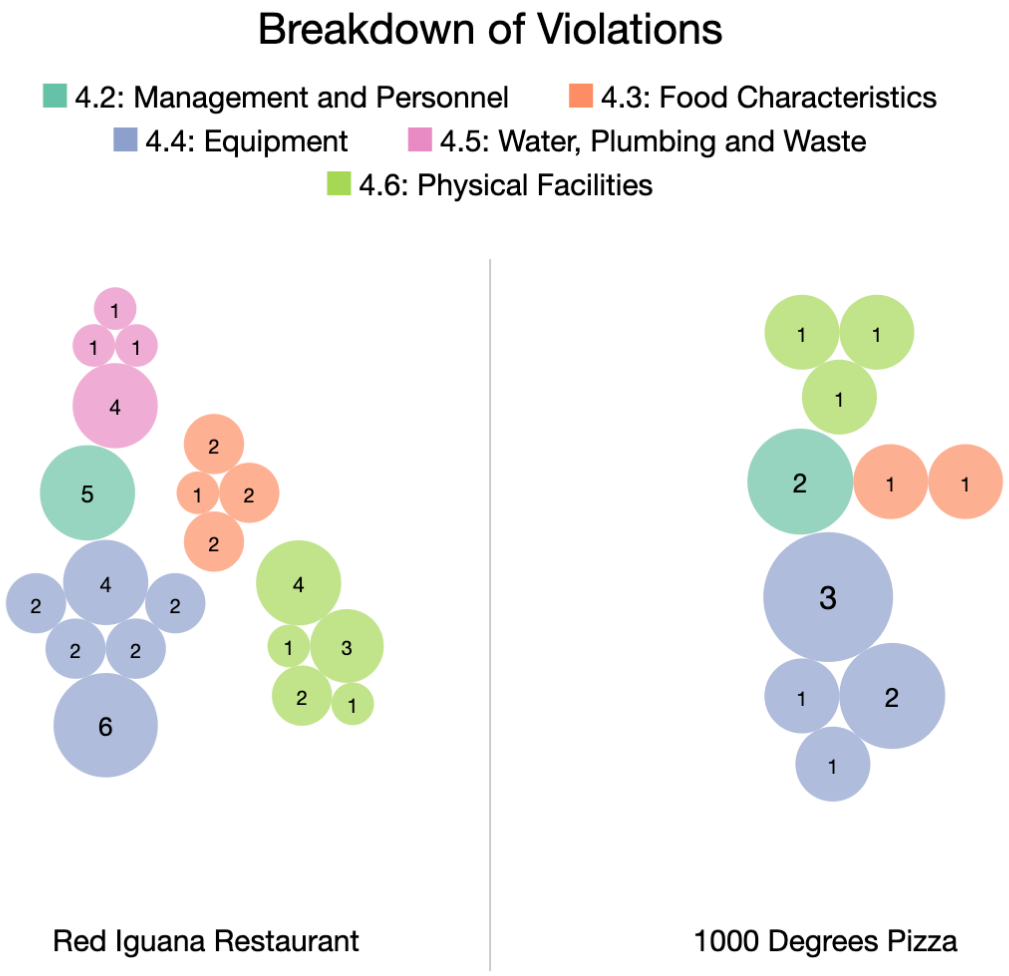


Figure 9: Final iteration of the bubble chart

Map Creation

We decided to add a map to our menu system. We first considered adding the map to sit on display as the visualization did, but we realized this would take up too much space on the screen. Therefore, we decided to return to the idea of adding a tab to our menu where one could choose the search table or the map.

We found a useful library called leaflet that provided lots of map functionality already. Using their quickstart guide we quickly got a map displayed on the "map" tab of our menu.

We took a long time trying to get the coordinates for all the restaurants. We used the Utah government's Geocoding API Methods to do the geocoding we sought. Firstly, we tried doing so by incorporating it into our process_data.js pipeline, but realized it is very difficult to do with

JavaScript, especially trying to handle the HTTP requests. So, we switched to using Python and then obtained the coordinates.

Unfortunately, almost half of the restaurants could not be assigned coordinates, due to issues such as missing the North/South/East/West designation. After trying to identify more of the addresses coordinates with the OpenStreetMap geocoding we decided enough were identified to leave the map in the menu. We strongly considered removing the map entirely otherwise.

Figure 10 shows the final iteration of the map:

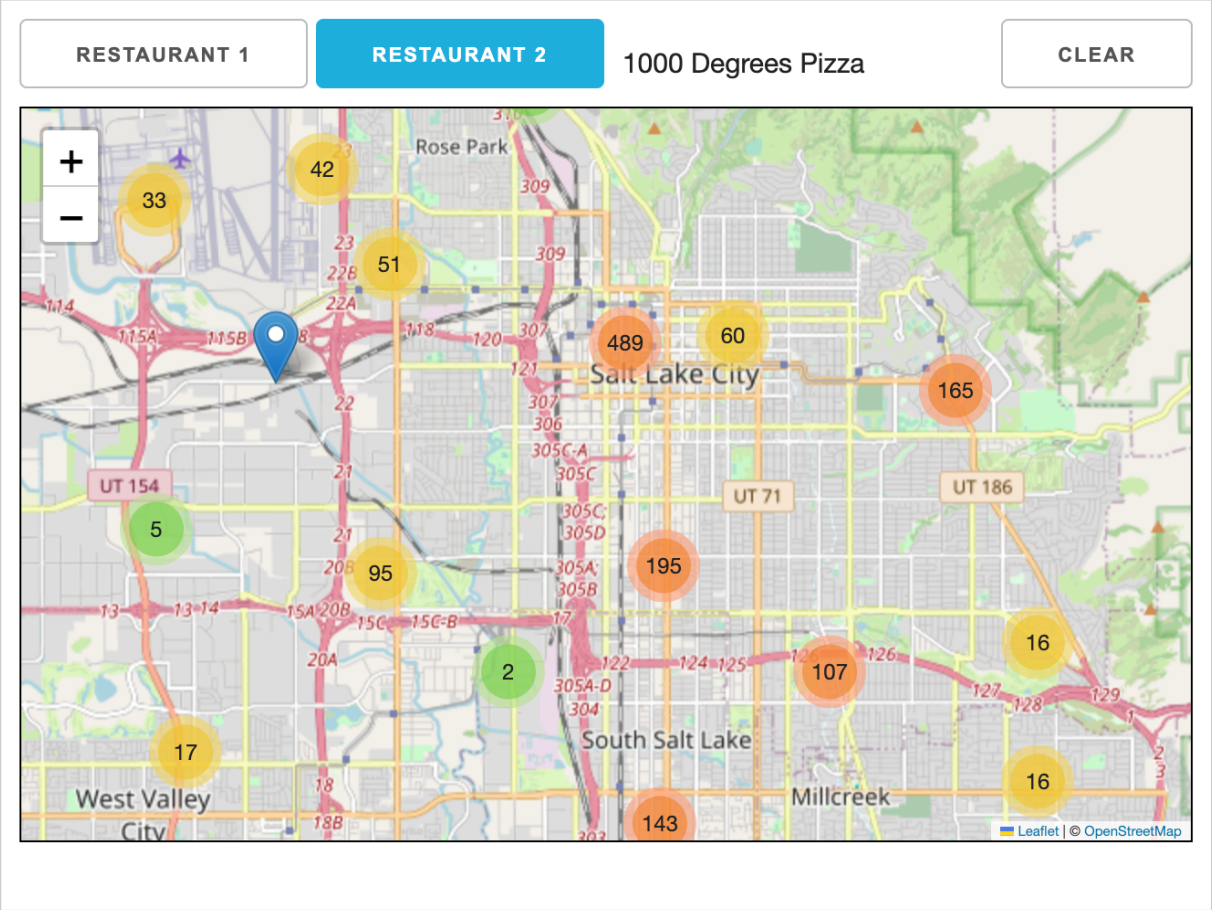


Figure 10: Final iteration of the menu map