CS179G FINAL REPORT

Nancy Li Vincent Chang Adam Fan

Phase 1

Requirements:

- Gather data

Our project requires housing data for many regions along with salary data for many occupations in each region. We plan to correlate occupation salary data with housing data so people are able to find out where they can afford to buy a house. This will be done by finding the ratio between salary and house price.

Design:

We are going to use data from www.bls.gov and www.zillow.com. API, however they set request limits with the API. Thus, we decided to use a web crawler instead. The data is already in a useable format on the web sites, however, there is a lot of data to download. Most of the data is stored in directories so we will use a web crawler to automate the downloads. The BLS website had an

Implementation:

The web crawler is done in Java using the Jsoup library. It takes a given page and extracts all the links in the page. It downloads every file from every link that is retrieved. It writes the files to a specific directory with the original file names.

```
public class crawler {
    public void crawl(String URL){
        try {
            Document doc = Jsoup.connect(URL).get();
            Elements link = doc.select("a[href]");
            PrintWriter out;
            String filename = "";

            for( Element page : link){
                String absHref = (page.attr("abs:href"));// == "/"
                int index = absHref.lastIndexOf('/');

                if(index == absHref.length() - 1){
                        continue;
                }

                filename = absHref.substring(index + 1);
                filename = "/Users/nancyli/Desktop/179_data/".concat(filename);
                System.out.println(filename);
                out = new PrintWriter(filename);

                URL url = new URL(absHref);
                URLConnection conn = url.openConnection();
                BufferedReader in = new BufferedReader(
                        new InputStreamReader(conn.getInputStream()));
                String inputLine = "";
                while ((inputLine = in.readLine()) != null){
                    out.println(inputLine);
                }
                in.close();
                out.close();
            }
```

Overall, we pulled 12 gigabytes of data. Below is a sample of some of the raw data we retrieved

```
series_id       seasonal        areatype_code   industry_code   occupation_code                         datatype_code
state_code      area_code       sector_code     series_title    footnote_codes begin_year       begin_period
end_year        end_period
OEUM001018000000000000001       U       M       000000  000000  01      48      0010180 00--01  Employment for All
Occupations in All Industries in Abilene, TX 1                  2015    A01     2015    A01
OEUM001018000000000000002       U       M       000000  000000  02      48      0010180 00--01  Employment percent
relative standard error for All Occupations in All Industries in Abilene, TX        3               2015    A01
2015    A01
OEUM001018000000000000003       U       M       000000  000000  03      48      0010180 00--01  Hourly mean wage for
All Occupations in All Industries in Abilene, TX                        2015    A01     2015    A01
OEUM001018000000000000004       U       M       000000  000000  04      48      0010180 00--01  Annual mean wage for
All Occupations in All Industries in Abilene, TX      2                 2015    A01     2015    A01
OEUM001018000000000000005       U       M       000000  000000  05      48      0010180 00--01  Wage percent relative
standard error for All Occupations in All Industries in Abilene, TX 3                   2015    A01     2015    A01
OEUM001018000000000000006       U       M       000000  000000  06      48      0010180 00--01  Hourly 10th percentile
wage for All Occupations in All Industries in Abilene, TX                       2015    A01     2015    A01
OEUM001018000000000000007       U       M       000000  000000  07      48      0010180 00--01  Hourly 25th percentile
wage for All Occupations in All Industries in Abilene, TX                       2015    A01     2015    A01
OEUM001018000000000000008       U       M       000000  000000  08      48      0010180 00--01  Hourly median wage for
All Occupations in All Industries in Abilene, TX                        2015    A01     2015    A01
OEUM001018000000000000009       U       M       000000  000000  09      48      0010180 00--01  Hourly 75th percentile
wage for All Occupations in All Industries in Abilene, TX                       2015    A01     2015    A01
OEUM001018000000000000010       U       M       000000  000000  10      48      0010180 00--01  Hourly 90th percentile
wage for All Occupations in All Industries in Abilene, TX                       2015    A01     2015    A01
OEUM001018000000000000011       U       M       000000  000000  11      48      0010180 00--01  Annual 10th percentile
wage for All Occupations in All Industries in Abilene, TX    2                  2015    A01     2015    A01
OEUM001018000000000000012       U       M       000000  000000  12      48      0010180 00--01  Annual 25th percentile
```

Contribution:

Nancy Li: Wrote the web crawler
Vincent Chang: Debugged on web crawler, downloaded data
Adam Fan: Put data into Spark

Phase 2

Requirements:

- Sorting through wage vs occupation data and loading onto Cassandra
- Find the moving average of house prices time series per 12 months and loading onto Cassandra

The wage vs occupation data was found on the bls.gov website. In this file, there are different measurements of wage per occupation in the form of time series. The measurements are taken annually and are to be used in conjunction with house prices annual time series. We have to load relevant files onto Cassandra and partition the files as they are relatively large. We also have to reduce the file size. We plan on doing that by taking out occupations that are not popular and/or are too specific. There are many overlapping classes of occupations so we need to take that out as well.
We need to calculate the moving average of house prices over 12 months. We downloaded data from Zillow.com and they processed the data to be a monthly time series. The values represent the median house prices for the month. After processing the data, we need to store it onto Cassandra.

Design:

For identifying useless occupation classes, we manually looked over the mapping file that listed the occupations that are part of the file.
Since the data is a time series, we cannot simply take the average of all the values over the years and call that the mean house price. This is because the prices are dependent on time and the value changes along with it. The design for calculating the moving average is to take the annual average. This is done by averaging 12 months worth of data and setting the obtained value to be the time series value for the month in the middle.

Implementation:

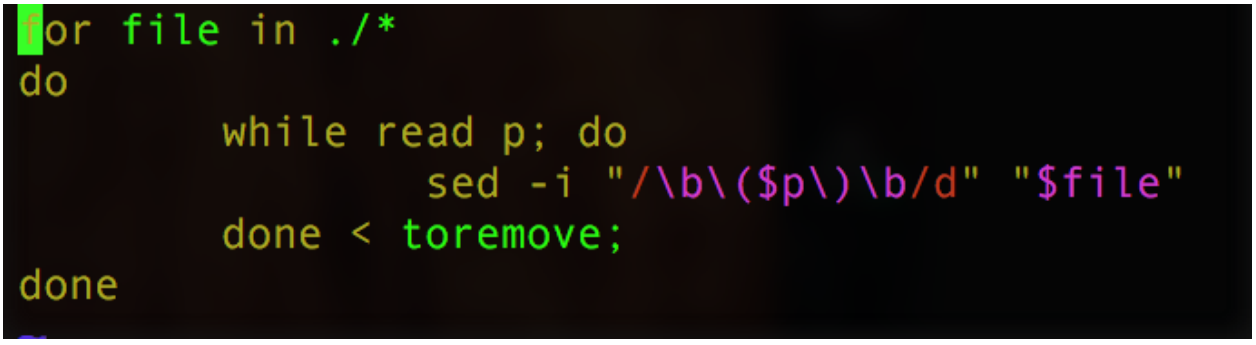
Occupation Data

For removing the classes and cleaning the data, we used bash scripts to clean the data. We looked through the mapping file with all the occupation classes and they have a unique occupation code per occupation.

occupation_code	occupation_name	display_level	selectable	sort_sequence			
000000	All Occupations	0	T	0			
110000	Management Occupations	0	T	1			
111000	Top Executives	1	T	2			
111011	Chief Executives	3	T	3			
111021	General and Operations Managers	3	T	4			
111031	Legislators	3	T	5			
112000	Advertising, Marketing, Promotions, Public Relations, and Sales Managers	1	T	6			
112011	Advertising and Promotions Managers	3	T	7			
112020	Marketing and Sales Managers	2	T	8			
112021	Marketing Managers	3	T	9			
112022	Sales Managers	3	T	10			
112031	Public Relations and Fundraising Managers	3	T	11			
113000	Operations Specialties Managers	1	T	12			
113011	Administrative Services Managers	3	T	13			
113021	Computer and Information Systems Managers	3	T	14			
113031	Financial Managers	3	T	15			
113051	Industrial Production Managers	3	T	16			
113061	Purchasing Managers	3	T	17			
113071	Transportation, Storage, and Distribution Managers	3	T	18			
113111	Compensation and Benefits Managers	3	T	19			
113121	Human Resources Managers	3	T	20			
113131	Training and Development Managers	3	T	21			
119000	Other Management Occupations	1	T	22			
119013	Farmers, Ranchers, and Other Agricultural Managers	3	T	23			
119021	Construction Managers	3	T	24			
119030	Education Administrators	2	T	25			
119031	Education Administrators, Preschool and Childcare Center/Program	3	T	26			
119032	Education Administrators, Elementary and Secondary School	3	T	27			
119033	Education Administrators, Postsecondary	3	T	28			
119039	Education Administrators, All Other	3	T	29			
119041	Architectural and Engineering Managers	3	T	30			
119051	Food Service Managers	3	T	31			
119061	Funeral Service Managers	3	T	32			
119071	Gaming Managers	3	T	33			
119081	Lodging Managers	3	T	34			
119111	Medical and Health Services Managers	3	T	35			
119121	Natural Sciences Managers	3	T	36			
119131	Postmasters and Mail Superintendents	3	T	37			
119141	Property, Real Estate, and Community Association Managers	3	T	38			
119151	Social and Community Service Managers	3	T	39			
119161	Emergency Management Directors	3	T	40			

(An example of the occupation mapping file)

We used a key-value approach to filter the selected values from the main file and deleted the corresponding rows containing the specified occupation code. We wrote two main scripts to modify the main file and used the split command to chunk the file into smaller pieces. The first script was used to insert the schema in the beginning of each file in the directory and it was done using sed to match the beginning of every file and insert the schema.



(first script)

The second script we wrote to trim out extraneous data from the series file. To do this, we read through the occupation list file and inserted the job codes for jobs that were either too general or too obscure into the file.

```
for file in ./*
do
        echo 'series_id seasonal        areatype_code   industry_code   occ
upation_code datatype_code   state_code      area_code       sector_code
  series_title    footnote_codes  begin_year      begin_period    end_year
      end_period' | cat - "$file" > temp && mv temp "$file"
done
```

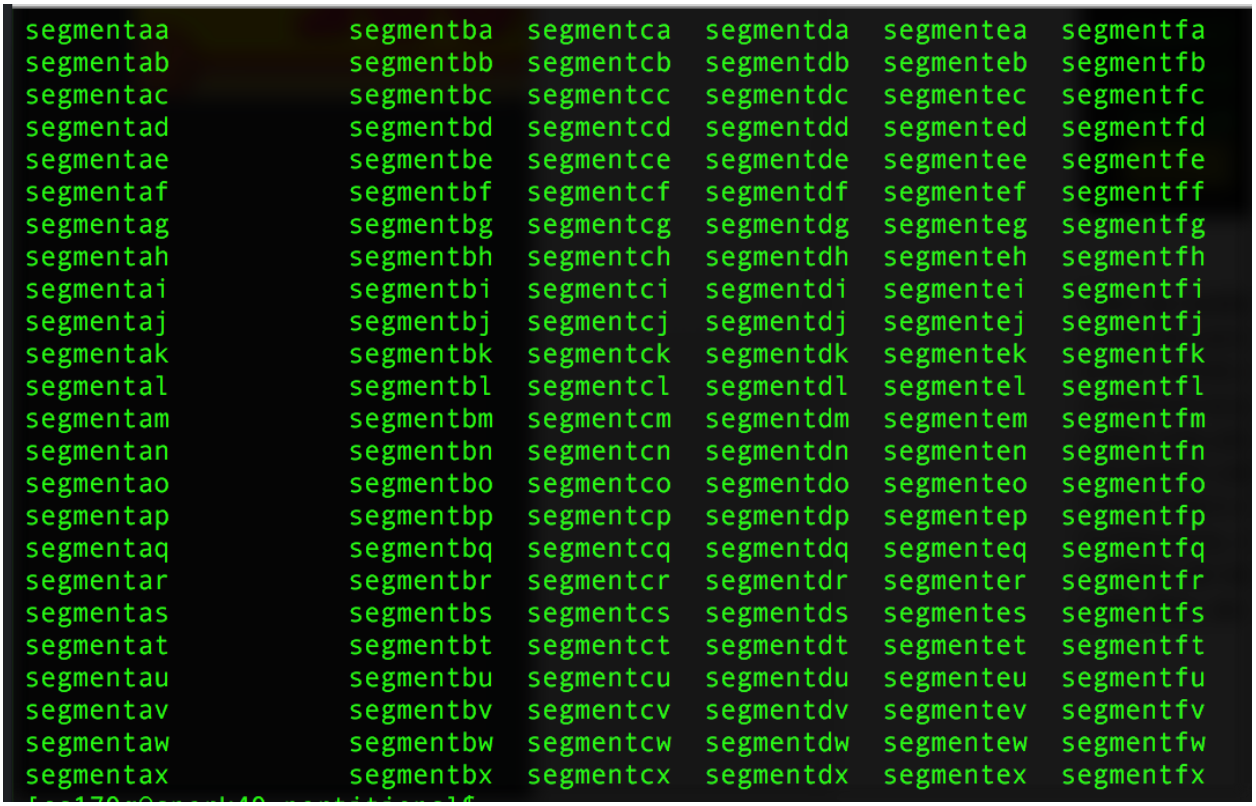
(second script)

We also filtered the data by data types. The data that we got had different measurements of wage. We did not need all of them and we filtered them out by using the filter function on the dataframe. Since the domain of the useless data values were much smaller, it was more feasible to do in pyspark than on bash.

```
test1 = test.filter("datatype_code != '01'")
test2 = test1.filter("datatype_code != '04'")
test3 = test2.filter("datatype_code != '12'")
test4 = test3.filter("datatype_code != '13'")
test5 = test4.filter("datatype_code != '14'")
```

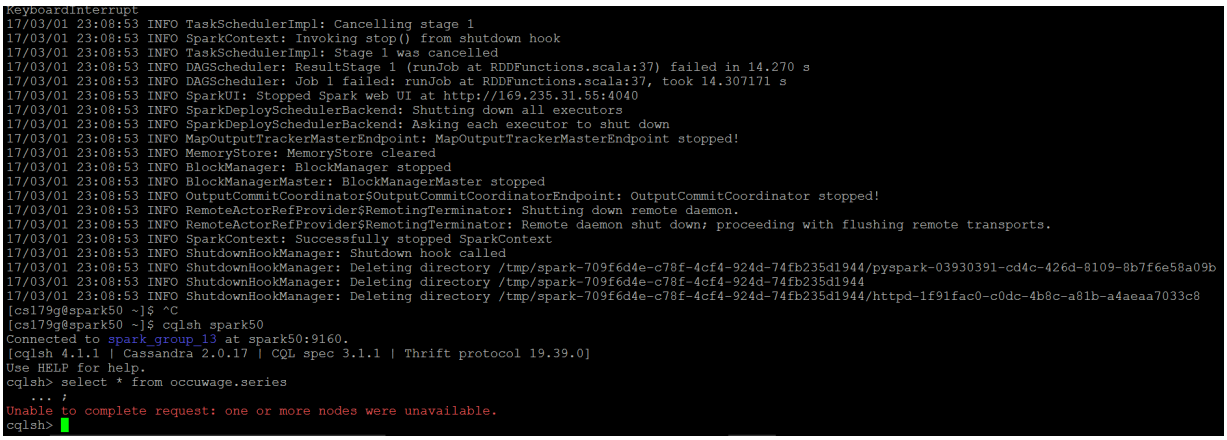
(use of filter function)

Then we loaded the codes in that file into a list and compared every line in the series file to the list and deleted any line that matched with any entry from our list. We ran into errors namely with removing the unneeded occupations. Some of the series ids contained occupation codes we were trying to delete as a substring, and in order to match only what we wanted, we had to check for tabs both before and after the occupation code. Inserting the data into Cassandra proved very difficult as well, as we had to read in the series file chunk by chunk and write them into Cassandra separately. We did not end up using this because when we further filtered the file, we were able to save to Cassandra.



(partitioned files)

Furthermore, two of the Cassandra nodes crashed, leading to the error depicted below in which we would be disconnected from the Cassandra server quickly after running the code. Also, we could not run any CQL queries on the dataset it wrote to.



(Cassandra errors)

Housing Data

For processing the housing data, we had to modify the original file on excel first. We had to replace all the empty cells with 0's or else the value would change to empty string

when we read it into spark. This made calculations impossible as spark will not do numerical calculations on strings. We also could not substitute the empty string values because Spark is unable to replace whitespaces with any integers. At first, we followed the mapping function that the TA sent out as a tutorial. Soon after, we figured out that it would cause a lot of errors with the way we are implementing our mean. I used a for loop to iterate through the columns and rows individually in order to add values and then divided it by the number of months there were in the data. We ran into errors namely with putting the results onto Cassandra. One of our main challenges was trying to iterate through rows and adding them without getting confused with the numbers. Since there was no function to help us iterate through rows, we had to try our best in making an iterator that could help. Our error was that our column names had capitalization on some of the letters, which Cassandra is unable to read. Cassandra only reads in lowercase letters. This resulted in an error about having a null value in the column. Turns out, Cassandra was looking at the wrong columns the whole time because of this mistake. We spent two days trying to debug this problem before going to the TA to get help.

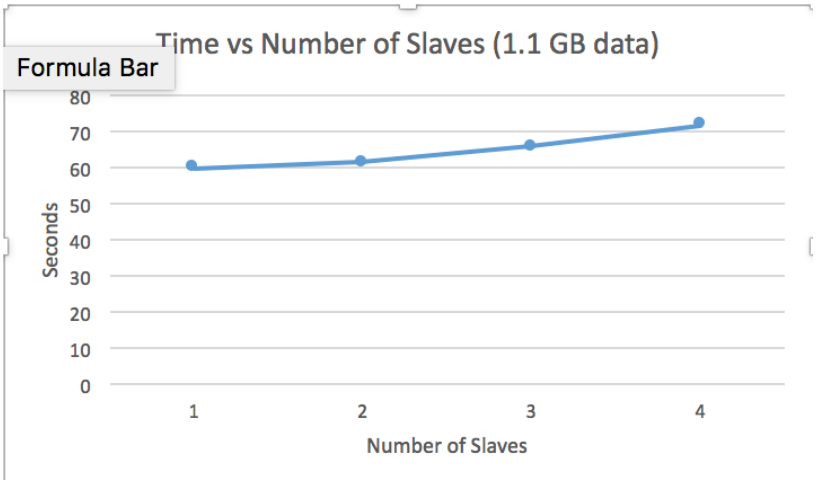
```
#  new_df = test.withColumn("Mean1",)(col("Mean")))
  rowMean  = (sum(col(x) for x in test.columns[7:]) / 249).alias("mean1")
  new_df = test.withColumn('Mean',rowMean)
  new_df.show(3)
  test_filter1 = new_df.filter(new_df.RegionID == '6181')


  newest_temp = test_filter1.map(lambda row: { 'regionid': row.RegionID,
                                               'mean': row.Mean,
                                               'regionname': row.RegionName,
                                               'state': row.State,
                                               'metro': row.Metro,
                                               'countyname': row.CountyName,
                                               'sizerank': row.SizeRank})
```

Sample of one of our mean functions

Node Analysis

When running spark code, we varied the number of worker nodes. We found that the time that it took to run increased with the number of nodes. This is due to the size of the data. The data was not big enough to justify the overhead of portioning the work between all the nodes. A decreasing trend may be observed if the data was larger since the overhead would be proportionally smaller than the amount of processing time needed for the larger dataset.



Contribution:

Vincent Chang: Worked on bash scripts to clean the data and helped with filtering out unnecessary data. Helped get the files onto Cassandra.
Adam Fan: Worked on the pyspark script to get the moving average of housing data. Moved Data onto Cassandra.
Nancy Li: Helped with implementation and debugging of both areas and resolving Cassandra related problems.

Phase 3

Requirements:

- Design a website
- Retrieve data from Cassandra and load it onto Google Fusion Maps
- Map cities by color based on the ratio between average salary and average housing price for a given industry

We needed to find a way to get the data from the Cassandra cluster and passing it into Google Fusion Maps to display our data. We also needed to design a website that would retrieve the correct map when the user requests it. Lastly, we need to give the user a way of quickly finding good areas to search for a job in.

Design:

To make the large amount of data easily viewable we decide to make color coded pins on a map with red representing areas with a poor salary to housing price ratio (below 0.3), beige having a ratio between 0.3 and 0.7, and green being anything above 0.7. To get the data, we needed to fetch the region names and the average occupation salary for all the industries in our data.

On our website, we let the users chose their desired industry and after clicking on it, a map of the United States is displayed with the color coded pins on the map. Then, the user can click on any of the pins to view the region name and salary to housing price ratio for it.

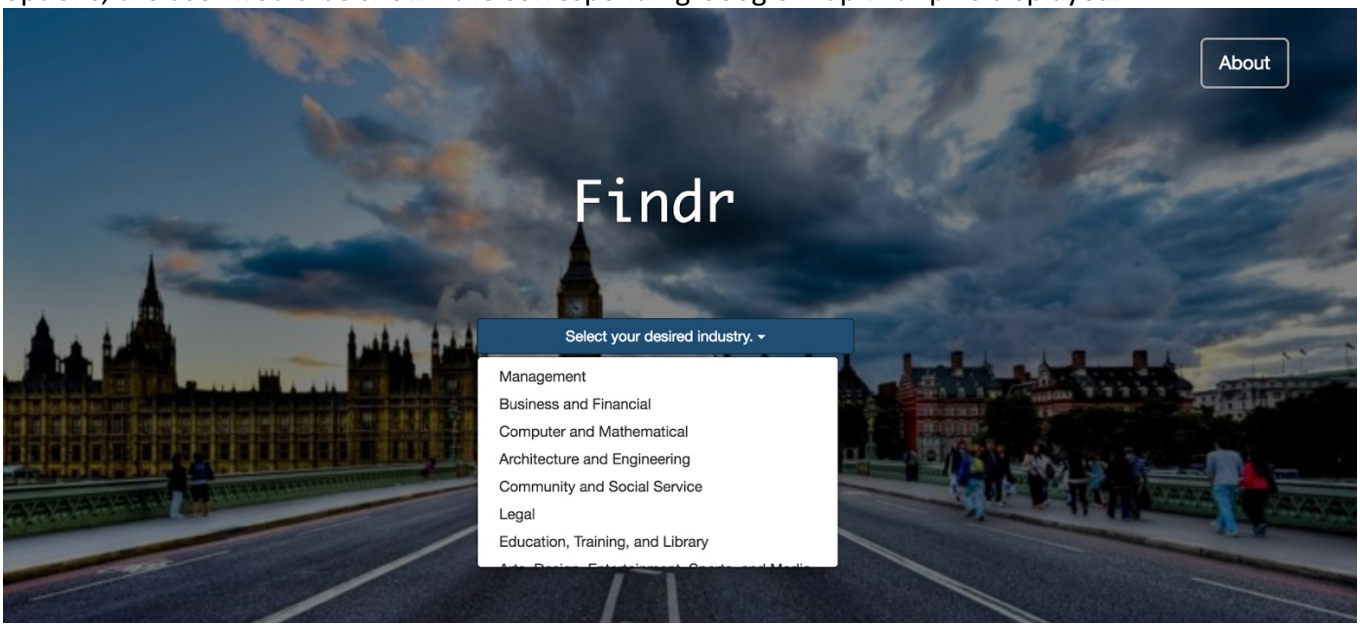
Implementation:

We tunneled the Cassandra connection to our local machine by using the method provided by the TAs. We wrote Spark code in order to retrieve all data for different industries, while filtering out data for specific occupations and data with salary percentiles. For example, we did not need the top ten percentile of earners for the mining industry since there is no actual salary values in those rows. Furthermore, the data we retrieved from the Bureau of Labor Statistics had its data split into multiple files, with one main mapping file. Instead of the industry name being in the main file, there was a code representing it. The code then is mapped to the corresponding name in a different file. In the end the code outputs a dictionary containing a tuple of region name and industry name as the key and average salary as the salary.

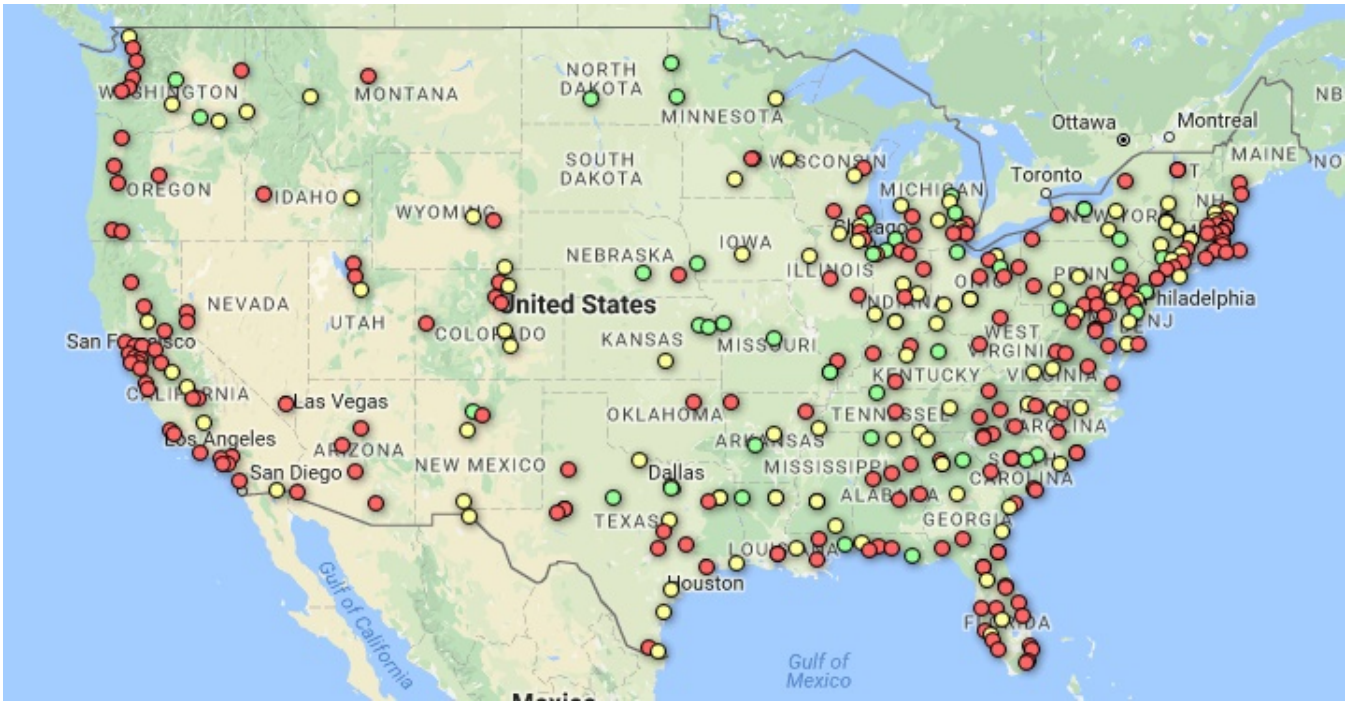
With the list of tuples, we passed into Google Fusion Maps, along with the salary to housing price ratio. We used Google's algorithm with Geocoding to map all of our region names onto the Google Maps API which would allow us to place our pins on the matching regions. We could not create a heat map because Google Maps API does not allow us to export the Geocoded heat map for display on our website.

The website was written using the Bootstrap framework. We used a modal body that is displayed by clicking on a button for our about page. For accepting the user's industry selection,

we created a button that triggered the display of the drop down menu. Upon clicking any of the options, the user would be shown the corresponding Google Map with pins displayed.



Home screen



Example of a map

The part we struggled the most with this part was writing the code to retrieve and merge all of the data from Cassandra into a readable format. We had to read in data from multiple tables, store them into a dictionary proved difficult as we were having issues where the key or values were null or values were being stored into the incorrect key.

Contribution:

Vincent Chang: Wrote the website front end and helped with writing the code for retrieving the data from Cassandra.
Adam Fan: Passed data into Google API which created the maps with color coded pins.
Nancy Li: Wrote the code for retrieving the data from Cassandra and parsing it into a list and helped with implementation of front end.

Comments from Previous Parts

No revisions requested.