Charley Ferrari

Project 4

Source Data: <https://github.com/fivethirtyeight/data/tree/master/college-majors>

Source Blog Post: <http://fivethirtyeight.com/features/the-economic-guide-to-picking-a-college-major/>

I’ve chosen to use the data from fivethirtyeight’s post: The economic guide to picking a college major. To conform to the scope of this project, I used the aggregated data already created by the author rather than the source data from the census. This aggregation, combined with the changing schema of the underlying data depending on time or for which types of students data was being pulled complicated my decisions in which database to use, and in order to provide a full analysis of the best data paradigm the source data must be looked at. Because of these complications, I have decided that MongoDB would be the best way to represent this data.

**Loading and Storing the data in R**

The first step of this analysis involved loading and storing the data into RStudio. R is used as the first step for both PostgreSQL and MongoDB, and the R file is listed as Appendix A.

The raw CSV files are brought into R directly from the source URLs. My first step was to clean up the data a bit. First, I made sure all the majors were unique. The majors.list had an extra entry of no major which I removed. With this removal, I was able to make the FOD1P column an integer. Lastly, I renamed a few column names so they will match across multiple datasets.

If I were to keep the data in RData format, I would delete the majors.list table, as an extra relationship table is not necessary. Any manipulation that needs to be done would be able to be done on these three tables at this point.

**Loading the Data into PostgreSQL**

My next step was to load the data into PostgreSQL. Taking the data from the end of part 1, I ended up removing the columns Major and Major\_category from the all.ages, grad.students, and recent.grads tables. Major\_code is the primary key for the majors.list table, and the foreign key in the other three. The schema is outlined in the excel file Appendix C. The table relationships are shown in a diagram, and the columns of each table are outlined. Major\_code serves as the primary key for all tables, and there is a one to one relationship between all the data.

Once the tables are manipulated, the data is loaded into the MajorIncome database. A connection is opened, and the tables are loaded directly. Once in the PGAdmin tool, the GUI is used to add the primary and foreign keys.

**Loading the Data into MongoDB**

Starting from the end of part 1, I first use R once more to manipulate the tables in R. I remove the majors.list, since this information is contained in the other three tables, and the structure of MongoDB doesn’t require a separate ID table. Then I add a Student\_type column to mark each table line as belonging to a table. Lastly, tables are exported to CSV for import into MongoDB.

The schema for the MongoDB database can be seen in Appendix C. The plan is to load all three tables into one collection, and take advantage of MongoDB’s nested schema. The columns in majors.list will be top line tables. The other three tables will be contained in three nested lists of attributes.

Appendix B shows the code used to load the data from CSV into a collection. Each of the three CSV files are loaded into the same collection individually, for a collection of 519 entries. In this raw collection, the differing schemas of each of the three tables don’t matter.

After this I use code to bring the data into its final structure in a collection called majorsorganized. Using a foreach loop, I go through each entry to realize the scheme outlined in Appendix C.

**Discussion**

Looking at the description of the data on fivethirtyeight’s github explains the nature of the census data. This data was compiled from the Census Public Use Microdata Series, and the American Community Survey 2010 – 2012.

In the source data there were a few changes in methodology that resulted in different data sets. Recent grads are separated out because there are more data fields available for recent grads than there are for other students. One could imagine that the Census changed their methodology with recent grads, and began publishing new data with these new attributes.

I chose MongoDB as the database of choice for two reasons. For the data as presented, the nested schema provides the best and most expandable way of displaying the aggregated data. This structure allows ways to add different slices of the source data by major type. If there are other aggregations that one would like to see, they could add them to the structure as another nested list.

The main reason why I chose MongoDB however was because of the changing nature of the source data. MongoDB would also work as a store for the individuals that need to be aggregated. The same database can be used even when methodologies are changed. Attributes that remain the same can be aggregated across both methodology types, and the schemaless structure allows changes in methodology to be seamlessly integrated into the overall database.

In my professional experience, economic data can suffer from methodology changes. Econometrics is centered around looking at how variables are related over time, and more history will increase the robustness of a model. Methodology changes force economists to make a choice between splicing data that might not be comparable, or dealing with limited historical data.

Organizing data in MongoDB can make it easier to log these changes, and make these decisions more informed. If further detail is added, nested structures can be added and accounted for. Categories can be compared more easily. They can be made equal on an ad-hoc way, and can be kept separate for other purposes.

Methodology changes are necessary. In my economic experience, it becomes clear that the economy changes over time, and these changes can’t be accounted for before they happen. High Tech didn’t exist as an industry before the 1980s, and types of services based industries have exploded in complexity. MongoDB not only allows these changes to be recorded, but can allow someone to view these changes in an instructive way. Because of this, I believe it would be the best database for this sort of evolving data.