ETL Project

The prevalence of fast food restaurants compared to US demographic data

By Tracey Ha, Pooja Mallard, Ruohong Yuan, Katherine Shamai

Table of Contents

Extract

The datasets we have chosen were extracted from three different sources.

Data set #1 - US Census Bureau Demographic Data

We obtained an API key for the US Census Bureau Demographic Data from http://www.census.gov/developers/.

We then ran “pip install census” in the Python working environment to enable us to access the US Census Bureau API.

We use a census wrapper to retrieve the data from the American Community Survey 5-Year Data (2009-2018) based on ZIP code tabulation area (ZCTA). This is the most recent dataset which covers the demographic data we require for this analysis.

To generate our desired dataset, the US Census data is structured in individual datasets by field. Each census field is denoted with a label like B19013_001E. The current metrics of interest are:

- "B19013_001E": Median household income in the past 12 months
- "B01003_001E": Total population
- "B01002_001E": Median age
- "B19301_001E": Per capita income in the past 12 months
- "B17001_002E": Number of persons whose income in the past 12 months is below the poverty level
- "B23025_005E": Number of unemployed, age 16 or older, in the civilian labor force

The results are then returned as a list of dictionaries, which can be converted into a dataframe.

Data set #2 - Fast Food Restaurants Across America

This dataset was extracted from Kaggle and it came in the form of a downloadable CSV. The dataset comprises of a list of 10,000 fast-food restaurants from Datafiniti’s Business Database, which provides instant access to web data. The CSV includes restaurant address, city, latitude and longitude, coordinates, just to name a few. The dataset was updated between April 2018 and June 2018.

Data set #3 - ZIP Code to ZCTA Cross Walk

This dataset was extracted from UDS Mapper and it came in the form of a downloadable CSV. It converts ZIP codes to ZCTAs (ZIP Code Tabulation Areas) and will allow us to link our US Census Bureau demographic data that uses ZCTAs to our Fast-Food Restaurants Across America data that uses ZIP codes. It is important to note the relationship between ZIP codes and ZCTAs: one ZCTA may contain one or more ZIP codes. This dataset was developed by John Snow Inc. (JSI), for use with UDS Service Area data and is not officially linked to the Census Bureau. This dataset is referred to as the “ZIP / ZCTA Data” throughout the remainder of this document.

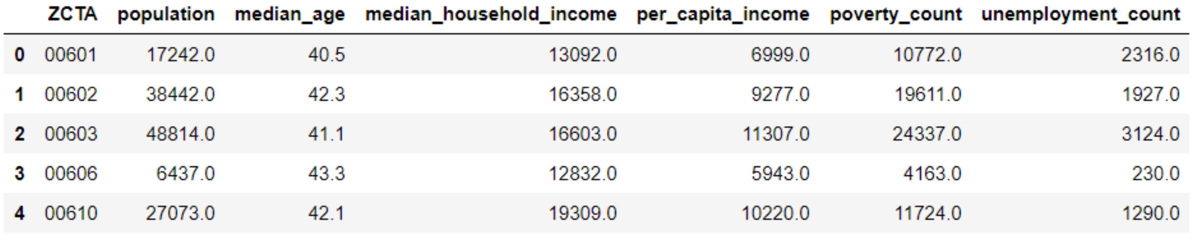
Transform

US Census Bureau Data

Before undertaking any major transformations on our Census data, we loaded the CSV into a Pandas dataframe. We ensured we stored the ZCTAs as string to avoid the dropping of leading 00s.

1st transformation – reordering of dataframe columns

The first transformation applied was to reorder the columns in a logical manner, with non-averaged variables on the left of the table and all averaged variables on the right.

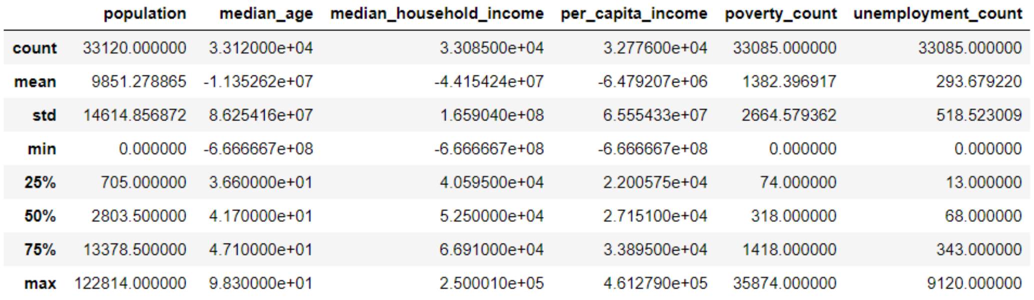


	ZCTA	population	median_age	median_household_income	per_capita_income	poverty_count	unemployment_count
0	00601	17242.0	40.5	13092.0	6999.0	10772.0	2316.0
1	00602	38442.0	42.3	16358.0	9277.0	19611.0	1927.0
2	00603	48814.0	41.1	16603.0	11307.0	24337.0	3124.0
3	00606	6437.0	43.3	12832.0	5943.0	4163.0	230.0
4	00610	27073.0	42.1	19309.0	10220.0	11724.0	1290.0

We then conducted Exploratory Data Analysis (EDA) to determine the types of transformations required. We used:

1. '.describe()' to get summary statistics of the different economic measures
2. '.info()' to identify the data types in our columns
3. '.isnull()' to see how many null values were in our dataframe

The summary statistic table as seen below produced negative mean values for the Median Age, Median Household Income and Per Capita Income columns, signalling to us that the data we contained inconsistencies and potentially inaccurate or invalid values. It also showed a minimum value that was consistent across those three columns: -666,666,666.0.

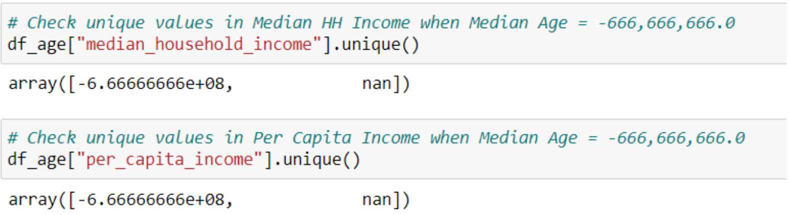


	population	median_age	median_household_income	per_capita_income	poverty_count	unemployment_count
count	33120.000000	3.312000e+04	3.308500e+04	3.277600e+04	33085.000000	33085.000000
mean	9851.278865	-1.135262e+07	-4.415424e+07	-6.479207e+06	1382.396917	293.679220
std	14614.856872	8.625416e+07	1.659040e+08	6.555433e+07	2664.579362	518.523009
min	0.000000	-6.666667e+08	-6.666667e+08	-6.666667e+08	0.000000	0.000000
25%	705.000000	3.660000e+01	4.059500e+04	2.200575e+04	74.000000	13.000000
50%	2803.500000	4.170000e+01	5.250000e+04	2.715100e+04	318.000000	68.000000
75%	13378.500000	4.710000e+01	6.691000e+04	3.389500e+04	1418.000000	343.000000
max	122814.000000	9.830000e+01	2.500010e+05	4.612790e+05	35874.000000	9120.000000

Using a "groupby" function, we were able to determine the frequency of these negative values in each column and concluded that -666,666,666.0 may represent a convention set by the US Census Bureau for data collection errors. Upon further investigation, we found Census reports on data collection errors across different ZCTAs in the 2018 datasets (1). However, there is no concrete information issued by the Census Bureau on whether the use of -666,666,666.0 is indeed a convention for data collection errors or some type of data divider.

2nd transformation – filtering of invalid data

This led us to the second transformation, which was to filter out the rows where the Median Age value was -666,666,666.0. We observed that for all the rows containing this value, there were similar as well as null values in the Median Household Income and Per Capita Income columns.



```
# Check unique values in Median HH Income when Median Age = -666,666,666.0
df_age["median_household_income"].unique()
```

```
array([-6.66666666e+08,             nan])
```

```
# Check unique values in Per Capita Income when Median Age = -666,666,666.0
df_age["per_capita_income"].unique()
```

```
array([-6.66666666e+08,             nan])
```

This reduces the analytical potential/usefulness of the dataset and offers very little insight from the other variables alone. Therefore, we filtered out a total of 564 rows.

3rd transformation – Dropping of null values and filtering

Our next objective was to remove rows in our dataframe that had either 4 null values or a combination of invalid (-666,666,666.0) and null values. Once again, our aim is to provide a database with maximum analytical value and this necessitates clean datasets. To achieve this, we undertook 2 transformations: dropping null values and filtering.

Our first step was to count the null values remaining after our previous transformation, leading us to find 31 rows that contained 4 NaN values. We then used '.dropna()' and subset Median Household Income, which removed those rows and left us with 69 null values in the Per Capita Income column.

We proceeded to identify the unique values in Median Household Income when the Per Capita Income value was NaN. We identified 66 rows that had a combination of null and invalid values and filtered them out, leaving us with only 3 null values in our entire dataset, all of which are in one column.

4th transformation – Conversion of invalid data to NaN format

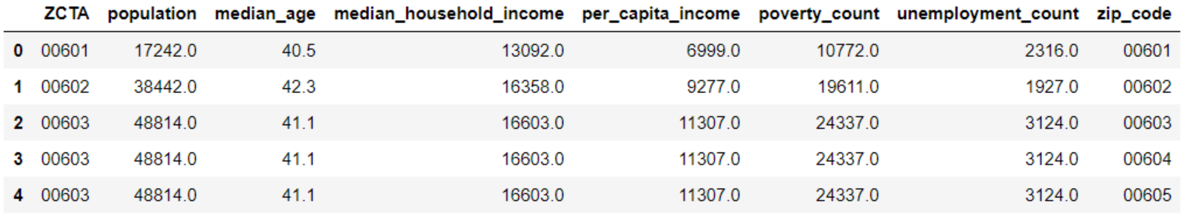
After having cleansed the null values, we were interested in inspecting the 1,568 rows that contained invalid Median Household Income values (-666,666,666.0). Within those rows, all other socio-economic measures had reasonable data and we wanted to preserve that. Therefore, we decided to replace all -666,666,666.0 values with NaN and retain them in our dataset.

5th transformation – Confirmation of useability of census dataset to ZCTA / ZIP dataset

The next transformation was most crucial in facilitating the loading of our datasets into our PostgreSQL database. We merged the census dataframe with the 'ZIP to ZCTA' dataframe (the junction table in our schema). We did this for two reasons:

1. We needed to ensure that all of the ZCTAs in our census table had a matching ZCTA and corresponding ZIP Code in the 'ZIP to ZCTA' junction table otherwise, there would be an error when loading the datasets into our PostgreSQL database

2. We needed to add all of the ZCTAs that are present **only** in the ‘ZIP to ZCTA’ table into the census table. This is done to preserve the one-to-many relationship between the census table and the ‘ZIP to ZCTA’ junction table



	ZCTA	population	median_age	median_household_income	per_capita_income	poverty_count	unemployment_count	zip_code
0	00601	17242.0	40.5	13092.0	6999.0	10772.0	2316.0	00601
1	00602	38442.0	42.3	16358.0	9277.0	19611.0	1927.0	00602
2	00603	48814.0	41.1	16603.0	11307.0	24337.0	3124.0	00603
3	00603	48814.0	41.1	16603.0	11307.0	24337.0	3124.0	00604
4	00603	48814.0	41.1	16603.0	11307.0	24337.0	3124.0	00605

We applied an outer join and did so on the ZCTA column to retain all of the data in the census and ‘ZIP to ZCTA’ tables. In using the outer join, null values were entered into each column for all ZCTAs added. We identified one ZCTA in the census data which was not in our ZIP/ZCTA table. Based on the information in the census data for this ZCTA, it was a low population density area and without further information available to identify a corresponding ZIP Code, we decided to remove this from our census data as it would not have a significant impact on future analysis. If further information becomes available, e.g. updated ZIP/ZCTA conversion tables or new census data, it may be addressed in the future.

6th transformation – Removal of ZCTA duplicates

The merge of the Census Data and ZCTA data resulted in ZCTA duplicates so we needed to drop those duplicated rows to ensure uniqueness as ZCTA represents our primary key in the schema.

7th transformation – Removal of unnecessary data

We then proceeded to drop the ZIP code column as we have created a junction table (‘ZIP to ZCTA’) that links ZCTAs and corresponding ZIP codes to our census table. Lastly, we exported the final census dataset to a CSV for loading to our database.

Fast Food Chain Data

1st transformation – Renaming of columns and removal of duplicates

In order to make field names consistent and match to database schema, we needed to rename our columns to lower_case_snake_style rather than their original camelCase.

Since there are duplicates in restaurants, we use the “key” column to drop duplicates. Before dropping the duplicates, there were 10,000 restaurants. After this transformation, there were 9,343 rows.

2nd transformation – Verification and standardisation of ZIP code format

Some ZIP codes in the dataset have irregular forms. We found out that most ZIP codes follow the format of XXXX, however some of them had this format: XXXXX-XXX. To rectify this, we split the string by the “-” and only kept the first 5 characters.

3rd transformation – Creation of additional reference tables

From a management and data engineering aspect, we needed to split the addresses into two parts, one part is for the street number, and the second part is for the street name. By doing so, our future database will have a quicker response and better performance. In order to achieve a Third Normal Form, we needed to normalise the original restaurant table. Since the city and state is dependent on the ZIP Code, they need to be split to an independent table (ZIP_code) to avoid transitive dependency. We then split the original table to a second independent table - restaurant_address, which only keeps information for address, restaurant ID, and ZIP code.

4th transformation – Confirmation of ZIP Codes

To achieve the many-to-one relationship between the junction ZIP_ZCTA table and the ZIP table, we need to ensure that every ZIP code in the ZIP_ZCTA table has ONE matching ZIP code in the ZIP table and every ZIP code in the ZIP table has AT LEAST ONE matching ZIP code in the junction table.

We firstly checked if every ZIP code in the current ZIP table is in the junction table. The result confirms that all ZIP codes in the ZIP table (normalised from the restaurant address table) are in the junction table. We also extracted all the different ZIP codes in the junction table into the ZIP table. There appears to be sufficient information regarding city and state for these different ZIP codes. From some analysis on the ZIP / ZCTA data, we understand that cities are labelled as “PO_NAME” in the ZIP / ZCTA data.

5th transformation – Replacing restaurant name with restaurant ID

For database management and data engineering purposes, we needed to remove restaurant names from the restaurant_address table, and replace it by unique restaurant_id for the restaurant_name.

We firstly created a third table – restaurant_name which stores the unique ids and their corresponding restaurant names.

Once created, we used restaurant_id from the restaurant table to replace the restaurant_name in the restaurant_address table. This is so that restaurant_name is replaced by restaurant_id and enables easier maintenance of restaurant names if changes were to occur. To replace the restaurant name with restaurant ID, we used a merge function to match the restaurants to their restaurant IDs. Once completed, we removed the restaurant name column.

ZIP to ZCTA Data

For the ZIP to ZCTA table, we loaded the data from CSV into a Pandas dataframe; taking care to load the all columns as string to preserve leading 00 in the ZIP Code data.

1st transformation – Removal of unnecessary columns

Once loaded, we explored the data to see what transformations needed to occur. One of the first transformation activities we undertook was to remove unnecessary columns. As the intent of this table is to map ZIP Codes used in the restaurant data to ZCTA codes used in the Census data, we removed the city, state, ZIP type and how the tables were joined columns. We are then left with a more streamlined table of ZIP Code and ZCTA codes; this will result in a dataset that requires less data storage and processing times.

2nd transformation – Verifying the length of ZIP / ZCTA codes and formatting accordingly

The next step was to check the data integrity of the ZIP Codes and ZCTA codes. A first step was to check the length of the ZIP Codes to ensure the leading 00 were imported correctly into the dataframe. We performed a minimum and maximum length test on both ZIP Codes and ZCTA codes. The ZIP Code results were as expected, i.e. 5 characters. However, we did identify a maximum length of 7 characters for ZCTA codes which indicates there are ZCTA codes which are not as expected, i.e. 5 characters in length.

We then extracted ZCTA codes which were 7 characters in length for further review. It was then evident that this was due to the notation used by the database developers to flag ZIP Codes where there are no corresponding ZCTA codes. We researched why a ZIP Code may not have a corresponding ZCTA code. Based on our research, we then dropped these from our dataset as generally these are ZIP Codes where there is a low population density resulting in a lack of census data, hence ZCTA code. This transformation removed a potential for error when performing further analysis without compromising on the data integrity of ZIP Codes by making an assumption or creating a value for those ZIP Codes.

We then performed a check to ensure all ZIP Codes were unique in the dataset. From a count of total records in the dataset, we noted that the unique count for ZIP Codes equals the total records in the dataset, i.e there were no duplicates ZIP Codes identified. As there may be multiple ZIP Code to one ZCTA code, we expected the total unique ZCTA code count to be less than the total records. In performing this test, our hypothesis was confirmed.

3rd transformation – Formatting of column names

Lastly, we renamed our ZIP Code column to a format that is PostgreSQL friendly, i.e. all lower-case characters. This would avoid potential issues with loading the data and make future analysis easier and more in the convention of SQL language code.

4th transformation – Verification of useability of ZIP / ZCTA data

To ensure our ZIP / ZCTA conversion table would work as expected to our restaurant data, we performed a series of activities to confirm our hypothesis. We imported our restaurant CSV data into a dataframe and merged the ZIP/ZCTA dataframe to the restaurant dataframe. The results showed us that all ZIP Codes in the restaurant data was matched to a ZCTA code but that there were ZIP Codes in the ZIP/ZCTA dataframe which were unmatched in the restaurant dataframe; again, this was expected as our restaurant dataframe is a sample only and not a comprehensive list of restaurant across all of the US.

Load

Selection of database

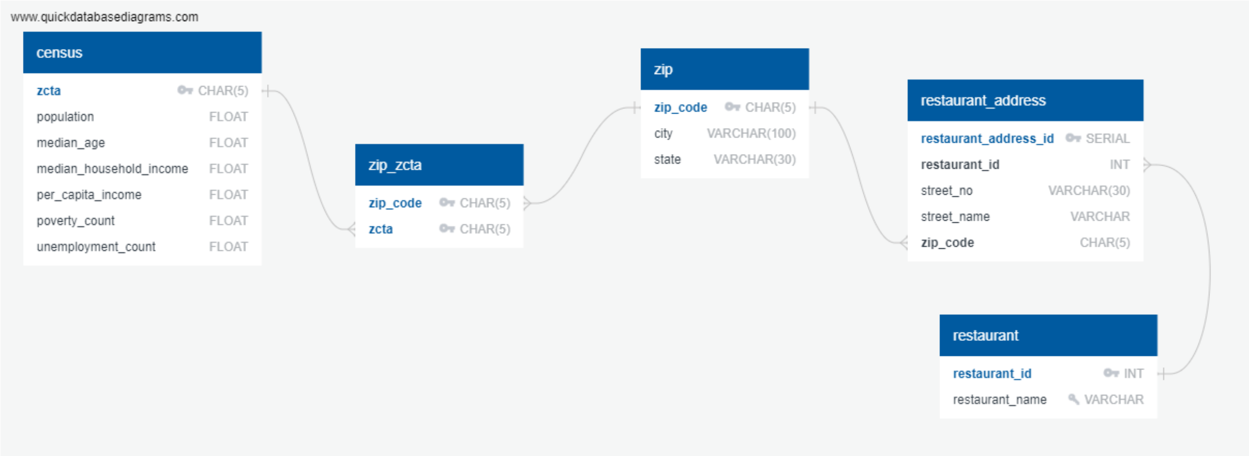
We chose PostgreSQL for our final database as our dataset lends itself to structured data tables linked by clearly defined relationships across the data tables. The datasets we identified have a clear schema definition, and are not frequently updated nor requires real time updating. The analysis we would expect a data analysis to perform using it would be more of a “at a point in time” nature than real time monitoring of trends.

We also researched what other organisations use, and the benefits and limitations of using a MongoDB compared to PostgreSQL. User experience spoke of processing times, data storage, database availability, and the pros and cons of having a flexible schema. Many of the issues identified in real world situations obviously did not apply to our project as our datasets are much smaller. In applying these lessons to our project, we opted an approach which suited our datasets better, and would make it easiest for our intended users to access the data.

References:
http://blog.shippable.com/why-we-moved-from-nosql-mongodb-to-postgressql
https://www.mongodb.com/compare/mongodb-postgresql
https://www.educba.com/mongodb-vs-postgresql/
https://medium.com/better-programming/why-the-guardian-switched-from-mongodb-to-postgresql-861b6cf01e1f
https://blog.panoply.io/postgresql-vs-mongodb#:~:text=PostgreSQL%20uses%20primary%20keys%20to,to%20store%20schema%2Dfree%20data.

Entity Relationship Diagram (ERD)

The ERD for our dataset is set out below, including any primary keys (PK), foreign keys (FK), and the data type associated with each field.



census	
zcta	CHAR(5)
population	FLOAT
median_age	FLOAT
median_household_income	FLOAT
per_capita_income	FLOAT
poverty_count	FLOAT
unemployment_count	FLOAT

zip_zcta	
zip_code	CHAR(5)
zcta	CHAR(5)

zip	
zip_code	CHAR(5)
city	VARCHAR(100)
state	VARCHAR(30)

restaurant_address	
restaurant_address_id	SERIAL
restaurant_id	INT
street_no	VARCHAR(30)
street_name	VARCHAR
zip_code	CHAR(5)

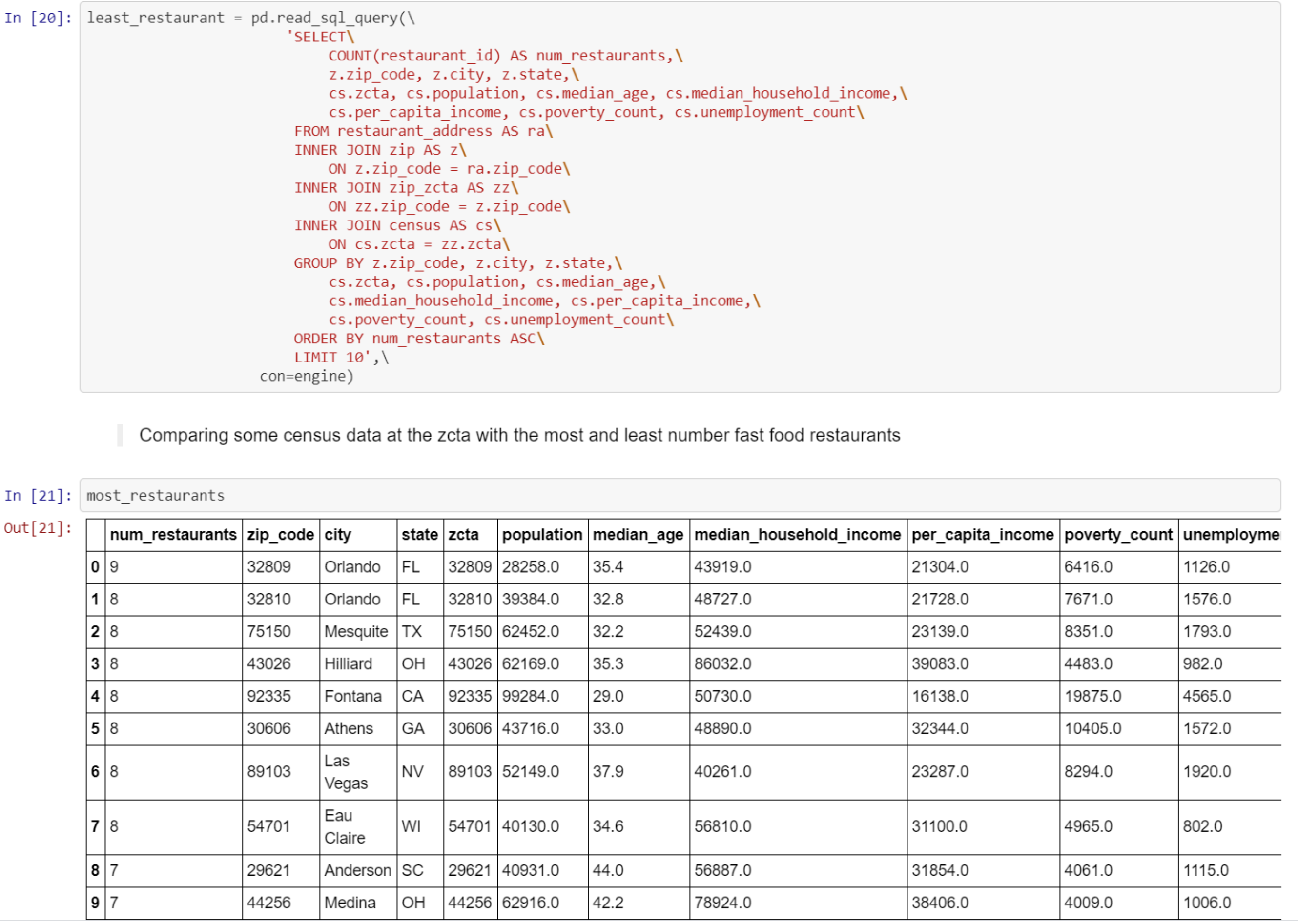
restaurant	
restaurant_id	INT
restaurant_name	VARCHAR

To get the tables to run successfully, the data inside each transformed table needed to comply with the stringent primary keys, foreign keys, and cardinalities set up in the schema. The order of the tables loaded is also important. All of the “parent” tables that have primary keys need to be loaded before any of their “child” tables with foreign keys.

We created a schema using QuickDatabaseDiagram based on the ERD above. We then ran this schema in PostgreSQL and then the load Jupyter Notebook to load our data into PostgreSQL.

Queries for confirm PostgreSQL data

We performed a range of sample queries in PostgreSQL to ensure our data tables are referencing to each other correctly and can be used as anticipated. Our sample script is as follows:



```
In [20]: least_restaurant = pd.read_sql_query(\
                                'SELECT\
                                     COUNT(restaurant_id) AS num_restaurants,\
                                     z.zip_code, z.city, z.state,\
                                     cs.zcta, cs.population, cs.median_age, cs.median_household_income,\
                                     cs.per_capita_income, cs.poverty_count, cs.unemployment_count\
                                 FROM restaurant_address AS ra\
                                 INNER JOIN zip AS z\
                                     ON z.zip_code = ra.zip_code\
                                 INNER JOIN zip_zcta AS zz\
                                     ON zz.zip_code = z.zip_code\
                                 INNER JOIN census AS cs\
                                     ON cs.zcta = zz.zcta\
                                 GROUP BY z.zip_code, z.city, z.state,\
                                     cs.zcta, cs.population, cs.median_age,\
                                     cs.median_household_income, cs.per_capita_income,\
                                     cs.poverty_count, cs.unemployment_count\
                                 ORDER BY num_restaurants ASC\
                                 LIMIT 10',\
                             con=engine)
```

> Comparing some census data at the zcta with the most and least number fast food restaurants

```
In [21]: most_restaurants
```

Out[21]:

	num_restaurants	zip_code	city	state	zcta	population	median_age	median_household_income	per_capita_income	poverty_count	unemployme
0	9	32809	Orlando	FL	32809	28258.0	35.4	43919.0	21304.0	6416.0	1126.0
1	8	32810	Orlando	FL	32810	39384.0	32.8	48727.0	21728.0	7671.0	1576.0
2	8	75150	Mesquite	TX	75150	62452.0	32.2	52439.0	23139.0	8351.0	1793.0
3	8	43026	Hilliard	OH	43026	62169.0	35.3	86032.0	39083.0	4483.0	982.0
4	8	92335	Fontana	CA	92335	99284.0	29.0	50730.0	16138.0	19875.0	4565.0
5	8	30606	Athens	GA	30606	43716.0	33.0	48890.0	32344.0	10405.0	1572.0
6	8	89103	Las Vegas	NV	89103	52149.0	37.9	40261.0	23287.0	8294.0	1920.0
7	8	54701	Eau Claire	WI	54701	40130.0	34.6	56810.0	31100.0	4965.0	802.0
8	7	29621	Anderson	SC	29621	40931.0	44.0	56887.0	31854.0	4061.0	1115.0
9	7	44256	Medina	OH	44256	62916.0	42.2	78924.0	38406.0	4009.0	1006.0

Limitations

Census data

The US census data is not a clean and neat dataset. There were a number of rows and fields which contained invalid values as well as null or zero values. It is unclear if the invalid values represented a particular value or if they were placeholders, section dividers or genuine invalid entries. Information on the census data website does not provide any further clarity on these invalid values nor did broader research yield any further clarity. We have left the null or zero values as they may represent the actual census outcomes but have removed the invalid data from our dataset.

Restaurant data

The CSV dataset we were able to identify for the restaurant data is a sample dataset only and contains 10,000 entries. It is unclear how these entries were limited, e.g. first 10,000 in a larger dataset or a random 10,000 entries. Due to this limitation, the results from any analysis performed may not be a full representation of the frequency of fast food business to population census information.

We have normalised the original restaurant data by:

1. Separating street number and street name from street address, and
2. Separating the repeated restaurant names into another table tracked by restaurant_id to ensure referential integrity in the sql database.

For the first normalisation, we would like to further break the restaurant addresses into street type (street, boulevard, avenue, etc.). However, there are some street addresses that have no clear structure of street number - street name - street type. To achieve this, we need to have a much more detailed look at the street addresses, which has not been feasible due to the short project timeline.

For the second normalisation, one issue is with the name of the same restaurant can sometimes be worded differently, and thus showing up as unique values in the table.

ZIP / ZCTA limitation

From our extensive research, there is no definitive conversion or mapping between ZCTA and ZIP Codes. The US Census bureau uses ZCTA codes which are an aggregation of ZIP Codes used and determined by the US Postal Services. From time to time, ZIP Codes may change based on operational requirements of the US Postal Services. As such, any mapping would need to be kept current. We identified a source of mapping between ZIP / ZCTA which was intended to be used for healthcare services. It is unclear if this is a full and comprehensive dataset. The authors did notate how they performed the ZIP to ZCTA joins but we have not validated these as being accurate.