City Measure

Marquise Trawick

### Configuration Details

The project will be performed on the Windows operating system. Three software’s used in the development; [Visual Studio Code](https://code.visualstudio.com/download), [Anaconda Navigator](https://www.anaconda.com/), and [MongoDB Community Edition](https://www.mongodb.com/docs/manual/installation/). It was programmed in [Python](https://www.python.org/downloads/) version 3.9.12.

Visual Studio Code or VS Code is a code editor refined and optimized for building and debugging modern web and cloud applications. Anaconda Navigator is a desktop graphical user interface included in anaconda that allows users to launch applications and easily manage conda packages, environments, and channels without the need to use command line. MongoDB is a NoSQL database program. It uses Json-like documents with optional schemas. MongoDB Compass is an interactive tool for querying, optimizing and analyzing the mongodb data, and can also be used to host a local database.

Python is a high-level, general purpose programming language. There are multiple python packages that need to be installed for this project. Of the packages needed the Re, JSON, and time module are built into python. The other packages needed are unidecode, geopy, matplotlib, seaborn, pandas, numpy, and pymongo. Unidecode is a library for converting Unicode data and tries to represent it in ASCII characters of the US Keyboard. Geopy is a library that can be used to locate the coordinates of an address, city, country, and landmark across the globe using third party geocoders and other sources. Matplotlib and Seaborn are for creating data visualizations. Pandas is a python data analysis library. Numpy is a library python uses to work with arrays, it is a crucial application that is used in many other python libraries.

### System Configuration and Execution

Open the Anaconda Navigator first as it manages the packages and environment. There is no need to change the environment as the base (root) environment is all that’s needed. Within the home tab is a list of applications that anaconda works with. Search for VS Code and click launch as shown below.

A screenshot of a computer

Description automatically generated

The Visual Studio Code app should open. On the left side there should be a vertical taskbar. The fifth icon from the top is the extensions tab, either click on it or press ctrl+shift+x. With the Extension marketplace open type in python in the search bar it should then show a list of python extensions. Click and install the Python Extension Package developed by Microsoft.

A screenshot of a computer

Description automatically generated

Once the extension is installed, click file in the top menu and choose open folder. Navigate to the folder that the project is in and open it. At the bottom of the window should be a terminal, using the links in the references get the install line and enter them into the terminal. Use the conda install if available if not use pip. Once the libraries are installed, then open the MongoDB Compass application, copy the URI link and click connect.

A screenshot of a computer

Description automatically generated with medium confidence

Copy URI

Within the VS Code Editor open the Main\_Complete.py file, at the top there is a variable called ‘MongDBLink’ on the fourth line, paste the URI there. Press F5 on the keyboard to execute the code. Be warned that because of some limitations it can take 30-60 minutes to run. If you would like to run the program after the data was cleaned.

A picture containing text, screenshot, software, font

Description automatically generated

Paste URI

If you would like to run the program after the data was cleaned just run the file Main\_Partial.py. Upload a csv file that has already had their cities located, therefore shortening the run time down to at most five minutes. But do know that nothing will appear in your mongo database due to not needing to connect to it.

The Similarity results will appear in the terminal but all histograms and graphs will be created and saved into the same directory as the python file.

### Goal

Initially starting with the yelp challenges; Find the difference between cities! I believed that the cities were similar based on the data of the business within them.

### Collection

The Dataset used in this project is the Yelp Open Dataset. Yelp is a company that helps connect people with local businesses. The dataset is a subset of yelps businesses, reviews, and user data that the company released that the public could use to learn natural language processing or sample production data in applications. The Dataset can be downloaded directly from their site, it is a compressed zip file with a size of 4 gigabytes. The zip file contains five datasets already in json format. The file used is only the ‘yelp\_academic\_dataset\_business.json’ file, that contains over 150,000 entries.

Data Process

After pasting in a local mongoDB URI link as shown before. The URI is used to connect with the local database. Using mongodb’s python library pymongo, a client connection is created to the database. If it succeeds it should print the list of databases within the server, first time users should only have three databases: admin, config and local. It will then create a database named ‘CIS492’ and a collection called ‘Yelp\_Dataset\_80,000’. If it fails to connect, then it will print that the connection failed in the terminal.

A screen shot of a computer program

Description automatically generated with low confidence

After the connection with the database is established, its time to insert the data from the json file into the database. It’s a simple method that opens and reads the file, using the loads function from the JSON library to parse a json string and convert it into a python dictionary before inserting the data entries into the database one at a time using pymongo’s insert\_one function. Although the dataset has 150,000 entries but due to a certain limitation, we will only insert the first 80,000 entries.

A screen shot of a computer code

Description automatically generated with low confidence

Once this is complete, reload the database in mongoDB Compass and open the Yelp\_Dataset\_80,000 collection, it should look like the image below.

A screenshot of a computer

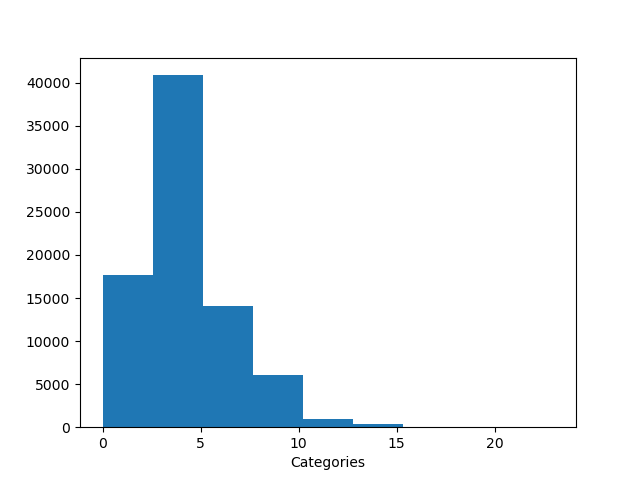
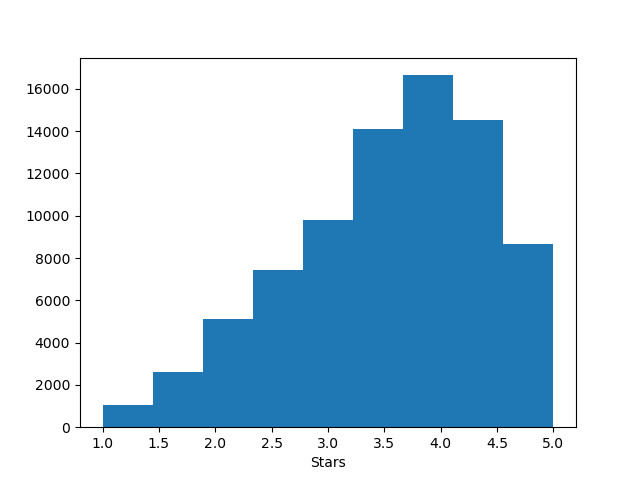
Description automatically generated with medium confidence

With the database getting its entries it was time to grab the essential fields. Using a query, all entries are grabbed but only taking seven fields: Business Name, State, City, Categories, Stars, Latitude and Longitude. A dictionary is created to contain those fields as a list, and using a loop they are appended into the lists. Finally using pandas to convert the dictionary into a data frame and return it. The city name had to be removed of special characters using the re module. Categories were written as strings so having it split turned it into a list from which only the length was needed.

A screen shot of a computer

Description automatically generated with low confidence

A Histogram is created of the Stars and Categories for reference. It appeared that most businesses had a rating between 3.5 to 4.5 stars. 3 stars was the next most businesses but was behind by at least 4000 compared to 3.5 to 4.5. A majority of businesses have 3 to 5 categories as it was twice the size of the next highest which was between 0 and 2 categories.



### Cleaning

After the chosen field is placed in a data frame, the city names must be cleaned as some names are misspelled, abbreviated or empty. Panda’s data manipulation makes it easy to take specific columns such as the city names, latitude and longitude. Taking these fields, they are inserted into a method meant to get correct city for the business based on its location. This method takes at least 30 minutes to find the correct city name.

As most cities are within the same city and most city names are correct, a dictionary was created that will store the incorrect name as a key and a correct city name as a value. A list named ‘cityNameList’ that will replace the city names in the data frame is also created. The three list from the data frame used in a for loop that grabs and entry from a certain index.

If the city name was in dictionary as a key, then we could just use insert the value into the cityNameList. If the city name does not match any keys in the dictionary, then another method will be called to geolocate business and return the city name.

After getting the city name from geolocation we have all its characters converted to those of the us keyboard. We then add a new entry into the dictionary with the incorrect name as the key and the correct name as the value. Of course, after realizing that some cities in the list do not have names, they were called unknown cities. When the loop is complete all cities should be correctly spelled and in English characters.

A picture containing text, screenshot

Description automatically generated

Geopy has a method called reverse that can take a set of coordinates and return the address. Geopy is an API with specific limitations. The first was that only 86,000 requests could be made a day, but this was solved by limiting the number of entries from the original 150,000 to 80,000. The second limitation was that it can only perform one request a second. It had a simple solution, which was to use pythons time module to wait one second before performing the geolocation. But this created a major limitation because it meant that it would take twenty-two hours to get every entry address. I tried it twice before I made a solution. I talked about the solution in the previous method where I created a dictionary that stored incorrect and correct city names in a dictionary so that I only had to locate city names that are not already stored in the dictionary. The time was shortened exponentially, but there was one more problem with geolocation, well it was a problem with lots of open API’s. There was a chance to receive an HTTP error, which can happen from disconnecting. The solution is within the next method.

In the image below is the method that used geopy’s geolocator to find a businesses address. It checks if the geolocator can find the position of the coordinates before getting to the location, it was to prevent false coordinates. A loop is initiated to get the address from the location and if there is an http error it will try to get the address again three more times. If it succeeds, the address is given as a dictionary so we can use the key to get the city name and if it fails it returns the city’s original name.

A picture containing text, screenshot, font

Description automatically generated

### Transformation

The city names were as correct as they can be, so it was time to merge the businesses into their respective cities. The method took a data frame as a parameter and returned another data frame. A dictionary was created that would contain an empty list of their respective columns; City, State, Businesses, Star Average, Category Average, Stars and Categories.

A loop is created to iterate through all entries of the data frame through the City, Stars, Category and State Columns. First it would check if the entry’s city was already in the city list within the dictionary, if it was then the grab the index that it was located. Then and at that index within the business, stars, and category list within the dictionary, when would the entries stars, category and increment the businesses amount. If the city was not in the list, then when would append them into the dictionary’s list. After the loop is complete another two loops that would get the average of stars ratings and categories amount and enter them in the Star\_Avg and Category\_Avg list within the dictionary. The dictionary is then converted into a new data frame. After merging the total entries should have decreased from 80,000 to around 773. It may not be exactly 773 in case all attempts of locating an address fail due to HTTP Errors, but there should be no more than 780 entries otherwise check internet connection.

A screen shot of a computer code

Description automatically generated with low confidenceThe difference between the original and merged data frame is shown below. Before merging the stars were within a range of one to five, because a business was rated within that range but after merging the star column shows the sum of business stars within the city. It was a similar situation with columns, but it had a range of zero to ten.  
A screenshot of a computer

Description automatically generated with medium confidence

A Histogram is shown below of the Businesses, Star\_Avg and Category\_Avg columns. As can be seen with the business category, a majority of the cities have less than a thousand businesses, with a few outliers having more than a thousand. Looking at the description gotten from pandas below the graphs, the max a city had was 7,782 businesses. It appeared that 75% of them had forty-four or less business.

Note: When the program runs the column descriptions will be printed in the terminal.

A screenshot of a graph

Description automatically generated with medium confidence

The Star\_Avg graph looked better than the businesses but most of them are concentrated between three and four. The description below the graph gives a better number as 50% of the cities are rated between 3.2 and 3.9 stars, which is more precise than the histogram shown from the original data where most business had a star rating between 3 and 4.5.

Note: When the program runs the column descriptions will be printed in the terminal.

A screenshot of a graph

Description automatically generated with medium confidence

The Category\_Avg histogram although more precise but was fairly similar to the histogram from the original data set. With almost all cities having a category average between 2.5 and 5. The Description below it shows that 50% of the cities have and average of 3.8 to 4.7 categories.

Note: When the program runs the column descriptions will be printed in the terminal.

A screenshot of a graph

Description automatically generated with medium confidence

### Normalization

From what was shown before in the histogram and descriptions, it was determined that the outliers needed to be Normalized by Clipping. Clipping is not really a normalization technique but a way to put a limit on the values above or below a certain value to a fixed value close where the majority of the values are concentrated. It can be used before or after using other normalization techniques. For this panda’s has a built in clipping function. From what was discovered before businesses needed an upper limit, Star\_Avg and Category\_Avg both need a lower and upper limit. For the lower limit I used the 25% range from their descriptions and for the upper limit I used the 75% range.

After clipping I used a Min-Max Normalization. Min-Max Normalization is a simple rescaling method that rescales the range of values to be between zero and one. It is simply taking a value and subtracting it by the minimum and dividing by difference between the maximum and minimum values.

A graph on a computer screen

Description automatically generated with low confidenceA screen shot of a computer code

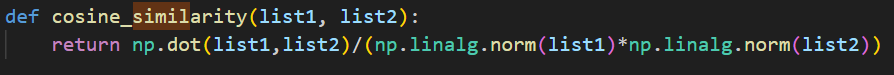
Description automatically generated with low confidence

Although the results may seem more unbalanced in the histograms above, but in the scatterplot below should give a better visual of what happened. The points at the edge are those outliers that were clipped, but what remains is the most important data. If I had shown the scatterplot before now it would have looked like a line because of the outliers, when in fact most of the data was clustered together.

Chart, scatter chart

Description automatically generated

The Columns were put through two similarity measures to determine how close they were. I used the cosine similarity to see how close the cities were on an angular level and the Euclidean to determine how close in distance the cities were. Cosine similarity was the measure of angle between two vectors. The Euclidean distance is the measure of distance between two points regardless of dimension.



A picture containing text, font, screenshot, line

Description automatically generated

Note: When the program runs the results of the measures will show in the terminal.

### Results

The results show that Cities are not as close as the businesses within them on an angular level. But quite the contrast businesses have a distance forty-seven times as much as cities do. The cities before being normalized had a greater lower angular distance than businesses within the original data, but when looking at things from a macro perspective it’s hard to tell how close things are.

A picture containing text, screenshot, font, line

Description automatically generated

A picture containing text, screenshot, font, number

Description automatically generated

A picture containing text, screenshot, font, number

Description automatically generated

### References

[Anaconda Navigator](https://www.anaconda.com/)

[MongoDB Community Edition](https://www.mongodb.com/docs/manual/installation/)

[Python](https://www.python.org/downloads/)

* [Geopy Install](https://pypi.org/project/geopy/)
* [Matplotlib Install](https://matplotlib.org/stable/users/installing/index.html)
* [NumPy Install](https://numpy.org/install/)
* [Pandas Install](https://pandas.pydata.org/pandas-docs/stable/getting_started/install.html)
* [PyMongo Install](https://www.mongodb.com/docs/drivers/pymongo/)
* [Seaborn Install](https://seaborn.pydata.org/installing.html)
* [Unidecode Install](https://pypi.org/project/Unidecode/)

[Visual Studio Code](https://code.visualstudio.com/download)

[Yelp Dataset](https://www.yelp.com/dataset)