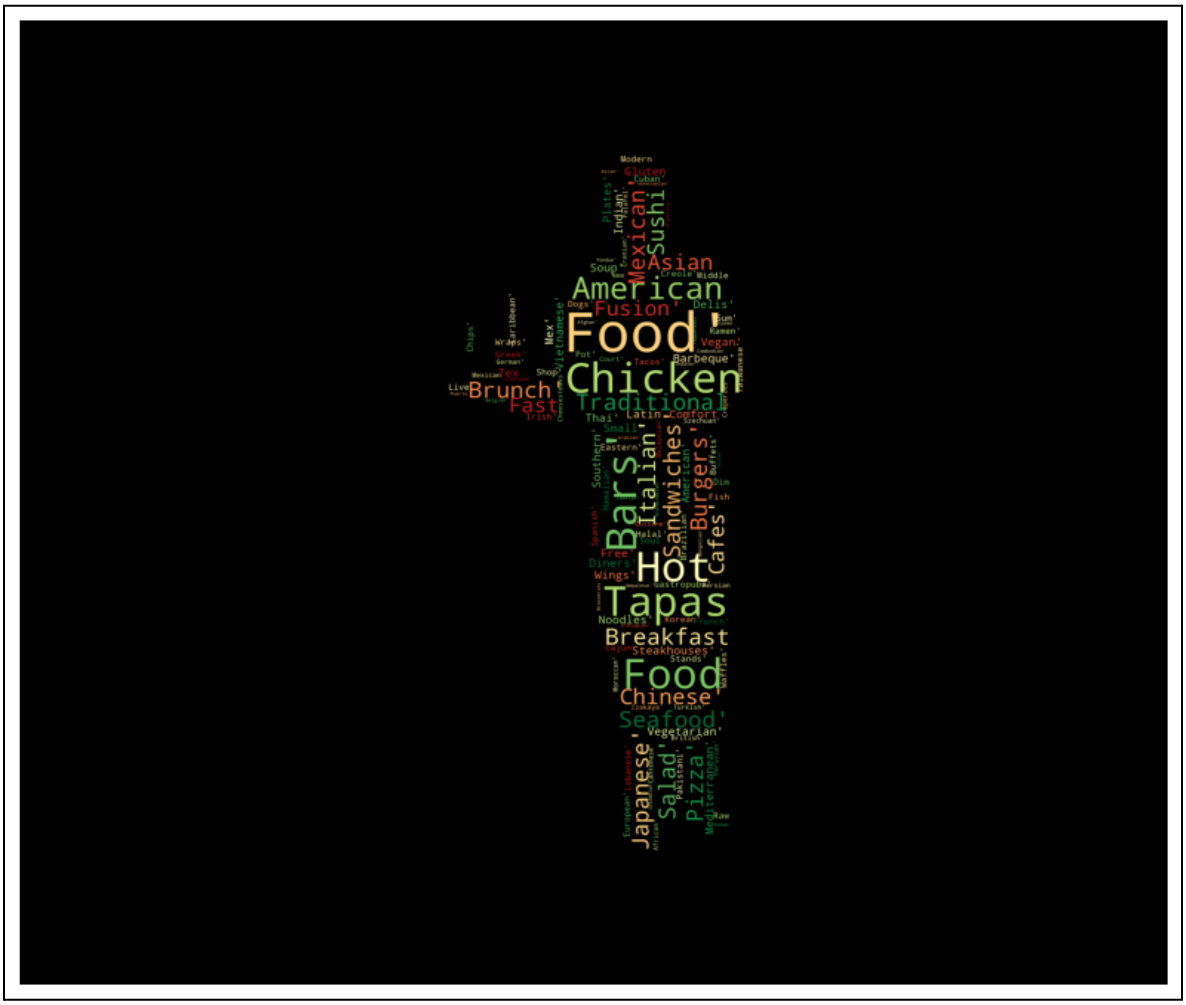
Restaurant Recommendation and Reviews Analysis



Name	Contributions
Arya Gupta	Descriptive Statistics, VADER model, Plots, Research Questions Parts 1 and 2
Jared Choy	Cover Letter, Intro, Dataset Description, Descriptive Statistics, Region Preferences, Conclusion, Reference
Rohan Arumugam	Descriptive Statistics, SVD, Recommendation System, Cover Letter, Graphs
Christopher Monzon	Data Cleaning/Wrangling, Web Scraping, Research Questions Parts 1 and 2

Contents:

Cover Letter ---------- 2

Introduction ----------2

Dataset Description ---------- 2-3

Descriptive Statistics---------- 3-4

Research Questions/Proposed Methods ---------- 5

Methodology & Analysis ---------- 5-11

Conclusion ---------- 12

Reference ----------13

Proofs ----------14

Appendix ----------14-43

Cover Letter:

From our project, we found that a restaurant's main focus should be improving the experience of the consumer. We found the attributes that are best reflective of influencing service, and we also discovered the cuisines/categories most present across the vast landscape of North America. Additionally, we used collaborative filtering techniques, such as truncated Singular Value Decomposition (SVD), to extract the most important information from a matrix with user ratings by restaurant to build a recommendation system. This recommendation system will take in the name of a restaurant and provide similar restaurants that are highly correlated with the original restaurant. In order to optimize our code, we utilized parallel computing techniques, such as parallelizing tasks across multiple processes and creating temporary files to have computations run across multiple processes.

Introduction:

According to the National Restaurant Association, the projected annual sales in the restaurant industry are around $863 billion dollars, which is roughly 4% of America's GDP. Additionally, consumers spend about 12.6% of their income on food [1]. Needless to say, restaurants play a significant role in our daily lives, and their importance extends far beyond providing nourishment. They are essential gathering places for socializing, celebrating special occasions, and creating memorable experiences. There are more than one million restaurants in the United States alone [2], and from these figures, one might understand how difficult it might be for one to choose a restaurant to satiate their hunger. Furthermore, densely populated cities provide far greater options for restaurant-goers, creating more chaos in decision making. Due to the sheer prevalence and importance of restaurants in enriching our lives, our aim is to build a recommendation system that can provide users a way to find restaurants suitable to their palate and reflective of their wants. Additionally, to help businesses understand the wants and needs of their clientele, we will provide different consumer preferences as understanding these preferences are instrumental for growing a business. This will help restaurateurs gain a competitive advantage in the market, increase their sales, and provide consumers with the best experience possible. We hope to elucidate the truths surrounding what consumers are looking for, as well as the general trends surrounding specific cuisines.

Dataset Description:

Pulled from *Kaggle* [3], we used two downloadable datasets to construct one combined main frame. Both datasets are pulled from Yelp reviews and are huge in size; 'precovid.csv' has 5,172,198 reviews, whilst 'postcovid.csv' has 400,295 reviews. Additionally, each data frame has the same 21 columns detailing various parameters. To combine both of these datasets, we used a merge function to see if restaurants were open both pre and post covid. This combined data frame serves as the basis

for the rest of our project. The following column headers with description are provided in the table below.

** *Note that not all columns were used depending on the task*

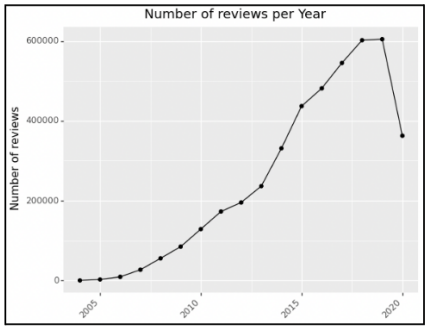
Attribute	Description
Business ID	Each restaurant's ID. Useful for differentiating locations in a chain of restaurants.
Name	The name of each restaurant.
Address	Each restaurant's address
State	The state that each restaurant resides in. 11 U.S. states, 2 Canadian territories
City	City the restaurant resides in
Postal Code	Zip code
Latitude/Longitude	The cartesian coordinates of each location, used for Tableau Visualization
Stars	The overall rating of the restaurant
Review Count	The number of reviews for each restaurant
Is Open	Binary variable. 0 if the restaurant is closed down. 1 if open.
Categories	The type of establishment a restaurant is and the cuisine it serves
Customer Stars	The rating the customer prescribed to the restaurant
Useful	A variable on a 0-5 scale that measures how relevant a review was.
Funny	A variable on a 0-5 scale that measures how funny a review was.
Cool	A variable on a 0-5 scale that measures how cool a review is.
Text	The actual review itself.
Date	The day of the reviews publication

Descriptive Statistics:

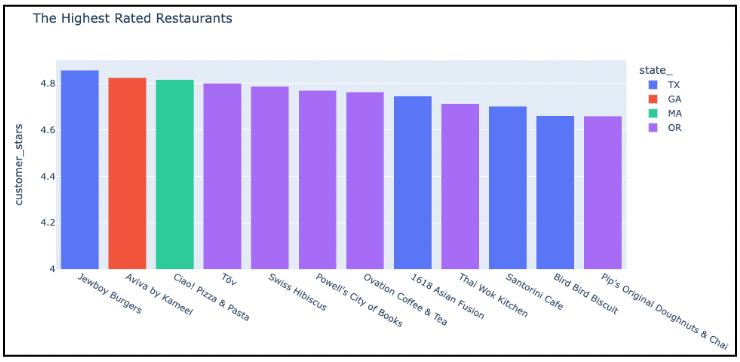
Our dataset has reviews from restaurants in eleven different cities spanning over the US and Canada. From the Tableau visualization [4], we saw that some cities were on the cusp of a regional hotspot, and thus combined them. In total, we looked at eight regions.

Oregon had the highest rated restaurants in the entire population, and it could be due to Portland's reputation as one of the best food cities in America [5].

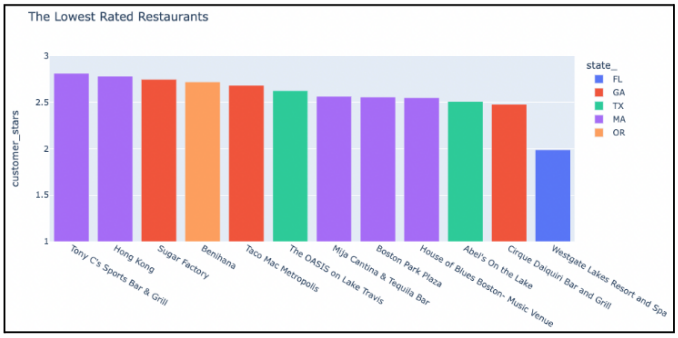
Massachusetts had the most restaurants across the entire region space which could be due to the population, average income of inhabitants, and how dense the city is [6].



The number of reviews drastically decreased during the start of the pandemic, which makes sense as people were not leaving their homes, and in turn did not eat out. And even if reviews were made, the overall restaurant experience was not fully brought to fruition.



The highest rated restaurants. Most of them are in Texas and Oregon. Recall that Oregon had the highest rated restaurants in the entire dataset.



The lowest rated restaurants. It's interesting to note how a good chunk of them are in Boston and Southern States.

	Overall Stars	Customer Stars	Useful	Funny	Cool	Number of Reviews
Mean	4.179587	4.173274	0.788767	0.298166	0.504876	2679.657661
Median	4.5	5	0	0	0	2435
Mode	4.5	5	0	0	0	3761
σ	0.444880	1.193325	2.722549	1.622924	2.384565	1921.518831

We performed descriptive statistics in order to better understand the data we're working with, as well as make assumptions and gather insight. From our findings, we only get a glimpse of the restaurant landscape, which is why we will perform further analysis to gain a full perspective.

Research Questions/Proposed Methods:

In this project, we wish to ask the following questions:

- *How does the sentiment of the review correspond with the star rating?*
    - *How does the sentiment of service correspond with the star rating?*
- *What phrases/words appear the most in positive/negative reviews?*
    - *What words appear the most in reviews with high sentiment about service?*
- *What preferences do certain regions in the United States & Canada lean towards?*
    - *What preferences perform the highest?*
    - *What preferences perform the lowest?*
- *Can we perform collaborative filtering to create a recommendation system?*

We wanted to ask these questions for various reasons. For our first question, we wanted to confirm our assumption that a better review would equal a better star rating, because if the two were not correlated, it would be difficult to move forward with our project. Furthermore, if our model showed that a more positive review sentiment did correlate with higher star ratings, we would be able to find restaurants with the best service using sentiment analysis. For our second question, we wanted to see if there was a tendency to use certain words in either a positive or negative review. For instance, if these words described a certain aspect of the restaurant, could a business owner use this to find ways to improve their establishment? Conversely, a restaurateur could also find keywords that consumers are looking for when they dine, for example, “fast service” or “friendly staff”.
For the third question, we wanted to see if there was a difference between preferences across the regions and if so, was there any reasoning behind it. We found this to be an important question to ask, as we wanted to see what the wants/needs of the consumers are across the continent.

To accomplish the aforementioned questions we have proposed, we will use the following methods:

- *Natural Language Processing (NLP)*
- *Web Scraping*
- *Singular Value Decomposition (SVD)*
- *Parallel Computing/Multiprocessing*

Methodology & Analysis:

How does the sentiment of the rating correspond with the star rating?

To find if the sentiment of the customer review corresponds with star rating, we used the VADER (**Valence Aware Dictionary for Sentiment Reasoning**) model. This model provides sentiment scores from -1 to 1 by analyzing terms and labeling them as positive, negative, or neutral, and also offers a “compound” score to quantify the overall sentiment. We created a new column in our dataframe called “compound” which found the compound score of each review using the VADER model. To verify that the model works, we plotted the compound score

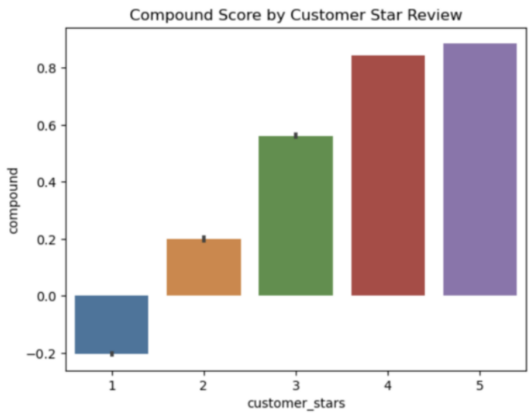


Figure 1

against the customer stars, and found a notable positive trend between the two (Figure 1). As the compound score became more positive, so did the number of stars.

We found that service is an extremely important factor in terms of a customer's star rating, so we wanted to find how the sentiment of a customer writing about service corresponds with their overall star rating. This can give an indicator of how well service correlates with a customer's start rating, showing the importance of service to a customer's overall experience. We extracted the specific sentences where a customer is talking about service, as well as their given star rating. We did this by manually finding words that commonly appear in these sentences and found that 'service,' 'server,' 'staff,' 'waiter', and 'waitress' almost always appear when a review is talking about service. Using the VADER model, we found the compound sentiment score of each of these sentences related to service and compared it to the customer's star rating. We find that the graph of service sentiment (Figure 2) looks very similar to the overall sentiment by customer star reviews, indicating that service is likely one of the most important factors in a customer's star rating.

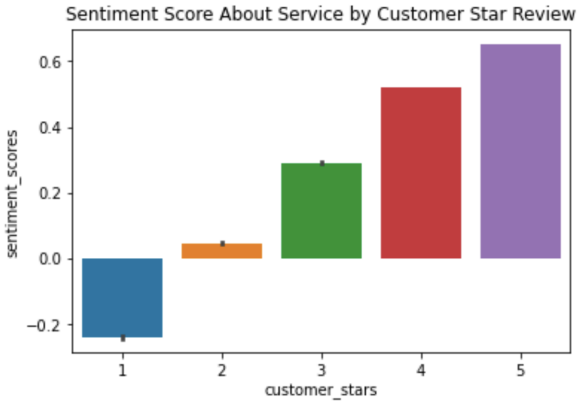


Figure 2

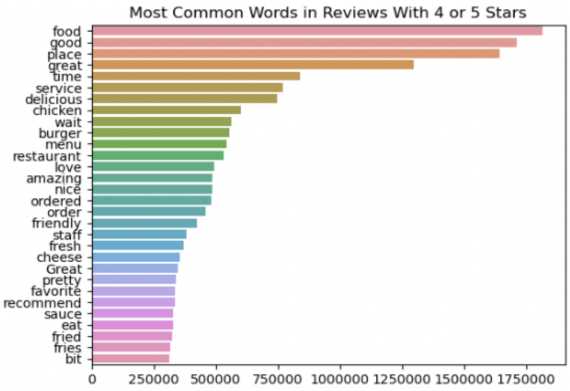


Figure 3

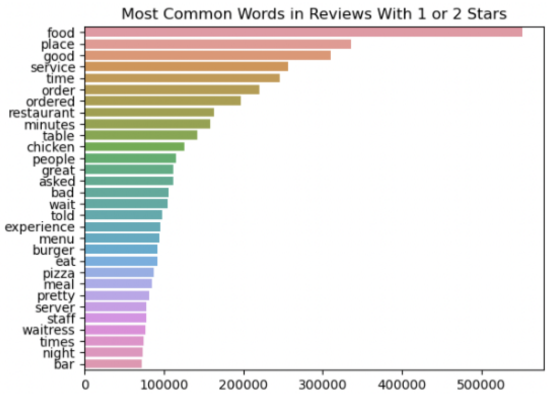


Figure 4

What words appear the most in positive/negative reviews?

To find which words appear the most in positive and negative reviews, we subsetted our dataset by customer ratings with 4 or more stars, and customer ratings with less than 2 stars. We called these new data frames 'positive_reviews' and 'negative_reviews' respectively. Since our goal was to find the most frequent words in reviews to help businesses evaluate consumer sentiment, we needed to remove "stop words" such as "the", "and", "go", etc as they provided no valuable feedback for a business. We created a function called "common_words", which accepted a dataframe and converted every review from the dataframe into a single list. That list was passed into the nltk.word_tokenize function, which extracted tokens from a string of characters and finally, we filtered out all stopwords from the list. Using Python's "collections" module we were able to find the most common words occurring in both positive and negative reviews, and also how frequently they were used. Figures 3 and 4 show the 30 most frequently used words in positive and negative reviews. There are 16 words that are in both positive and negative reviews, including 'service', 'wait', 'order', etc. Words that were frequently used in negative

reviews but not positive reviews were “bad”, “experience”, “minutes”, “server”, etc. This shows that negative reviews were highly impacted by the service of the restaurant.

After finding that service is a large factor in a customer’s rating of the restaurant, we wanted to find which words related to service correspond the most with a positive experience. Using the sentences related to services we’ve previously extracted, we found the sentiment rating of these specific sentences and separated them by positive or negative service, where positive service has a sentiment rating greater than zero. We used the “common_words” function to find the most common words when reviews talk about positive service. We found that ‘friendly,’ ‘nice,’ ‘fast,’ ‘attentive,’ and ‘quick’ were the most common words describing a restaurant’s positive service. This shows that customers value timely service, thoughtful servers, and when staff pay attention to the needs of a restaurant-goer.

What preferences do certain regions in the United States & Canada lean towards?

The Total Most Popular Cuisines by Reviews

Cuisines	Number of Reviews
American (Traditional)	806853
Breakfast & Brunch	739158
Sandwiches	564755
Seafood	462278
Pizza	397756
Mexican	389948
Burgers	364958
Italian	362105
Japanese	319495
Salad	284964

Figure 5

To see what preferences certain regions in the U.S. and Canada lean towards, we first read in the dataframes to prepare our samples. On the first go around, we saw that the time to load in the datasets was around nine minutes of computation time, and to combine the two it took six minutes. Because it would take far too long to run this on each of our unique devices, we thought that perhaps we could save the files to CSV format. However, these files were also too big to download. Instead, we found out that if you only read in the column values that you were actually going to use, you could cut computation time significantly. Thus for this question, we only used seven columns. Once we had loaded in the dataframes, we merged them by seeing if the restaurants were open in both the pre-covid frame, and the post-covid frame.

To find out what preferences each leaned towards, we first had to perform some web scraping. We web scraped a Yelp published blog [7], detailing a complete list of the business categories that Yelp offers. As we were only concerned with the culinary aspect of the list, we scraped the restaurant section specifically, and concatenated each entry to a python list of words that contained all the

The Total Most Popular Categories by Reviews

Category	Number of Reviews
Restaurants	4302877
Food	1414395
Nightlife	1279483
Bars	1238157
American (New)	765249
Coffee & Tea	354412
Event Planning & Services	335515
Cocktail Bars	309954
Desserts	245858
Bakeries	212156

Figure 6

types of cuisine. After that, we subsetted the categories column of the dataframe to only add the row value, if the cuisine was found in our cuisine list. Once this was accomplished, we made a frequency distribution and converted the top ten into a dataframe. To get the most common establishments, we simply subsetted the categories column by adding values not found in the cuisine list. Figures 5 and 6 show the most popular cuisines/categories across the entire dataframe.

A lot of these are unsurprising, as these are all extremely common cuisines and establishments that are popular for all walks of life. And as we are looking at a mostly American appetite, this list would make sense. The next thing we wanted to do was see the preferences for each region. To accomplish this, we ran a multiprocessing function to be able to create eight dataframes for most popular cuisine, and most popular category. As we mentioned before, we made ephemeral files within a cell, and then called the function when we needed to. Using the function, we appended each frame to a list via the parallel computing function. We were able to do this by subsetting the main dataframe by region, converting each frame to a CSV file, and then accessing each region's file for each thread in our process. An example of a region's preferences can be seen in figure 7 and 8 of Texas's most common cuisines, and establishments.

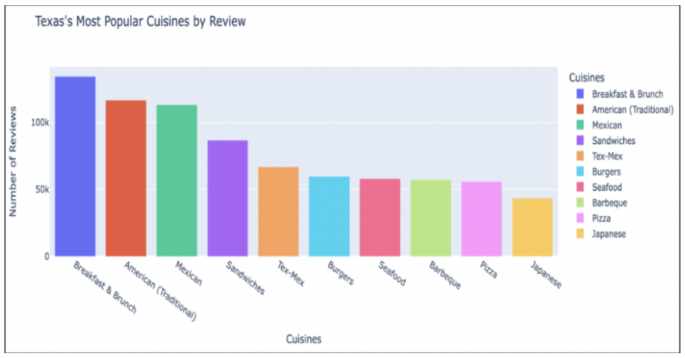


Figure 7

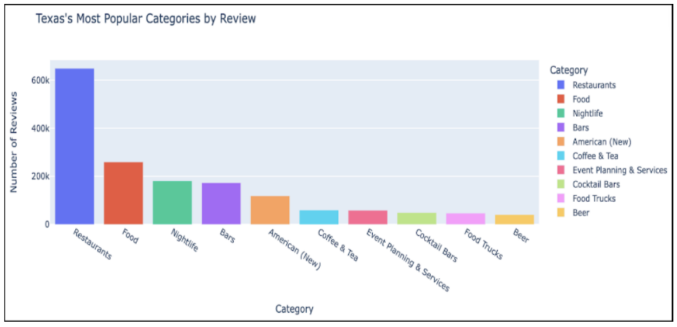


Figure 8

From our findings, we found that there wasn't a high amount of variance across all regions for the categories, with the most differences being just three. However, there were interesting unique values for cuisine depending on the state. For instance, Texas is widely renowned for their barbecue, but they also boast a large hispanic/latino population according to the U.S. Census Bureau [8]. In turn, it makes sense that BBQ and Tex-Mex are unique to Texas alone. Additionally, I found it interesting that Canada was not significantly different from the other regions, despite being on a whole different continent. Canada only had a few differences between the other regions, and still had American (Traditional) in its top ten.

The next thing we wanted to see was which cuisines/categories performed the best, as it might provide insight on what cuisines are preferred across the continent. To do this, we subsetted our main frame by only looking at reviews greater or equal to four stars. Once we did this, we created a frequency distribution like before to see what cuisines/categories were most common in the star

rating dataframe. To do this, we separated our dataframe by restaurants with at least 4 stars, and those with at most 2 stars. Then, we created a table showing the top 10 cuisines in each section by the most number of reviews, which we use as a measurement of popularity (Figures 9 and 10).

The Most Popular Cuisine for Restaurants with at Least 4 Stars

Cuisines	Number of Reviews
Breakfast & Brunch	479308
American (Traditional)	378790
Sandwiches	375204
Seafood	279596
Italian	229470
Pizza	215276
Mexican	213040
Japanese	195858
Vegetarian	172930
Salad	172163

Figure 9

The Most Popular Cuisine for Restaurants with at Most 2 Stars

Cuisines	Number of Reviews
Fast Food	45612
Burgers	33449
American (Traditional)	18662
Chicken Wings	15556
Breakfast & Brunch	13503
Pizza	12794
Sandwiches	10662
Mexican	10155
Tacos	4943
Tex-Mex	4834

Figure 10

We found that there is not a strong difference between cuisine types present, as there is a lot of overlap in cuisines between the two categories (low and high rating). Much of what we find in both groups are cuisines that are generally popular in the US and Canada such as sandwiches, American (Traditional), Pizza, and Mexican. Since there is a large amount of overlap, it is reasonable to conclude that there is not a strong difference between the types of cuisines present in high versus low star ratings, and other factors such as quality of food and service are more important. Interestingly, we find that fast food is associated with low star ratings, which is unsurprising as fast food is often associated with poor service and food quality. In addition, some cuisines which are often associated with fine dining, such as Italian and Japanese, are popular in the high rating category. Although some cuisines may be correlated with higher ratings, cuisine type is not one of the most important factors for star rating.

From our results, we observed that while certain regions do have their own styles of cuisine, there seems to be a general branch of cuisines. Like we saw before, Texas is a great example of personal cuisine style, as it has Tex-Mex and barbecue, which is quite region specific. Additionally, places with higher Asian populations, or that are along the coast are more likely to exhibit Asian and Seafood cuisine respectively. Nevertheless, the variance among the cuisines served across the continent was not extremely high. Nor were the categories of restaurants. Additionally, when we compared high rated cuisines/categories and low rated cuisines/categories with the overall population, we didn't see too many outliers between both of them. Thus we might infer that the cuisine being served at a restaurant does not make or break the overall rating of the restaurant. And if restaurateurs want to attract more customers, they should focus on aspects of their business that pertain to the experience they are selling.

Recommendation System

In order to create a recommendation system, we will be performing collaborative filtering, which works under the assumption that a user will have similar preferences to users who left similar ratings for a restaurant. In order to do collaborative filtering, we need to perform truncated SVD. The matrix that we will be decomposing is a sparse matrix with user ids, business ids, and customer stars. Truncated SVD produces a low-rank approximation of the original user-business-customer star matrix, and it allows us to specify the number of components (features) that are kept for our final recommendation system. SVD will produce 3 matrices from our user-business-customer star matrix , which we will refer to as matrices U, Σ, and V, respectively. Matrix U contains the relationship between users and the stars they left for restaurants, matrix Σ contains the top singular values extracted from our sparse matrix, and matrix V contains the relationship between businesses and the most important features that will be identified after performing SVD.

Before performing SVD, we filtered out any restaurants that were closed, as we wanted our recommendation system to have restaurants that are still open today. We combined the precovid and postcovid datasets, deleting any duplicate values that appeared. We also merged the user and business ids in order to give equal weight to the preferences of users before and after covid. We then subsetted the data, taking only the user stars, business ids, user ids, date, location, and restaurant name (this will be important later on when we create the actual recommendation function). Once we created our dataframe, we created a sparse matrix that contained restaurant names along the index, user ids along the columns, and customer stars as the values in the matrix, making sure to drop any duplicate values. The resulting matrix is 99.5% sparse, which is due to a lack of reviews for multiple restaurants by most users in our dataset. Our matrix that we decompose with truncated SVD contains 2,847 restaurants with 83,569 user ids. Now that we have our matrix, we can extract the most relevant relationships between restaurants and customer stars to create a recommendation system.

Performing truncated SVD allows us to choose the n largest singular values, which will determine the size of matrices U, Σ. In order to choose the number of singular values that we will keep in our recommendation system, we need to maximize the cumulative explained variance while simultaneously keeping the most important features from our original matrix.

The cumulative explained variance tells us how much of the variance in the data is captured by each of the singular values in our matrix Σ. The explained variance ratio tells us how much variance is captured by each of the top n singular values. In order to capture the most important features in our user-business-star matrix, we plotted the explained variance ratio for all of the components in the original dataframe (in this case, each component represents a linear combination of the customer ids and the customer stars). Figure 11 shows how much variance is explained by each restaurant in the

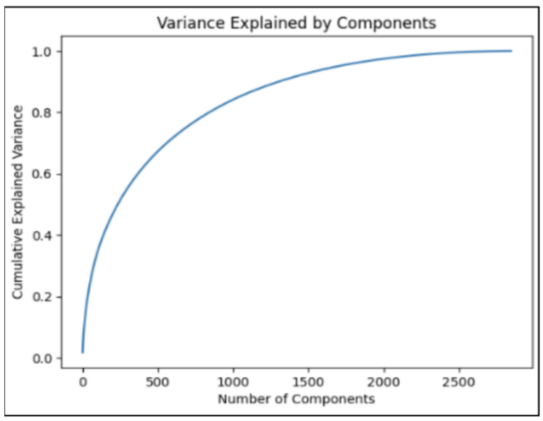


Figure 11

dataset. We want to try and maximize the cumulative explained variance (~ 80% of the total variance is considered a good amount when reducing the number of dimensions in our data [9]) while simultaneously trying to minimize the number of components in our final recommendation system. We ultimately ended up selecting the first 800 components to construct our correlation matrix, which will be used to produce recommendations based on the restaurant entered. This is a major reduction in the size of our original matrix, which had over 80,000 users.

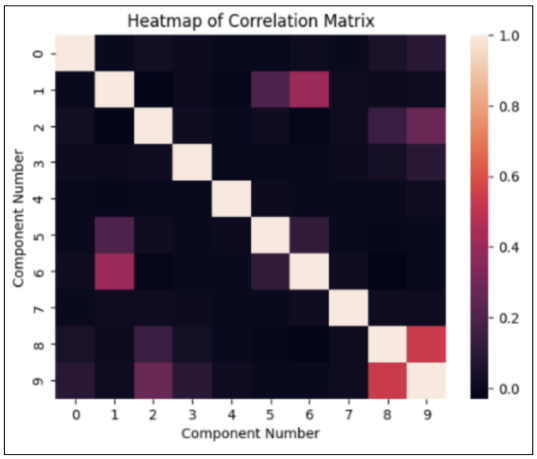


Figure 12

After performing truncated SVD, we can construct a correlation matrix by calculating the Pearson correlation coefficient for the different combinations of user stars obtained from SVD. Figure 12 shows the correlation coefficient for 10 components extracted from our original user-business-customer star matrix. Despite the sparsity of the data, we are still able to see some correlation between the user stars and the restaurant that received those star ratings.

```
top_restaurants(restaurant = "7-Eleven")
```

stars	name	address	city	state
4.5	Broad Street Paulie's	64 Broad St	Boston	MA
4.0	P & C Market & Caribbean Cuisine	35 Lebanon St	Malden	MA
3.0	7-Eleven	900 SE 164th Ave	Vancouver	WA
3.0	Domino's Pizza- Bee Cave	15500 W Hwy 71	Bee Cave	TX
3.0	Donair Affair	2540 Mary Hill Road, Suite 105	Port Coquitlam	BC
2.5	Dairy Queen Grill & Chill	613 Jq Adams St	Oregon City	OR
2.5	Kipos Pizzeria & Restaurant	106 Bunker Hill St	Charlestown	MA
2.0	Bud Light Port Paradise	1200 Red Cleveland Blvd	Sanford	FL
2.0	Everett House of Pizza	722 Broadway	Everett	MA
2.0	Papa Gino's	134 Nahatan St	Norwood	MA

Figure 13

In order to create a recommendation system, we created a function that finds the index of the restaurant selected and returns the top n restaurants with the highest correlation coefficient. Our function also returns pertinent information about the restaurants with the highest correlation coefficient, such as the overall number of stars and the location. Because this dataset consists of restaurants spread out across 11 cities, it is very important that we display the location so people who use our recommendation system can know how far away they are from potential suggestions for their next meal. Figure 13 depicts a screenshot of the recommendation function, along with the top 10 restaurants that had the highest Pearson correlation coefficient with the restaurant selected.

Conclusion:

In the modern world, the restaurant experience and business is an immutable part of life. Restaurants are the perfect place to embark on new experiences, and spend time with one another. Just like every business, a restaurateur should understand what they need to do to sell the best experience possible to new customers. From our study, we saw that service can highly influence the way customers perceive the restaurant experience. Additionally, this in turn can impact how many give their business to these establishments. Furthermore, we saw that the type of cuisine is not necessarily the most important aspect of a restaurant when looking at star ratings, or the overall picture. This in turn, implies that restaurateurs should focus again on selling the experience; as the style of cuisine does not diminish nor heighten a business' rating.

It is impossible to create a recommendation system that accommodates everyone's taste. However, we can try to do so with collaborative filtering. Although our recommendation system produces results that seem accurate, the sparsity of the data makes it difficult to produce recommendations that perfectly fit a person's taste. Additionally, restaurants with more reviews from customers will also appear more often when we are recommending restaurants to people. This factor played a role in influencing the correlation between customer stars and the restaurant they reviewed, but we were still able to make some informative recommendations.

Reference:

Whilst we still have a code appendix, we also thought it would be more accessible to provide a GitHub repository that contains all of the files we worked on. Here is the link to the repository: https://github.com/SeolHaeJuan/STA-141C-Restaurant-Project/

[1] Duyne, Allie Van. "60 Restaurant Industry Statistics and Trends for 2023 - Toasttab." *Toast*, https://pos.toasttab.com/blog/on-the-line/restaurant-management-statistics.

[2] Jay, Allan. "Number of Restaurants in the US 2022/2023: Statistics, Facts, and Trends." *Financesonline.com*, FinancesOnline.com, 15 Feb. 2023, https://financesonline.com/number-of-restaurants-in-the-us/.

[3] Syed, Fahad. "Restaurant Reviews." *Kaggle*, 29 Mar. 2022, https://www.kaggle.com/datasets/fahadsyed97/restaurant-reviews.

[4] Arumugam, Rohan. "Locations." 11 Mar. 2023 https://public.tableau.com/app/profile/rohan.arumugam/viz/locations_16786079033610/Sheet1

[5] Chang, Katie. "10 Reasons Why Portland Remains One of the Country's Best Food Cities." *Forbes*, Forbes Magazine, 1 Nov. 2021, https://www.forbes.com/sites/katiechang/2021/10/31/10-reasons-why-portland-remains-one-of-the-countrys-best-food-cities/?sh=39ecd107ee6e.

[6] *Population Density Data for Boston, MA - Open Data Network*. https://www.opendatanetwork.com/entity/1600000US2507000/Boston_MA/geographic.population.density?year=2018.

[7] Blog, Yelp – Official, and Yelp Inc. "The Complete Yelp Business Category List." *Yelp*, 25 Jan. 2023, https://blog.yelp.com/businesses/yelp_category_list/.

[8] *U.S. Census Bureau Quickfacts: Texas*. https://www.census.gov/quickfacts/fact/table/TX/PST045222.

[9] "PRINCIPAL COMPONENTS (PCA) AND EXPLORATORY FACTOR ANALYSIS (EFA) WITH SPSS." *OARC Stats*, https://stats.oarc.ucla.edu/spss/seminars/efa-spss/#:~:text=Some%20criteria%20say%20that%20the,about%20four%20to%20five%20components.

Further elaboration on SVD methodology:

The formula for SVD on a given matrix A is $A = U\Sigma V^T$. Matrix A, which represents our user-business-star matrix, is decomposed into , where A is a matrix of size n by m (for the purposes of our report, let n = 2847 and m = 83569). Matrix U is an n x m matrix that contains information not readily apparent about the users in our dataset. Matrix Σ is an m x m matrix that contains the top 800 singular values from the original matrix A (determined earlier by cumulative explained variance ratio), and matrix V^T is a m x n matrix that contains latent features about the restaurants in our dataset. Truncated SVD returns a low-rank approximation of the user-business-customer star matrix (A), which is important as it will reduce the number of dimensions in our data and extract latent features in our matrix A. The matrix returned from truncated SVD is equal to matrix $U\Sigma$, which means that we have a matrix that contains the latent features about the users in our dataset. With this information, we can compute the Pearson correlation coefficient to create a correlation matrix and create recommendations.

The Pearson correlation coefficient is defined as $\mathrm{Corr}(R, C) = \dfrac{\mathrm{Cov}(R, C)}{\sigma_R \sigma_C}$, where $\mathrm{Cov}(R, C)$ is the covariance between the rows (R) of a given matrix and the columns (C). The symbols σ_R and σ_C denote the standard deviation of R and C, respectively. We can use this formula to create a correlation matrix that stores the correlation coefficient between the rows and columns in matrix $U\Sigma$. By computing the correlation coefficient, we can see how related the stars that customers left at restaurants are with their preference of restaurant. We used this identity to create a function that, when given the name of a restaurant in our dataset, will return the top n restaurants with the highest correlation coefficient.

Appendix:

As all of our datasets consist of more than 400,000 observations, it is imperative for us to make use of various optimization techniques. As we touched upon before, it is unnecessary to use every single column value depending on the question we are answering. Thus, by subsetting the columns we want to read using pandas, we can cut the time to load a dataframe tenfold. For instance, the time to read both ‘precovid.csv’ and ‘postcovid.csv’ and then combine the two was approximately sixteen minutes of computation time. However, by only subsetting the columns we wanted to observe, we were able to read everything in one minute and nineteen seconds.

Given that our dataframe is more string focused, we found the use of *parallel computing* to be extremely suitable for our needs. To do so, we created temporary files that would be able to hold the desired function we needed. Then, when we wanted to start the computation, we would import the function from the temporary text file and run it from the file. We used this to be able to create eight dataframes simultaneously. This pool function was more effective for us than regular standard methods of multiprocessing, as we didn’t need to share more files with each other, and the function

was written within the script. To compare computation times, we saw that to manually compute over eight iterations, it took around five minutes. With parallel computing however, we cut that time to only around one minute and thirty seconds.

Although the matrix is very sparse, there were many observations within our matrix, causing it to take up a lot of memory storage. By using joblib's parallel_backend function, we were able to parallelize truncated SVD by conducting it across multiple processes. The optimization of this code allowed us to reduce the computation time of truncated SVD from 9 minutes thirty seconds to 1 minute 20 seconds, which was helpful for troubleshooting code and analyzing the dataset in a timely manner.

Code for Questions 1 and 2

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import nltk
import pprint
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from matplotlib.ticker import StrMethodFormatter
import seaborn as sns
import string
from collections import Counter
import csv
import datetime
import plotnine
from plotnine import *
import plotly.express as px
from plotly import graph_objs as go
pd.set_option('display.float_format', lambda x: '%.3f' % x)

post_covid = pd.read_csv("postcovid_reviews.csv")
pre_covid = pd.read_csv("precovid_reviews.csv")

pre_covid['Pre_covid'] = True
post_covid['Pre_covid'] = False
colnames = list(pre_covid)
reviews = pre_covid.merge(post_covid,on=colnames, how='outer')
reviews = reviews[reviews['is_open'] == True]
# creates plot to find number of reviews over time
reviews[['Date', 'Time']]= reviews['date_'].str.split(" +",expand = True)
reviews[['Year', 'Month', 'Day']] = reviews['Date'].str.split("-",expand = True)
reviews.drop(['date_', 'Date'],axis=1,inplace=True)

cnt = reviews.groupby('Year').size().rename('Count')
dates = reviews.drop_duplicates(subset='Year').merge(cnt, left_on='Year', right_index=True)
dates['Year'] = pd.to_numeric(dates['Year'])
dates = dates[~(dates['Year'] == 2021)]

(   ggplot(dates)
    + aes(x="Year", y="Count", group=1)
    + geom_point()
    + geom_line()
    + theme(axis_text_x=element_text(rotation=45, hjust=1))
    + plotnine.labels.ylab('Number of reviews')
)
```

```
#combining ABE/BC and WA/OR since reviews from these places are in cities very close together
reviews['state_'] = np.where(reviews['state_'] == 'ABE', 'BC', reviews['state_'])
reviews['state_'] = np.where(reviews['state_'] == 'WA', 'OR', reviews['state_'])
reviews = reviews.loc[reviews['state_'].isin(['GA', 'BC', 'TX', 'MA', 'OR', 'FL', 'OH', 'CO'])]

#creates plot for regions with most restaurants
unique_restaraunt_state = reviews.drop_duplicates(['business_id']).groupby('state_')['business_id'].count()
                          .reset_index().sort_values('business_id', ascending = False)
unique_restaraunt_state = unique_restaraunt_state[unique_restaraunt_state['business_id'] > 2]

fig = px.bar(unique_restaraunt_state, x='state_', y='business_id', color = 'state_', title = "Regions With
Most Restaurants", labels={
                     "state_": "State",
                     "business_id": "Count",
                 },)
fig.show()

# creates plot for 12 most popular restaurants with at least 500 reviews
most_popular = reviews[['name', 'state_','review_count', 'customer_stars']]
popularity = most_popular.groupby('name', as_index=False).agg({'review_count':'max', 'customer_stars':'mean',
'state_':'max'})
most_popular = popularity[popularity['review_count'] >= 500].sort_values('customer_stars', ascending =
False).head(12)

fig = px.bar(most_popular, x='name', y='customer_stars', color = 'state_', title = "The Highest Rated
Restaurants")
fig.update_layout(barmode='stack', xaxis={'categoryorder':'total descending'})
fig.update_layout(yaxis=dict(range=[4,4.9]))
fig.show()

# creates plot for 12 least popular restaurants with at least 500 reviews
least_popular = popularity[popularity['review_count'] >= 500].sort_values('customer_stars', ascending =
True).head(12)

fig = px.bar(least_popular, x='name', y='customer_stars', color = 'state_', title = "The Lowest Rated
Restaurants")
fig.update_layout(barmode='stack', xaxis={'categoryorder':'total descending'})
fig.update_layout(yaxis=dict(range=[1,3]))
fig.show()

# creates plot for states with highest rated restaurants
best_state = reviews[['state_', 'review_count', 'customer_stars']]
best_state = best_state.groupby('state_', as_index=False).agg({'review_count':'max',
'customer_stars':'mean'})
best_state = best_state[best_state['review_count'] > 28].sort_values('customer_stars', ascending =
False).head(12)
```

```
fig = px.bar(best_state, x='state_', y='customer_stars', color = 'state_', title = "Regions With Highest
Rated Restaurants", labels={
                     "state_": "State",
                     "customer_stars": "Stars",
                 },)
fig.update_layout(yaxis=dict(range=[3.5,3.92]))
fig.show()



# Question 1: How does customer sentiment correlate with star ratings?

vader = SentimentIntensityAnalyzer() # creates an instance of the vader object
%%capture --no-display

f = lambda text: vader.polarity_scores(text)['compound'] # lambda function to apply vader model on 'text'
subset_reviews['compound'] = reviews['text_'].apply(f) # creates new column for compound score
ax = sns.barplot(data=reviews, x='customer_stars', y='compound') # creates bar plot to visualize
ax.set_title('Compound Score by Customer Star Review')
plt.savefig('Compound_Score_by_Customer.png', bbox_inches='tight')
plt.show()

# Question 2: What words appear the most in positive/negative reviews?

positive_reviews = reviews[reviews['customer_stars'] >= 4] # subsets by finding positive ratings
negative_reviews = reviews[reviews['customer_stars'] <= 2] # subsets by finding negative ratings

gist_file = open("stopwords.txt", "r")  # creates function to open stop_words file
try:
    content = gist_file.read()
    stopwords = content.split(",")
    stopwords=[i.replace('"',"").strip() for i in stopwords]
finally:
    gist_file.close()

def common_words(df, column, punctuations, stop_words): # creates function to find word frequency in reviews
    check = " ".join(df[column].tolist())
    check = check.replace('\r', '').replace('\n', '')

    filtered_sentence = []
    words = word_tokenize(check)

    for w in words:
        if w.lower() not in stop_words:
```

```
            filtered_sentence.append(w)

    filtered_sentence = [i for i in filtered_sentence if (i not in punctuations) and (len(i)>2)]
    filtered_sentence = [w for w in filtered_sentence if all(c not in w for c in "'`")]
    common = Counter(filtered_sentence).most_common(30)
    return(common)

punctuations = list(string.punctuation) # finds types of punctuations and adds other punctuation to list
punctuations.append("'`")
punctuations.append('..')
punctuations.append("``")
punctuations.append('...')
punctuations.append('--')

positive_reviews = common_words(positive_reviews, 'text_', punctuations, stopwords) # calls function
negative_reviews = common_words(negative_reviews, 'text_', punctuations, stopwords) # calls function

# creates frequency plot for most common words in positive reviews
x_positive = [x for x,y in positive_reviews]
y_positive = [y for x,y in positive_reviews]
ax = sns.barplot(x=y_positive, y=x_positive)
ax.set_title('Most Common Words in Reviews With 4 or 5 Stars')
ax.set_xlabel('Frequency')
ax.set_ylabel('Words')
plt.show()

# creates frequency plot for most common words in negative reviews
x_negative = [x for x,y in negative_reviews]
y_negative = [y for x,y in negative_reviews]
ax = sns.barplot(x=y_negative, y=x_negative)
ax.set_title('Most Common Words in Reviews With 1 or 2 Stars')
ax.set_xlabel('Frequency')
ax.set_ylabel('Words')
plt.show()

# imports (packages, functions, etc.)

### Vader
from nltk.sentiment.vader import SentimentIntensityAnalyzer

### Tokenize
from nltk.tokenize import word_tokenize, sent_tokenize
from collections import Counter

### Punctuation list
import string
punctuations = list(string.punctuation)
```

```
punctuations.append("''")
punctuations.append('..')
punctuations.append("``")
punctuations.append('...')
punctuations.append('--')

### Stopwords (from https://gist.github.com/sebleier/554280?permalink_comment_id=3431590#gistcomment-3431590)
gist_file = open("stopwords.txt", "r")
try:
    content = gist_file.read()
    stopwords = content.split(",")
    stopwords=[i.replace('"',"").strip() for i in stopwords]
finally:
    gist_file.close()

### Get common words function
def common_words(df, column, punctuations, stop_words):
    check = " ".join(df[column].tolist())
    check = check.replace('\r', '').replace('\n', '')

    filtered_sentence = []
    words = word_tokenize(check)

    for w in words:
        if w.lower() not in stop_words:
            filtered_sentence.append(w)

    filtered_sentence = [i for i in filtered_sentence if (i not in punctuations) and (len(i)>2)]
    filtered_sentence = [w for w in filtered_sentence if all(c not in w for c in "''")]
    common = Counter(filtered_sentence).most_common(30)
    return(common)

service_words = ['service', 'server', 'staff', 'waiter', 'waitress']

# Find the sentences that talk about service
service_sentences = []
for i in range(len(reviews)):
    for word in service_words:
        if word in reviews['text_'][i]:
            sent = ([sentence + '.' for sentence in reviews['text_'][i].split('.') if word in sentence])
            service_sentences.append(sent)
        break

# Get rid of length 0 sentences
service_sentences = [item for item in service_sentences if len(item) == 1]
service_sentences = [sent for sublist in service_sentences for sent in sublist] # unnest everything from list
```

```
# Get vader compound score of each of the sentences talking about service
service_scores = [vader.polarity_scores(sentence)['compound'] for sentence in service_sentences]

# Create dataframe with columns of sentences talking about service and the sentiment score
scores_dict = {"text": service_sentences, "sentiment_score": service_scores}
df_service = pd.DataFrame(scores_dict)
df_service

# Conditions for if service is good, bad, or neutral
conditions = [
    (df_service['sentiment_score'] > 0),
    (df_service['sentiment_score'] < 0),
    (df_service['sentiment_score'] == 0)
]

values = [1, 0, -1] # 1 = positive, 0 = negative, -1 = neutral

# Create column to check if service is good (positive)
df_service['is_positive'] = np.select(conditions, values)

positive_service = df_service[df_service['is_positive'] == 1]

common_words(positive_service, 'text', punctuations, stopwords)

service_dict = {'customer_stars': [], 'sentence': []}

for i in range(len(reviews)):
    sent = ([sentence + '.' for sentence in str(reviews.iloc[i]['text_']).split('.')
             if ('service' or 'server' or 'staff' or 'waiter' or 'waitress') in sentence])
    service_dict['customer_stars'].append(reviews.iloc[i]['customer_stars'])
    service_dict['sentence'].append(sent)

service_df = pd.DataFrame(service_dict)
service_df = service_df.loc[service_df['sentence'].str.len() == 1] # get non-empty values
service_df = service_df.explode('sentence') # remove listing from each sentence

service_df['sentiment_scores'] = service_df.apply(lambda x: vader.polarity_scores(x['sentence'])['compound'],
axis = 1)

ax = sns.barplot(data=service_df, x='customer_stars', y='sentiment_scores')
ax.set_title('Sentiment Score About Service by Customer Star Review')
plt.show()
```

Code for Question 3

```
# Packages Used

import pandas as pd
import numpy as np
import nltk
from bs4 import BeautifulSoup
import requests
from dash import Dash, html, Input, Output, dash_table
import plotly.express as px
import plotly.io as pio
import plotly.graph_objects as go
from wordcloud import WordCloud, STOPWORDS
import matplotlib.pyplot as plt
from timeit import default_timer as timer
from PIL import Image
from wordcloud import ImageColorGenerator

# Running it like this takes way too long
start = timer()
precovid = pd.read_csv("/Users/fish/Documents/STA 141C/STA 141C Project Code/precovid_reviews.csv")
postcovid = pd.read_csv('/Users/fish/Documents/STA 141C/STA 141C Project Code/postcovid_reviews.csv')
end = timer()
print("The time it took to load the entire dataset was", ((end-start)/60), "minutes.")

# Now we need to combine them
start = timer()
combined = pd.concat([precovid, postcovid])
reviews = combined[combined['is_open'] == 1]
end = timer()
print("The time it took to combine both datasets was", ((end - start)/60), "minutes.")

# Cuts time down by ~7 whole minutes
start = timer()
cols = ['business_id', 'name', 'is_open', 'categories', 'review_count', 'state_', 'stars']

pre = pd.read_csv("/Users/fish/Documents/STA 141C/STA 141C Project Code/precovid_reviews.csv", usecols=cols)
post = pd.read_csv("/Users/fish/Documents/STA 141C/STA 141C Project Code/postcovid_reviews.csv",
usecols=cols)
# Cuts time down by more than 7 minutes
comb = pd.concat([pre, post])
rev = comb[comb['is_open'] == 1]
end = timer()
print("The time to subset the dataframe was:", (end-start), "seconds")
```

```
# Getting the total number of observations for each dataframe

print(pre.shape)
print(post.shape)
print(rev.shape)

# Web scraping cuisines list to use as a way to filter cuisine vs establishment
categories_url = 'https://blog.yelp.com/businesses/yelp_category_list/'
header = {'user-agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/110.0.0.0 Safari/537.36'}
page = requests.get(categories_url, headers = header)

html = BeautifulSoup(page.content, 'html.parser')
uls = html.find_all('ul', class_ = None)     # get all <ul> tags
categories_html = uls[90]     # get <ul> containing the restaurant categories

# Get all categories into list format
cuisines = [li.text.split('\n', 1)[0] for li in categories_html.findAll('li')]

# Most Common Cuisine Across Entire Dataset
# Converting to just a list of the words in the category column
review_cat = reviews.categories
review_cat = reviews.categories.apply(lambda x:x.split(", "))
review_cat = review_cat.tolist()

# because it is a concated list, i flatten so that it's all just one giant list, which is the format we need
for FreqDist
flatword = [item for sublist in review_cat for item in sublist if item in cuisines]

# Create a frequency distribution
fq = nltk.FreqDist(flatword)

# Take the top ten cuisines
review_cuis = pd.DataFrame(fq.most_common(10), columns=['Cuisines', 'Number of Reviews'])

# Converting to just a list of the words in the category column

# Testing to see if we will receive the same output even though the data was procured in a different manner

start = timer()
revs_cat = rev.categories
revs_cat = rev.categories.apply(lambda x:x.split(", "))
revs_cat = revs_cat.tolist()

# because it is a concated list, i flatten so that it's all just one giant list, which is the format we need
for FreqDist
flatword = [item for sublist in revs_cat for item in sublist if item in cuisines]
```

```
# Create a frequency distribution
fq = nltk.FreqDist(flatword)

# Take the top ten cuisines
revs_cuis = pd.DataFrame(fq.most_common(10), columns=['Cuisines', 'Number of Reviews'])
end = timer()

# For wordcloud on front page

rev_cfive = pd.DataFrame(fq.most_common(100), columns = ['Cuisines', 'Number of Reviews'])
text = rev_cfive['Cuisines'].values

# Wordcloud function using a mask of a waiter holding a tray

# https://stackoverflow.com/questions/43606339/generate-word-cloud-from-single-column-pandas-dataframe
def cloud(data, mask=None):
    cloud = WordCloud(scale=4,
                      max_words=150,
                      colormap='RdYlGn',
                      mask=mask,
                      background_color='black',
                      stopwords=stopwords,
                      collocations=True).generate(str(text))
    plt.figure(figsize=(10,10))
    plt.imshow(cloud)
    plt.axis('off')
    plt.show()

# Use the function with the rome_corpus and our mask to create word cloud
cloud(text, mask=np.array(Image.open("waiter.jpg")))

# Plot function for cuisine
def cuis_plot(df, given):
    fig = px.bar(df, x="Cuisines", y="Number of Reviews", color="Cuisines", title= given)
    fig.show()

cuis_plot(review_cuis, 'Most Common Cuisine Types in The Entire Dataset (Reviews)')
cuis_plot(revs_cuis, 'Most Common Cuisine Types in the Entire Dataset (Revs)')
old = reviews
reviews = rev

display(old.head(1))
display(reviews.head(1))

# generating the interactive table
```

```
fig = go.Figure(data=[go.Table(
    columnwidth = [5,5],
    header=dict(values=list(review_cuis.columns),
                fill_color='teal',
                align='center'),
    cells=dict(values=[review_cuis['Cuisines'], review_cuis['Number of Reviews']],
               fill_color='lavender',
               align='center'))
])
fig.update_layout(
    title_text = "The Total Most Popular Cuisines by Reviews",
    title_font_size=30,
    title_x=0.5,
    font_family="Times New Roman",
    font_color="black",
    title_font_family="Times New Roman",
    title_font_color="black"
)
fig.update_traces(cells_font=dict(size = 12))
fig.show()

 Making a wordcloud for cuisine

revs_cuis_50 = pd.DataFrame(fq.most_common(50), columns=['Cuisines', 'Number of Reviews'])
comment_words = ''
stopwords = set(STOPWORDS)

# iterate through the csv file
for val in revs_cuis_50.Cuisines:

    # typecaste each val to string
    val = str(val)

    # split the value
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    comment_words += " ".join(tokens)+" "

wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(comment_words)
```

```
# plot the WordCloud image
plt.figure(figsize = (4, 4), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()

## **Most Common Establishment Type Across Entire Dataset**

# all we have to do to get the most common establishment type, is do item not in cuisines
flatword = [item for sublist in review_cat for item in sublist if item not in cuisines]

# Create a frequency distribution
fq = nltk.FreqDist(flatword)

# Take the top ten cuisines
review_est = pd.DataFrame(fq.most_common(10), columns=['Category', 'Number of Reviews'])

# Plot function for establishment type
def est_plot(df, given):
    fig = px.bar(df, x = "Category", y = "Number of Reviews", color = "Category", title = given)
    fig.show()

est_plot(review_est, 'The Most Common Establishment Type Across The Entire Dataset')

# generating the interactive table

fig = go.Figure(data=[go.Table(
    columnwidth = [5,5],
    header=dict(values=list(review_est.columns),
                fill_color='lavenderblush',
                align='center'),
    cells=dict(values=[review_est['Category'], review_est['Number of Reviews']],
               fill_color='lavender',
               align='center'))
])
fig.update_layout(
    title_text = "The Total Most Popular Categories by Reviews",
    title_font_size=30,
    title_x=0.5,
    font_family="Times New Roman",
    font_color="black",
    title_font_family="Times New Roman",
    title_font_color="black"
)
fig.update_traces(cells_font=dict(size = 12))
```

```
fig.show()

revs_est_50 = pd.DataFrame(fq.most_common(50), columns=['Category', 'Number of Reviews'])
comment_words = ''
stopwords = set(STOPWORDS)

# iterate through the csv file
for val in revs_est_50.Category:

    # typecaste each val to string
    val = str(val)

    # split the value
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    comment_words += " ".join(tokens)+" "

wordcloud = WordCloud(width = 800, height = 800,
                background_color ='salmon',
                stopwords = stopwords,
                min_font_size = 10).generate(comment_words)

# plot the WordCloud image
plt.figure(figsize = (4, 4), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()

## **Multiprocessing**

bc_reviews = reviews[(reviews['state_'] == 'BC') | (reviews['state_'] == 'ABE')]
co_reviews = reviews[reviews['state_'] == 'CO']
fl_reviews = reviews[reviews['state_'] == 'FL']
ga_reviews = reviews[reviews['state_'] == 'GA']
ma_reviews = reviews[reviews['state_'] == 'MA']
oh_reviews = reviews[reviews['state_'] == 'OH']
or_reviews = reviews[(reviews['state_'] == 'OR') | (reviews['state_'] == 'WA')]
tx_reviews = reviews[reviews['state_'] == 'TX']

# Converting the reviews to csv for multiprocessing
```

```
bc_reviews.to_csv('bc_reviews.csv')
co_reviews.to_csv('co_reviews.csv')
fl_reviews.to_csv('fl_reviews.csv')
ga_reviews.to_csv('ga_reviews.csv')
ma_reviews.to_csv('ma_reviews.csv')
oh_reviews.to_csv('oh_reviews.csv')
or_reviews.to_csv('or_reviews.csv')
tx_reviews.to_csv('tx_reviews.csv')

from multiprocessing import Pool
from functools import partial
import inspect

#
https://stackoverflow.com/questions/47313732/jupyter-notebook-never-finishes-processing-using-multiprocessing
-python-3

# We are making a tempoary file in order to call our function later for parallel computing
def parallel_task(func, iterable):

    with open(f'./tmp_func.py', 'w') as file:
        file.write(inspect.getsource(func).replace(func.__name__, "task"))

    from tmp_func import task

    if __name__ == '__main__':
        func = partial(task)
        pool = Pool(processes=101)
        res = pool.map(func, iterable)
        pool.close()
        return res
    else:
        raise "Not in Jupyter Notebook"

# Multiprocessing for the Cuisines

start = timer()
frame_bank = []

def multi_norm(q):

    # Packages must recall everything as we are treating this like it is a brand new text file
    import pandas as pd
    import numpy as np
    import nltk
    from bs4 import BeautifulSoup
    import requests
```

```
    # Web scraping cuisines list to use as a way to filter cuisine vs establishment
    categories_url = 'https://blog.yelp.com/businesses/yelp_category_list/'
    header = {'user-agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/110.0.0.0 Safari/537.36'}
    page = requests.get(categories_url, headers = header)

    html = BeautifulSoup(page.content, 'html.parser')
    uls = html.find_all('ul', class_ = None)     # get all <ul> tags
    categories_html = uls[90]     # get <ul> containing the restaurant categories

    # Get all categories into list format
    cuisines = [li.text.split('\n', 1)[0] for li in categories_html.findAll('li')]

    # List of dataframes we have already subsetted that just need to be called. Making a list of strings to
access
    names = ['bc_reviews.csv', 'co_reviews.csv', 'fl_reviews.csv', 'ga_reviews.csv', 'ma_reviews.csv',
'oh_reviews.csv', 'or_reviews.csv', 'tx_reviews.csv']

    # access the ith dataframe
    frame = pd.read_csv(names[q])
    frame_words = frame.categories
    frame_words = frame.categories.apply(lambda x: x.split(", "))
    frame_words = frame_words.tolist()

    flatword = [item for sublist in frame_words for item in sublist if item in cuisines]

    fq = nltk.FreqDist(flatword)

    cuis_frame = pd.DataFrame(fq.most_common(10), columns=['Cuisines', 'Number of Reviews'])

    return cuis_frame

for res in parallel_task(multi_norm, range(8)):
    frame_bank.append(res)

end = timer()

print("The time it took to construct all of the dataframes for cuisines was", (end - start), "seconds")

# Multiprocessing for category

start = timer()

cat_bank = []

def kat(q):
```

```
    # Packages must recall everything as we are treating this like it is a brand new text file
    import pandas as pd
    import numpy as np
    import nltk
    from bs4 import BeautifulSoup
    import requests

    # Web scraping cuisines list to use as a way to filter cuisine vs establishment
    categories_url = 'https://blog.yelp.com/businesses/yelp_category_list/'
    header = {'user-agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/110.0.0.0 Safari/537.36'}
    page = requests.get(categories_url, headers = header)

    html = BeautifulSoup(page.content, 'html.parser')
    uls = html.find_all('ul', class_ = None)     # get all <ul> tags
    categories_html = uls[90]     # get <ul> containing the restaurant categories

    # Get all categories into list format
    cuisines = [li.text.split('\n', 1)[0] for li in categories_html.findAll('li')]

    # List of dataframes we have already subsetted that just need to be called. Making a list of strings to
access
    names = ['bc_reviews.csv', 'co_reviews.csv', 'fl_reviews.csv', 'ga_reviews.csv', 'ma_reviews.csv',
'oh_reviews.csv', 'or_reviews.csv', 'tx_reviews.csv']

    # access the ith dataframe
    frame = pd.read_csv(names[q])
    frame_words = frame.categories
    frame_words = frame.categories.apply(lambda x: x.split(", "))
    frame_words = frame_words.tolist()

    flatword = [item for sublist in frame_words for item in sublist if item not in cuisines]

    fq = nltk.FreqDist(flatword)

    cat_frame = pd.DataFrame(fq.most_common(10), columns=['Category', 'Number of Reviews'])

    return cat_frame

for res in parallel_task(kat, range(8)):
    cat_bank.append(res)

end = timer()

print("The time it took to construct all of the dataframes for cuisines was", (end - start), "seconds")
```

```
bc_food, co_food, fl_food, ga_food, ma_food, oh_food, or_food, tx_food = frame_bank
bc_rest, co_rest, fl_rest, ga_rest, ma_rest, oh_rest, or_rest, tx_rest = cat_bank

## **Most Common Cuisine For Each Region by Reviews | The Standard Way**

# Vancouver, BC area

# Subsetting just cuisine
bc_words = bc_reviews.categories
bc_words = bc_reviews.categories.apply(lambda x: x.split(", "))
bc_words = bc_words.tolist()

# because it is a concated list, i flatten so that it's all just one giant list, which is the format we need
for FreqDist
flatword = [item for sublist in bc_words for item in sublist if item in cuisines]

# Create a frequency distribution
fq = nltk.FreqDist(flatword)

# Take the top ten cuisines
bc_cuis = pd.DataFrame(fq.most_common(10), columns=['Cuisines', 'Number of Reviews'])

# Boulder, CO area

# Subsetting just cuisine
co_words = co_reviews.categories
co_words = co_reviews.categories.apply(lambda x: x.split(", "))
co_words = co_words.tolist()

# because it is a concated list, i flatten so that it's all just one giant list, which is the format we need
for FreqDist
flatword = [item for sublist in co_words for item in sublist if item in cuisines]

# Create a frequency distribution
fq = nltk.FreqDist(flatword)

# Take the top ten cuisines
co_cuis = pd.DataFrame(fq.most_common(10), columns=['Cuisines', 'Number of Reviews'])
# Orlando, FL area

# Subsetting just cuisine
fl_words = fl_reviews.categories
fl_words = fl_reviews.categories.apply(lambda x: x.split(", "))
fl_words = fl_words.tolist()

# because it is a concated list, i flatten so that it's all just one giant list, which is the format we need
for FreqDist
```

```
flatword = [item for sublist in fl_words for item in sublist if item in cuisines]

# Create a frequency distribution
fq = nltk.FreqDist(flatword)

# Take the top ten cuisines
fl_cuis = pd.DataFrame(fq.most_common(10), columns=['Cuisines', 'Number of Reviews'])

# Atlanta, GA area

# Subsetting just cuisine
ga_words = ga_reviews.categories
ga_words = ga_reviews.categories.apply(lambda x: x.split(", "))
ga_words = ga_words.tolist()

# because it is a concated list, i flatten so that it's all just one giant list, which is the format we need
for FreqDist
flatword = [item for sublist in ga_words for item in sublist if item in cuisines]

# Create a frequency distribution
fq = nltk.FreqDist(flatword)

# Take the top ten cuisines
ga_cuis = pd.DataFrame(fq.most_common(10), columns=['Cuisines', 'Number of Reviews'])

# Boston, MA area

# Subsetting just cuisine
ma_words = ma_reviews.categories
ma_words = ma_reviews.categories.apply(lambda x: x.split(", "))
ma_words = ma_words.tolist()

# because it is a concated list, i flatten so that it's all just one giant list, which is the format we need
for FreqDist
flatword = [item for sublist in ma_words for item in sublist if item in cuisines]

# Create a frequency distribution
fq = nltk.FreqDist(flatword)

# Take the top ten cuisines
ma_cuis = pd.DataFrame(fq.most_common(10), columns=['Cuisines', 'Number of Reviews'])

# Columbus, OH area

# Subsetting just cuisine
oh_words = oh_reviews.categories
oh_words = oh_reviews.categories.apply(lambda x: x.split(", "))
```

```
oh_words = oh_words.tolist()

# because it is a concated list, i flatten so that it's all just one giant list, which is the format we need for FreqDist
flatword = [item for sublist in oh_words for item in sublist if item in cuisines]

# Create a frequency distribution
fq = nltk.FreqDist(flatword)

# Take the top ten cuisines
oh_cuis = pd.DataFrame(fq.most_common(10), columns=['Cuisines', 'Number of Reviews'])

# Portland, OR area

# Subsetting just cuisine
or_words = or_reviews.categories
or_words = or_reviews.categories.apply(lambda x: x.split(", "))
or_words = or_words.tolist()

# because it is a concated list, i flatten so that it's all just one giant list, which is the format we need for FreqDist
flatword = [item for sublist in or_words for item in sublist if item in cuisines]

# Create a frequency distribution
fq = nltk.FreqDist(flatword)

# Take the top ten cuisines
or_cuis = pd.DataFrame(fq.most_common(10), columns=['Cuisines', 'Number of Reviews'])

# Austin, TX area

# Subsetting just cuisine
tx_words = tx_reviews.categories
tx_words = tx_reviews.categories.apply(lambda x: x.split(", "))
tx_words = tx_words.tolist()

# because it is a concated list, i flatten so that it's all just one giant list, which is the format we need for FreqDist
flatword = [item for sublist in tx_words for item in sublist if item in cuisines]

# Create a frequency distribution
fq = nltk.FreqDist(flatword)

# Take the top ten cuisines
tx_cuis = pd.DataFrame(fq.most_common(10), columns=['Cuisines', 'Number of Reviews'])
cuis_plot(tx_cuis, "Texas's Most Popular Cuisines by Review")
```

```
## **Most Common Category For Each Region by Reviews**
# Canada

# Don't need to redo all of the defining variables because it's all the same regardless
# because it is a concated list, i flatten so that it's all just one giant list, which is the format we need
for FreqDist
# the main thing is just respecifying to not in, that's all
flatword = [item for sublist in bc_words for item in sublist if item not in cuisines]
# Create a frequency distribution
fq = nltk.FreqDist(flatword)
# Take the top ten categories
bc_est = pd.DataFrame(fq.most_common(10), columns=['Category', 'Number of Reviews'])

# Colorado

flatword = [item for sublist in co_words for item in sublist if item not in cuisines]

# fq dist
fq = nltk.FreqDist(flatword)

# Top ten categories
co_est = pd.DataFrame(fq.most_common(10), columns = ['Category', 'Number of Reviews'])

# Florida

flatword = [item for sublist in fl_words for item in sublist if item not in cuisines]

# fq dist
fq = nltk.FreqDist(flatword)

# Top ten categories
fl_est = pd.DataFrame(fq.most_common(10), columns = ['Category', 'Number of Reviews'])

# Georgia

flatword = [item for sublist in ga_words for item in sublist if item not in cuisines]

# fq dist
fq = nltk.FreqDist(flatword)

# Top ten categories
ga_est = pd.DataFrame(fq.most_common(10), columns = ['Category', 'Number of Reviews'])

# Massachusetts

flatword = [item for sublist in ma_words for item in sublist if item not in cuisines]
```

```
# fq dist
fq = nltk.FreqDist(flatword)

# Top ten categories
ma_est = pd.DataFrame(fq.most_common(10), columns = ['Category', 'Number of Reviews'])

# Ohio

flatword = [item for sublist in oh_words for item in sublist if item not in cuisines]

# fq dist
fq = nltk.FreqDist(flatword)

# Top ten categories
oh_est = pd.DataFrame(fq.most_common(10), columns = ['Category', 'Number of Reviews'])

# Oregon

flatword = [item for sublist in or_words for item in sublist if item not in cuisines]

# fq dist
fq = nltk.FreqDist(flatword)

# Top ten categories
or_est = pd.DataFrame(fq.most_common(10), columns = ['Category', 'Number of Reviews'])

# Texas

flatword = [item for sublist in tx_words for item in sublist if item not in cuisines]

# fq dist
fq = nltk.FreqDist(flatword)

# Top ten categories
tx_est = pd.DataFrame(fq.most_common(10), columns = ['Category', 'Number of Reviews'])

# Output plot

est_plot(tx_est, "Texas's Most Popular Categories by Review")

## **Comparing the Multiprocessing Way vs the Manual Way**

# Comparing if the dataframes are equivalent to eachother
def comp(df1, df2):
    if df1.equals(df2):
        print("The two dataframes are equal!")
    else:
```

```
        print("The two dataframes are not equal.")

comp(bc_food, bc_cuis)

# From the result above, we can see that our multiprocessing function gave us the same result as the manual
version. We also saw that we were able to do the calculation much faster than before, as well as more
efficiently. Thus, the usage of parallel computing is much better to use for every aspect of the problem.

# Difference between Each Region and the overall | Cuisine

# Canada
bcrcdiff = bc_food.merge(review_cuis.drop_duplicates(), on=['Cuisines'],
                   how='left', indicator=True)

print("The number of cuisines differing between Canada and the total population is",
len(bcrcdiff[bcrcdiff["_merge"]=='left_only']))

# Colorado
corcdiff = co_food.merge(review_cuis.drop_duplicates(), on=['Cuisines'],
                   how='left', indicator=True)

print("The number of cuisines differing between Colorado and the total population is",
len(corcdiff[corcdiff["_merge"]=='left_only']))

# Florida
flrcdiff = fl_food.merge(review_cuis.drop_duplicates(), on=['Cuisines'],
                   how='left', indicator=True)

print("The number of cuisines differing between Florida and the total population is",
len(flrcdiff[flrcdiff["_merge"]=='left_only']))

# Georgia
garcdiff = ga_food.merge(review_cuis.drop_duplicates(), on=['Cuisines'],
                   how='left', indicator=True)

print("The number of cuisines differing between Georgia and the total population is",
len(garcdiff[garcdiff["_merge"]=='left_only']))

# Massachusetts
marcdiff = ma_food.merge(review_cuis.drop_duplicates(), on=['Cuisines'],
                   how='left', indicator=True)

print("The number of cuisines differing between Massachusetts and the total population is",
len(marcdiff[marcdiff["_merge"]=='left_only']))

# Ohio
ohrcdiff = oh_food.merge(review_cuis.drop_duplicates(), on=['Cuisines'],
```

```
                  how='left', indicator=True)

print("The number of cuisines differing between Ohio and the total population is",
len(ohrcdiff[ohrcdiff["_merge"]=='left_only']))

# Oregon
orrcdiff = or_food.merge(review_cuis.drop_duplicates(), on=['Cuisines'],
                  how='left', indicator=True)

print("The number of cuisines differing between Oregon and the total population is",
len(orrcdiff[orrcdiff["_merge"]=='left_only']))

# Texas
txrcdiff = tx_food.merge(review_cuis.drop_duplicates(), on=['Cuisines'],
                  how='left', indicator=True)

print("The number of cuisines differing between Texas and the total population is",
len(txrcdiff[txrcdiff["_merge"]=='left_only']))

comp(bc_rest, bc_est)

# Difference between Each Region and the overall | Category

# Canada
bcrediff = bc_rest.merge(review_est.drop_duplicates(), on=['Category'],
                  how='left', indicator=True)

print("The number of establishments differing between Canada and the total population is",
len(bcrediff[bcrediff["_merge"]=='left_only']))

# Colorado
corediff = co_rest.merge(review_est.drop_duplicates(), on=['Category'],
                  how='left', indicator=True)

print("The number of establishments differing between Colorado and the total population is",
len(corediff[corediff["_merge"]=='left_only']))

# Florida
flrediff = fl_rest.merge(review_est.drop_duplicates(), on=['Category'],
                  how='left', indicator=True)

print("The number of establishments differing between Florida and the total population is",
len(flrediff[flrediff["_merge"]=='left_only']))

# Georgia
garediff = ga_rest.merge(review_est.drop_duplicates(), on=['Category'],
                  how='left', indicator=True)
```

```
print("The number of establishments differing between Georgia and the total population is",
len(garediff[garediff["_merge"]=='left_only']))

# Massachusetts
marediff = ma_rest.merge(review_est.drop_duplicates(), on=['Category'],
                   how='left', indicator=True)

print("The number of establishments differing between Massachusetts and the total population is",
len(marediff[marediff["_merge"]=='left_only']))

# Ohio
ohrediff = oh_rest.merge(review_est.drop_duplicates(), on=['Category'],
                   how='left', indicator=True)

print("The number of establishments differing between Ohio and the total population is",
len(ohrediff[ohrediff["_merge"]=='left_only']))

# Oregon
orrediff = or_rest.merge(review_est.drop_duplicates(), on=['Category'],
                   how='left', indicator=True)

print("The number of establishments differing between Oregon and the total population is",
len(orrediff[orrediff["_merge"]=='left_only']))

# Texas
txrediff = tx_rest.merge(review_est.drop_duplicates(), on=['Category'],
                   how='left', indicator=True)

print("The number of establishments differing between Texas and the total population is",
len(txrediff[txrediff["_merge"]=='left_only']))

## **Subset to Only Reviews With 4 Stars or Above**

stars = reviews[reviews['stars'] >= 4.0]
stars.head(10)

# Getting the most common cuisine for restaurants over 4 Stars

star_cuis = stars.categories
star_cuis = stars.categories.apply(lambda x:x.split(", "))
star_cuis = star_cuis.tolist()

# because it is a concated list, i flatten so that it's all just one giant list, which is the format we need
for FreqDist
flatword = [item for sublist in star_cuis for item in sublist if item in cuisines]
```

```
# Create a frequency distribution
fq = nltk.FreqDist(flatword)

# Take the top ten cuisines
star_cuis = pd.DataFrame(fq.most_common(10), columns=['Cuisines', 'Number of Reviews'])

# Plot

fig = go.Figure(data=[go.Table(
    header=dict(values=list(star_cuis.columns),
                fill_color='teal',
                align='center'),
    cells=dict(values=[star_cuis['Cuisines'], star_cuis['Number of Reviews']],
               fill_color='lavender',
               align='center'))
])
fig.update_layout(
    title_text = "The Most Popular Cuisine for Restaurants with at Least 4 Stars",
    title_font_size=30,
    title_x=0.5,
    font_family="Times New Roman",
    font_color="black",
    title_font_family="Times New Roman",
    title_font_color="black"
)
fig.update_traces(cells_font=dict(size = 12))
fig.show()

# Category now

star_e = stars.categories
star_e = stars.categories.apply(lambda x:x.split(", "))
star_e = star_e.tolist()

flatword = [item for sublist in star_e for item in sublist if item not in cuisines]

# Create a frequency distribution
fq = nltk.FreqDist(flatword)

# Take the top ten cuisines
star_cat = pd.DataFrame(fq.most_common(10), columns=['Category', 'Number of Reviews'])

# Plot

fig = go.Figure(data=[go.Table(
    header=dict(values=list(star_cat.columns),
                fill_color='lavenderblush',
```

```
                align='center'),
    cells=dict(values=[star_cat['Category'], star_cat['Number of Reviews']],
               fill_color='lavender',
               align='center'))
])
fig.update_layout(
    title_text = "The Most Popular Categories for Restaurants with at Least 4 Stars",
    title_font_size=30,
    title_x=0.5,
    font_family="Times New Roman",
    font_color="black",
    title_font_family="Times New Roman",
    title_font_color="black"
)
fig.update_traces(cells_font=dict(size = 12))
fig.show()

# Let us compare

comp(star_cuis, review_cuis)
comp(star_cat, review_est)

star_cuis.merge(review_cuis.drop_duplicates(), on=['Cuisines'],
                  how='left', indicator=True)

star_cat.merge(review_est.drop_duplicates(), on = ['Category'], how = 'left', indicator=True)

## **Subsetting to Only do Reviews With at Most Two Stars**
star_low = reviews[reviews['stars'] <= 2.0]
star_low.head(10)

# Getting the most common cuisine for restaurants less than two Stars

star_lowc = star_low.categories
star_lowc = star_low.categories.apply(lambda x:x.split(", "))
star_lowc = star_lowc.tolist()

# because it is a concated list, i flatten so that it's all just one giant list, which is the format we need
for FreqDist
flatword = [item for sublist in star_lowc for item in sublist if item in cuisines]

# Create a frequency distribution
fq = nltk.FreqDist(flatword)

# Take the top ten cuisines
star_lows = pd.DataFrame(fq.most_common(10), columns=['Cuisines', 'Number of Reviews'])
```

```
# Plot

fig = go.Figure(data=[go.Table(
    header=dict(values=list(star_lows.columns),
                fill_color='teal',
                align='center'),
    cells=dict(values=[star_lows['Cuisines'], star_lows['Number of Reviews']],
               fill_color='lavender',
               align='center'))
])
fig.update_layout(
    title_text = "The Most Popular Cuisine for Restaurants with at Most 2 Stars",
    title_font_size=30,
    title_x=0.5,
    font_family="Times New Roman",
    font_color="black",
    title_font_family="Times New Roman",
    title_font_color="black"
)
fig.update_traces(cells_font=dict(size = 12))
fig.show()

# Category now

# because it is a concated list, i flatten so that it's all just one giant list, which is the format we need
for FreqDist
flatword = [item for sublist in star_lowc for item in sublist if item not in cuisines]

# Create a frequency distribution
fq = nltk.FreqDist(flatword)

# Take the top ten cuisines
star_low_rest = pd.DataFrame(fq.most_common(10), columns=['Category', 'Number of Reviews'])

# Plot

fig = go.Figure(data=[go.Table(
    header=dict(values=list(star_low_rest.columns),
                fill_color='lavenderblush',
                align='center'),
    cells=dict(values=[star_low_rest['Category'], star_low_rest['Number of Reviews']],
               fill_color='lavender',
               align='center'))
])
fig.update_layout(
    title_text = "The Most Popular Categories for Restaurants with at Most 2 Stars",
    title_font_size=30,
```

```
    title_x=0.5,
    font_family="Times New Roman",
    font_color="black",
    title_font_family="Times New Roman",
    title_font_color="black"
)
fig.update_traces(cells_font=dict(size = 12))
fig.show()

# Let us compare

comp(star_lows, review_cuis)
comp(star_low_rest, review_est)


star_lows.merge(review_cuis.drop_duplicates(), on=['Cuisines'],
                  how='left', indicator=True)

star_low_rest.merge(review_est.drop_duplicates(), on=['Category'],
                  how='left', indicator=True)
```

Code for Recommendation System

```
import pandas as pd
import os
from sklearn.decomposition import TruncatedSVD
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from joblib import parallel_backend
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

os.chdir("C:\\Users\Rohan\Downloads")

# Data processing
precovid = pd.read_csv('precovid_reviews.csv')
postcovid = pd.read_csv('postcovid_reviews.csv')

cols = ['business_id', 'name', 'address', 'latitude', 'longitude', 'is_open', 'categories', 'hours',
'review_count']
merger = pd.merge(precovid, postcovid, on = cols)
merger = merger[merger['is_open'] != 0] # get rid of closed restaurants
# Formatting data for analysis
business_ids = pd.concat([merger['business_id'], merger['business_id']]).reset_index(drop = True) # merge
before and after business ids for creation of sparse matrix later
## will help format matrix for SVD

bus_stars = pd.concat([merger['stars_x'], merger['stars_y']]).reset_index(drop = True)
user_ids = pd.concat([merger['user_id_x'], merger['user_id_y']]).reset_index(drop = True) # merge pre and
post covid data by column
stars = pd.concat([merger['customer_stars_x'], merger['customer_stars_y']]).reset_index(drop = True)
dates = pd.concat([merger['date__x'], merger['date__y']]).reset_index(drop = True)
names = pd.concat([merger['name'], merger['name']]).reset_index(drop = True)
addresses = pd.concat([merger['address'], merger['address']]).reset_index(drop = True)
state = pd.concat([merger['state__x'], merger['state__y']]).reset_index(drop = True)
city = pd.concat([merger['city_x'], merger['city_y']]).reset_index(drop = True)
postal_code = pd.concat([merger['postal_code_x'], merger['postal_code_y']]).reset_index(drop = True)
useful = pd.concat([merger['useful_x'], merger['useful_y']]).reset_index(drop = True)
funny = pd.concat([merger['funny_x'], merger['funny_y']]).reset_index(drop = True)
cool = pd.concat([merger['cool_x'], merger['cool_y']]).reset_index(drop = True)


businesses2 = [business_ids, user_ids, stars, useful, funny, cool, dates, names, addresses, state, city,
postal_code, bus_stars]
businesses3 = pd.concat(businesses2, axis = 1) # combine pre and post covid data
businesses4 = businesses3.rename(columns = {'business_id': 'business_id', 0: 'user_id', 1: 'stars', 2:
'useful', 3: 'funny', 4: 'cool', 5: 'date', 'name': 'name', 'address': 'address', 6 :'state', 7 : 'city', 8 :
'postal_code', 9: 'bus_stars'})
```

```
s_mat = businesses4.pivot_table(values = 'stars', index = 'user_id', columns = 'name', fill_value=
0).drop_duplicates(keep = False) # create sparse matrix with customer stars
s_mat.shape

res_names = s_mat.columns.tolist()
businesses5 = businesses4.loc[businesses4['name'].isin(res_names)]
# businesses4.groupby('name')['stars'].count().sort_values(ascending = False)
# Descriptive Statistics

avg_bus_stars = bus_stars.sum() / len(bus_stars) # calculate average overall stars
avg_cust_stars = stars.sum() / len(stars)
avg_rev_counts = merger['review_count'].sum() / len(merger['review_count'])
useful_avg = useful.sum() / len(useful)
funny_avg = funny.sum() / len(funny)
cool_avg = cool.sum() / len(cool)

averages = [avg_bus_stars, avg_cust_stars, useful_avg, funny_avg, cool_avg, avg_rev_counts]

bus_star_mode = bus_stars.mode().values[0] # calculate mode for overall stars
cust_stars_mode = stars.mode().values[0]
useful_mode = useful.mode().values[0]
funny_mode = funny.mode().values[0]
cool_mode = cool.mode().values[0]
rev_counts_mode = merger['review_count'].mode().values[0]

modes = [bus_star_mode, cust_stars_mode, useful_mode, funny_mode, cool_mode, rev_counts_mode]

bus_star_std = bus_stars.std() # calculate standard deviation for overall stars
cust_stars_std = stars.std()
useful_std = useful.std()
funny_std = funny.std()
cool_std = cool.std()
rev_counts_std = merger['review_count'].std()

stds = [bus_star_std, cust_stars_std, useful_std, funny_std, cool_std, rev_counts_std]

bus_star_median = bus_stars.median() # calculate median for overall stars
cust_stars_median = stars.median()
useful_median = useful.median()
funny_median = funny.median()
cool_median = cool.median()
rev_counts_median = merger['review_count'].median()

medians = [bus_star_median, cust_stars_median, useful_median, funny_median, cool_median, rev_counts_median]

data = {'Mean': averages, 'Median': medians, 'Mode': modes, 'Standard Deviation': stds}
```

```
des_stats = pd.DataFrame(data = data)
des_stats.index = ['Overall Stars', 'Customer Stars', 'Useful', 'Funny', 'Cool', 'Number of Reviews']
des_stats.T


# SVD
A = s_mat.values.T # extract numerical values for SVD
svd = TruncatedSVD(n_components=800, random_state=12, algorithm = 'randomized')

with parallel_backend('multiprocessing', n_jobs = -1): # runs SVD on all processes available
    #u,s,vt = randomized_svd(A, n_components = 800, random_state = 21)
    res_matrix = svd.fit_transform(A)
    corr_matrix = np.corrcoef(res_matrix)
    usig = res_matrix # is actually u * sigma
    s = svd.singular_values_
    vt = svd.components_
    ex_var_rat = svd.explained_variance_ratio_

#print("U: ", u.shape, "S: ", s.shape, "V: ", vt.shape)
#sparse = 1 - np.count_nonzero(A) / A.size # sparsity of matrix

svd2 = TruncatedSVD(n_components=2847, random_state=1, algorithm = 'randomized') # SVD to determine number of
ideal components

with parallel_backend('threading', n_jobs = -1): # runs SVD on all threads available
    #u,s,vt = randomized_svd(A, n_components = 800, random_state = 21)
    res2_matrix = svd2.fit_transform(A)
    corr2_matrix = np.corrcoef(res2_matrix)
    usig2 = res_matrix # is actually u * sigma
    s2 = svd2.singular_values_
    vt2 = svd2.components_
    ex_var_rat2 = svd2.explained_variance_ratio_

# Graphs
plt.plot(np.cumsum(ex_var_rat2))
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Explained Variance')
plt.title('Variance Explained by Components')

plt.plot(np.cumsum(ex_var_rat))
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Explained Variance')
plt.title('Variance Explained by Components')

plt.bar(range(1, len(ex_var_rat) + 1), ex_var_rat)
plt.ylabel('Explained Variance')
```

```
plt.xlabel('Number of Components')
plt.title('Variance Explained by Features')

businesses5.head()

top_5 = businesses4.groupby('name')['stars'].count().sort_values(ascending = False)

sns.heatmap(corr_matrix)
plt.xlabel('Restaurant Index')
plt.ylabel('Restaurant Index')
plt.show()

subset_corr = corr_matrix[10:20, 10:20]
sns.heatmap(subset_corr)

# Recommendation Function

def top_restaurants(restaurant, n = 10, correlation_factor = 0.9):
    """
    Function returns top n correlated restaurants with given restaurant name. Can adjust Pearson correlation
factor as needed in order to return more restaurants.
    """
    selected_restaurant = int(list(s_mat.columns).index(str(restaurant))) # get index of restaurant from
matrix
    recommendations = s_mat.columns[(corr_matrix[selected_restaurant] < 1.0) &
(corr_matrix[selected_restaurant] > correlation_factor)].sort_values(ascending = False).values.tolist() #
returns highest correlated restaurants with restaurant in index of correlation matrix



    res_df = pd.merge(businesses4, pd.DataFrame({'name': recommendations}), on = 'name').drop_duplicates(keep
= False) # create dataframe that contains the info about the recommended restaurants
    res_df = res_df[['bus_stars', 'name', 'address','city', 'state']].drop_duplicates() # creating dataframe
with recommended restaurants
    res_df = res_df.rename({'bus_stars': 'stars'}, axis = 1)
    restaurants = res_df.sort_values('name').drop_duplicates(subset = ['address']) # drop duplicated
instances of restaurants in recommendation dataframe



    return restaurants[0:n].sort_values(by = ['stars'], ascending = False)

top_restaurants(restaurant = "7-Eleven")
```

Code for Parts 1a and 2a

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

precovid = pd.read_csv("/Users/cmonzon/Desktop/school/sta141c/yelp_reviews/precovid_reviews.csv")

postcovid = pd.read_csv("/Users/cmonzon/Desktop/school/sta141c/yelp_reviews/postcovid_reviews.csv")

# Combine pre and post covid
combined = pd.concat([precovid, postcovid])
# Subset by still open
reviews = combined[combined['is_open'] == 1]

# Remove states with very few unique restaurants
list_states = reviews.state_.unique()

for state in list_states:
    print(state, len(reviews[reviews['state_'] == state].business_id.unique()))
# Atlanta, GA area
ga_reviews = reviews[reviews['state_'] == 'GA']
# Vancouver, BC area
bc_reviews = reviews[(reviews['state_'] == 'BC') | (reviews['state_'] == 'ABE')]
# Boston, MA area
ma_reviews = reviews[reviews['state_'] == 'MA']
# Austin, TX area
tx_reviews = reviews[reviews['state_'] == 'TX']
# Portland, OR area
or_reviews = reviews[(reviews['state_'] == 'OR') | (reviews['state_'] == 'WA')]
# Orlando, FL area
fl_reviews = reviews[reviews['state_'] == 'FL']
# Columbus, OH area
oh_reviews = reviews[reviews['state_'] == 'OH']
# Boulder, CO area
co_reviews = reviews[reviews['state_'] == 'CO']
region_reviews = [ga_reviews, bc_reviews, tx_reviews, ma_reviews,
                  or_reviews, fl_reviews, oh_reviews, co_reviews]
# reset index for all dfs
for i in range(len(region_reviews)):
    region_reviews[i] = region_reviews[i].reset_index(drop=True)
# Get top 100 restaurants with most reviews in each region
list_top100 = []

for region in region_reviews:
    group = region.groupby(['name','business_id'])
    region100 = group.mean().sort_values(by='review_count', ascending=False).head(100)
```

```
    region100 = region100[['latitude', 'longitude', 'stars', 'review_count']]
    list_top100.append(region100)

df_top100 = pd.concat(list_top100)

# Scrape cuisines
from bs4 import BeautifulSoup
import requests
categories_url = 'https://blog.yelp.com/businesses/yelp_category_list/'
header = {'user-agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/110.0.0.0 Safari/537.36'}
page = requests.get(categories_url, headers = header)

html = BeautifulSoup(page.content, 'html.parser')
uls = html.find_all('ul', class_ = None)     # get all <ul> tags
categories_html = uls[90]     # get <ul> containing the restaurant categories

# Get all categories into list format
cuisines = [li.text.split('\n', 1)[0] for li in categories_html.findAll('li')]
```