Zileng Chua, Anh Nguyen, Jiacheng Yu

# YELP RESTAURANT STAR RATING PREDICTION

BUAN 5510 – Capstone Project in Business Analytics

Date: December 6, 2022

# Table of Contents


**ABSTRACT** ..... **3**

**INTRODUCTION** ..... **3**

Background Information ..... 3
Problem Statement ..... 3
Contributions ..... 4

**DATASET DESCRIPTION AND EXPLORATION** ..... **4**

Yelp Dataset ..... 4
*Business Dataset* ..... *5*
*Review Dataset* ..... *6*
U.S. Census Income Dataset ..... 8

**LITERATURE REVIEW** ..... **8**

Identifying Influential Factors for Yelp Business Ratings ..... 8
Applications of Machine Learning Models on Yelp Data ..... 8
Sentiment Analysis of Yelp's Ratings Based on Text Reviews ..... 9
Prediction of Rating Based on Review Text of Yelp Reviews ..... 10
Predicting a Business' Star in Yelp from Its Reviews' Text Alone ..... 10
Yelp Dataset Challenge: Review Rating Prediction ..... 11

**DATA PRE-PROCESSING** ..... **11**

Business Data ..... 11
*Sub-setting* ..... *11*
*Merging Business and Census Income Datasets* ..... *12*
*Processing the 'Attribute' Column* ..... *12*
*Final Processed Business Dataset* ..... *13*
Review Data ..... 14
*Sub-setting* ..... *14*
*Text Review Pre-processing* ..... *14*
*Sentiment Analysis* ..... *14*
*Merging Review with Business Data* ..... *15*

**MODELS AND EVALUATION** ..... **15**

All Features Models ..... 16
Reduced Features Models ..... 16
PCA Models ..... 16
K-Fold Cross Validation ..... 17
Hyperparameter Tuning ..... 17
Clustering ..... 17
Interpretations ..... 18

**DISCUSSION** ..... **19**

Domain Knowledge ..... 19
Methodological Contributions ..... 19

**CONCLUSION** ..... **20**

*Summary* ..... *20*
Limitations ..... 20
Future Projects ..... 21

**REFERENCES** ..... **21**

**APPENDICES** ..... **23**

A. DATA DICTIONARY ..... 23
*A-1. Yelp Business Dataset* ..... *23*
*A-2. Yelp Review Dataset* ..... *24*
*A-3. U.S. Census Income Dataset* ..... *25*
B. ENTITY RELATIONSHIP DIAGRAM (ERD) OF YELP DATABASE ..... 26
C. PYTHON CODE ..... 27
*C-1. Business Pre-processing* ..... *27*
*C-2. Review Pre-processing* ..... *35*
*C-3. Modeling* ..... *40*
*C-4. Hyperparameter Tuning* ..... *54*
*C-5. Clustering* ..... *57*

# Abstract

In this research, we use Yelp Open Dataset [1] to perform predictive analytics to determine Yelp business ratings based on the attributes of the business, as well as the socioeconomic factors and the reviews it receives from Yelp users over time. For the socioeconomic factors, we retrieve household income data from U.S. Census Bureau [11] to combine with the original Yelp business and review datasets. Our motivation to do this project is that the datasets are about real businesses and are large in terms of size. Consequently, the insights from our study can be applied in the real world so that business owners can understand the effects of different factors on their business ratings on Yelp. To predict business ratings, we build different Machine Learning models using different algorithms: Decision Tree, Random Forest, Gradient Boosting, Neural Network, and Support Vector Machine. Our predictive models are based on regression algorithms, so the performance evaluation metric used in our study is R-squared. Our result shows that Gradient Boosting is the most stable algorithm as it has consistently good performance in all the models (more than 80% R-squared) across different feature sets: original (all) features, features selected through Primary Component Analysis (PCA), and reduced features based on feature importances.


# Introduction

## Background Information

With the increased internet and social media, online reviews are playing a more crucial role, and star ratings can greatly influence customers' behaviors in selecting local businesses. Users are more likely to make purchases when the businesses have high star ratings and more positive reviews. According to Bright Local 2022 [2] research, there are 98% of customers read online reviews for local businesses, and 77% of them 'always' and 'regularly' read online reviews. This shows that how important the online reviews to customers when searching for local business on the internet. With the large number of reviews available on the online crowd-sourced review forums such as Yelp, Google, and Amazon, customers would also prefer to look at the star ratings to the evaluate the local businesses based on previous customers experience than reading all the reviews, which usually takes longer time to make decision, and no one would want to do this. Online crowd-sourced websites also provide a platform for local businesses to connect to diverse of customers using the advertisement strategies and target customers with personalized suggestions using the recommender systems. Yelp is one of the most popular online crowdsource platform for users to search for restaurants and other local businesses. In our research, we are using the dataset from Yelp Dataset Challenge [1] that consists of Yelp's businesses, reviews, and user data.

## Problem Statement

Online reviews can be subjected to different people depending on their preferences, personalities, priorities, and personal experiences. Because of this fact, online reviews are usually inconsistent in terms of length and opinions for the same businesses and same star ratings. For example, two users may have experienced the same services and served with the same quality of food in the same restaurant, but they might give different ratings for different reasons. One rated five star for overall good experience while the other rated three stars for long wait time. It is important for us to understand the context of reviews and to use relevant information to predict the ratings that are relevant to users' reviews. Additionally, Yelp rounds their business average ratings to the nearest

half-star, for example, 2.75 will be shown as 3.0, and 4.4 will be shown as 4.0. This might not show the accurate ratings to users and will lead to higher or lower expectations. And when the businesses do not meet customer expectations, they will receive low ratings. In this paper, we would like to utilize machine learning techniques on the Yelp Open Dataset [1] to conduct predictive analytics of business star ratings based on the business attributes and the review text. To be more specific, we focus on predicting business ratings for open restaurant businesses in the states of Florida and Pennsylvania so that we can keep the area of study more tractable.

### Contributions

Restaurants have always played an essential role in the business, social, intellectual, and artistic life of a thriving society. (Feldman, 2015) [3]. According to Finances Online statistics (Andre, 2022) [4], Food and Restaurants are also the topics that are most discussed by Yelp users. Therefore, our focus of the study can be widely applied by many food businesses to educate owners on factors that have great influences on their ratings. This can help not only improve their customer satisfaction but also attract new customers. Besides, by incorporating both business attributes and reviews to predict business ratings, businesses will be less dependent on their customers' experiences to get a high rating. Since customer satisfaction varied among different individuals, relying on reviews alone to determine rating could have some bias issues, and we think that our approach to evaluating a business will be more balanced and objective.

## Dataset Description and Exploration

For this research, we obtained two datasets from two sources. One is Yelp open dataset from Kaggle/Yelp website and the other is the income data table from the U.S. Census Bureau database. The full data dictionary of all data files is included in Appendix A of this paper. The data dictionary provides information about variables' name, description, valid domain value, data type, length, example values, data attributes, and null ratio.

### Yelp Dataset

The Yelp dataset [1] is a subset of Yelp's businesses, reviews, and user data. It was initially made available by Yelp, Inc. for the Yelp Dataset Challenge, which supports students' academic research and analysis. We retrieved the latest version (last updated in March 2022) of the dataset from Kaggle, an online data science community platform. This dataset is provided directly by Yelp, Inc. and can also be accessed at its website https://www.yelp.com/dataset. It provides information of real businesses across 8 metropolitan areas in the USA and Canada. The complete dataset contains 6 files including 1 PDF file that is about user agreement and 5 JSON data files; each is named yelp_academic_dataset_(file name).json (file name takes one of the following: business, check-in, review, tip, and user) and consists of one json-object-per-line. The descriptions of each data file are summarized in Table 1 below, and the relationship among the sub-datasets within Yelp database is demonstrated through an Entity Relationship Diagram (ERD) in Appendix B.

| Data File | Size | Observations | Features |
|---|---|---|---|
| **business** | 119 MB | 150,346 | 14 |
| **check-in** | 287 MB | 131,930 | 2 |
| **review** | 5.34 GB | 6,990,280 | 9 |

| | | | |
|---|---|---|---|
| **tip** | 181 MB | 908,915 | 5 |
| **user** | 3.36 GB | 1,987,897 | 22 |
| **Total Size** | **9.3 GB** | | |

*Table 1. Description of Yelp data files*

The whole dataset includes nearly 7 million reviews from 1.9 million users evaluating more than 150,000 businesses from 2005 to 2022. However, for our problem statement of this research, we only use the review and business datasets. And in terms of time frame, we are utilizing all the review years because the rating of a business is usually calculated as the weighted average of all users' ratings it received over time.

## Business Dataset

Each data point of Yelp business data file is a json object which specifies the business ID, its name, location, attributes, stars, review count, opening hours, etc. All business IDs are uniquely represented by randomly assigned combinations of numbers and letters. An example of an object line in the business dataset is shown in Figure 1 below.

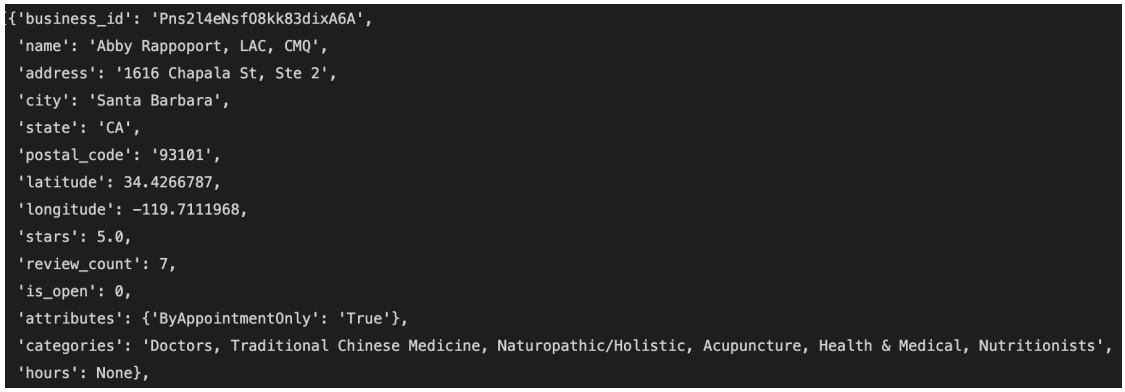



*Figure 1. An example data point (json object) of Yelp business dataset*

The preliminary exploratory analysis of the business dataset includes descriptive statistics of relevant numerical data (Table 2) and the study of the distributions of businesses with respect to category and location (Figures 2 and 3).

| | **stars** | **review_count** | **is_open** |
|---|---|---|---|
| **count** | 150,346 | 150,346 | 150,346 |
| **mean** | 3.6 | 44.87 | 0.8 |
| **standard deviation** | 0.97 | 121.12 | 0.4 |
| **min** | 1 | 5 | 0 |
| **25%** | 3 | 8 | 1 |
| **50%** | 3.5 | 15 | 1 |
| **75%** | 4.5 | 37 | 1 |
| **max** | 5 | 7,568 | 1 |

*Table 2. Descriptive Statistics of numerical variables in Yelp business dataset*

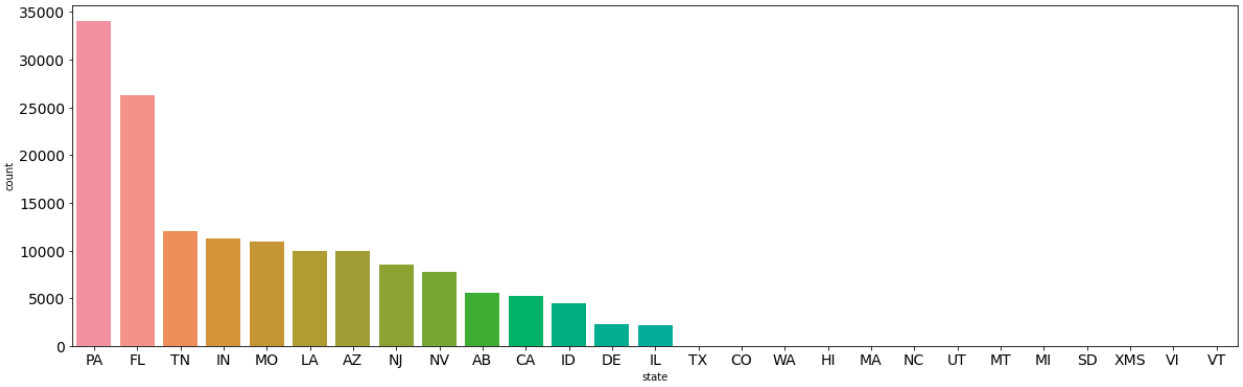



*Figure 2. Number of businesses in each state*

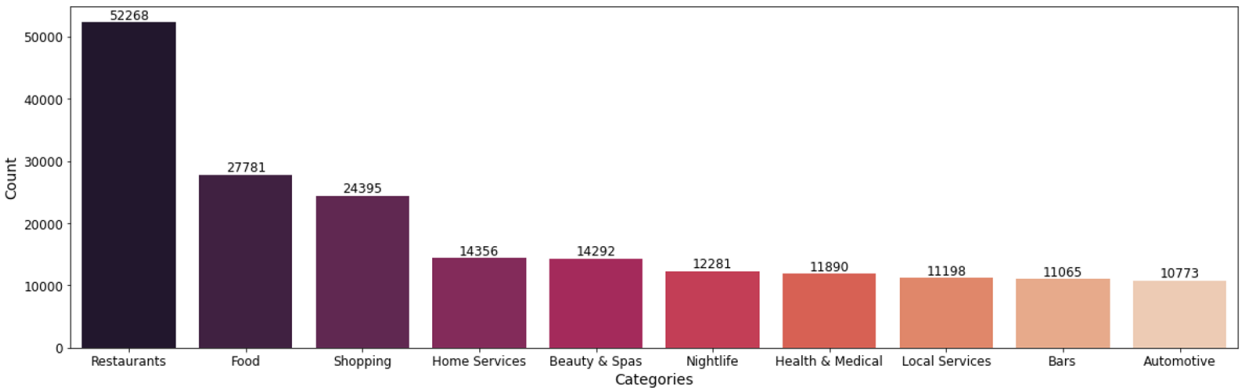



*Figure 3. Top 10 business categories by Number of businesses*

From the table and visualizations above, Pennsylvania (PA) and Florida (FL) are the two states with the highest number of unique businesses (Yelp's data set may be only a part of their complete data that they have released; hence, we are not seeing popular cities like New York, Los Angeles, San Francisco, Seattle, etc.). Besides, we have the greatest number of unique businesses that belong to the 'restaurant' category. And in the dataset, there are about 80% of all business observations that are open, meaning some businesses had been closed due to various reasons. As a result of this EDA step, we have reduced the data file to select approximately 14,000 open restaurants in Pennsylvania (PA) and Florida (FL) to focus on for this study.

## Review Dataset

Like the business dataset, the Yelp review dataset is also in JSON format, which means each line is a json object that looks like Figure 5 below. Each data point is comprised of review ID, user ID, business ID, review stars rating, review votes (useful, funny, and cool), text review, and review date. The exploratory data analysis of the review dataset includes the variables' correlation analysis and the findings of top keywords or topics from the text review.

```
{'review_id': 'BiTunyQ73aT9WBnpR9DZGw',
 'user_id': 'OyoGAe7OKpv6SyGZT5g77Q',
 'business_id': '7ATYjTIgM3jUlt4UM3IypQ',
 'stars': 5.0,
 'useful': 1,
 'funny': 0,
 'cool': 1,
 'text': "I've taken a lot of spin classes over the years, and nothing compares to the classes at Body Cycle. From the nice, clean space and amazing bikes, to the welcoming and motivating instructors, every class is a top notch work out.\n\nFor anyone who struggles to fit workouts in, the online scheduling system makes it easy to plan ahead (and there's no need to line up way in advanced like many gyms make you do).\n\nThere is no way I can write this review without giving Russell, the owner of Body Cycle, a shout out. Russell's passion for fitness and cycling is so evident, as is his desire for all of his clients to succeed. He is always dropping in to classes to check in/provide encouragement, and is open to ideas and recommendations from anyone. Russell always wears a smile on his face, even when he's kicking your butt in class!",
 'date': '2012-01-03 15:28:18'},
```


*Figure 4. An example data point (json object) of Yelp review dataset*

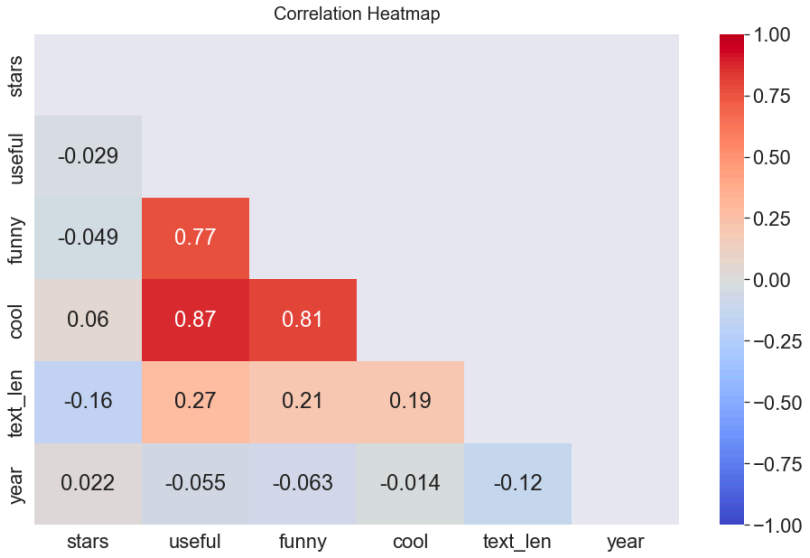



*Figure 5. Correlation heatmap of variables in Yelp review dataset*


*Figure 6. Most frequent words in Yelp text reviews*

The correlation analysis (Figure 6) shows that there is a high correlation among votes' variables (funny, cool, useful). Therefore, these variables should not be chosen as features when we build models. In addition, since review is the largest dataset of all the files in Yelp dataset, it is important to get a good picture of all 7 million users' reviews to know what most of them are about. Figure 7 is the word cloud showing the most frequent words appeared in all the text reviews in the review dataset. Larger font indicates higher word frequency. As it can be seen, most reviews are about food or restaurant businesses. This finding also helped us define the focus of our study more clearly so that the size of our data could be significantly reduced for easier and faster processing.

### U.S. Census Income Dataset

The income dataset (income data.csv) [11] was retrieved from the U.S. Census Bureau database. For this project, we extracted the ACS 5-year estimates of average income and total number of households for all 5-digit postal zip codes fully and partially within Florida and Pennsylvania in 2020. The size of the CSV data file is 179 KB, which consists of 2,781 rows and 3 columns (geographic area name, estimated total households, and estimated mean household income). Geographic area name is the full Census 5-digit ZIP code Tabulation Area, which is under the format as ZCTA5 followed by a 5-digit postal code. As we are interested in finding out how socioeconomic factors affect a business's Yelp ratings in our study, we want to include in our predictive analysis the total households and average income of the area where a business is located, and this dataset will be used to merge with the business dataset on the postal code.

## Literature Review

### Identifying Influential Factors for Yelp Business Ratings

Link [5]: https://cseweb.ucsd.edu/classes/wi15/cse255-a/reports/fa15/010.pdf
Summary:
The author of this article wants to accurately predict the ratings and identify certain key attributes that tend to influence the overall performance of the business. Except review sentiment, he considers that some attributes such as location, price range, acceptance of credit cards, and reservations should also be important features that will affect the star rating of the business. The author uses Linear Regression, Ridge Regression, Multinomial Naive Bayes Regression, Decision Tree Regression, and Random Forests Regression as his machine learning algorithms and provides the conclusion that selected features greatly influence a business' star ratings. Review sentiments play an important role in deciding the star ratings so businesses should focus more on customer sentiment to improve their performance and evaluation.

### Applications of Machine Learning Models on Yelp Data

Link [6]: http://www.apjis.or.kr/pdf/02_%EC%9A%B0%EC%A2%85%EC%9A%B1(35-49).pdf
Summary:
Yelp has a monthly average of 34 million unique visitors and more than 171 million reviews written. It generates a vast data lake through user interaction which makes its analysis a challenge that can only be handled with Machine Learning models. In the research, the authors presented the results of different Machine Learning models including text analysis, classification, and recommendation systems applied on Yelp Data. Their models are done on powerful cloud

platforms, particularly Databricks and Azure, which can affordably speed up machine learning computations.

Firstly, the goal of the recommender tool is to provide Yelp users with recommendations for business categories based on their previous business ratings, as well as the business ratings of other users. It also has a feature to predict the future ratings by user for a category. To build the recommender, the authors split the joined dataset in SparkML to train and test fractions by 0.7 to 0.3 ratio then used Alternating Least Square (ALS) algorithm. The RMSE was chosen to be the evaluation metrics to compare the performance of similar models on two platforms, and it was concluded that models run on SparkML easier and faster with much lower RMSE, which makes SparkML the better choice than AzureML.

Secondly, the classification problem addressed in the research is to predict the popularity of the business to have stars greater than 3 (popular) to have stars less than 3 (unpopular). All attribute columns related to the food category are considered for the classification models. These models resulted in predicting a very small percentage of food business having less than 3 stars, which indicates that the model does not train well to predict unpopular business. This is due to unbalanced data between popular and unpopular businesses. Lastly, text analysis was conducted to predict the number of likes based on two columns (compliment_count and text) of the tip dataset. Text data was cleaned through removing stop words, which will only leave relevant texts for the number of likes prediction. Comparing two algorithms, uni-gram feature extraction and n-feature extraction in Azure ML studio and logistic regression model in Spark, the authors found that the logistic regression model gave perfect score for predicting the likes with an ideal AUC (area under the ROC curve) value of 1.0.

### Sentiment Analysis of Yelp's Ratings Based on Text Reviews

Link [7]:
http://cs229.stanford.edu/proj2014/Yun%20Xu,%20Xinhui%20Wu,%20Qinxia%20Wang,%20Sentiment%20Analysis%20of%20Yelp's%20Ratings%20Based%20on%20Text%20Reviews.pdf
Summary:
Sentiment Analysis is one of the most popular techniques used in natural language processing to analyze text data. It has been widely applied in many cases such as predicting books' or movies' ratings based on critiques or assigning ratings to YouTube videos based on viewers' comments.to reduce redundancy in subsequent feature selection.

Two feature selection algorithms implemented are opinion lexicon and a variation of feature dictionary built by looping over the training set word by word. Each method has both pros and cons. While the opinion lexicon is fast as no looping required, it might select irrelevant features because its feature set is small and consists exclusively of adjectives that has sentiment meaning extracted not from Yelp dataset, which is not representative and can contain bias problems. In contrast, building the feature set using training data results in a larger feature set that selects only relevant features from the Yelp data set itself and improves both precision and recall significantly. However, looping over the training set to select relevant features can be slow when our training size becomes large.

The authors eventually conducted the Perceptron, Naïve Bayes, and Multi-Class Support Vector Machine (Multi-Class SVM) and Nearest Centroid algorithms to model the selected features and used Precision and Recall as the evaluation metric to measure the rating prediction performance. Multi-class SVM and Nearest Neighbor both have low precision and recall. The perceptron algorithm has the highest precision and recall for Star 1 and 5 Ratings, but the predictions are poor for Star 2, 3, and 4. It also suffers from high bias on the training dataset. While Naive Bayes (binarized) does not have high precision and recall due to insignificant distinguishment between ratings 4 and 5 stars and among 1-, 2-, 3-star ratings., its performance is the best overall. The authors expected improvement in prediction accuracy if ratings are combined into classification categories.

### Prediction of Rating Based on Review Text of Yelp Reviews

Link [8]: https://cseweb.ucsd.edu/classes/wi15/cse255-a/reports/fa15/031.pdf
Summary:
In this article, authors use review text only to predict the star rating for business. They notice that the text of a review is usually overlooked, and people are more likely to look at users' historic rating. In fact, text should be the core information to predict the star rating for a business because a review is the opinion of the user himself about the business. To achieve the goal, authors decide to use frequency distribution of the most common words and apply with a total of two classification models which are Naive Bayes' Classifier and the Support Vector Machine Classifier and one prediction model which is linear regression. It is correct to admit the importance of text itself, however, other attributes of business which can affect the star rating system should also be considered to add into the feature sets. Other than that, some other algorithms can be used to achieve the best result.

### Predicting a Business' Star in Yelp from Its Reviews' Text Alone

Link [9]: https://arxiv.org/ftp/arxiv/papers/1401/1401.0864.pdf
Summary:
The authors state that is almost impossible for users to find the expected result due to the extreme large number of reviews. That's why Yelp provide star rating system to tell users about the business overview. They make a prediction that the rating score of business is based on reviews only and find the best prediction result through a combination of three feature generation and four machine learning models.

To decide the features, authors explore the dataset and decide to use word frequency from the reviews. They choose general frequent words, frequent words after doing part-of-speech analysis and adjectives as their three hypotheses of selected combination of features.

Linear Regression, Support Vector Regression with normalized features, Support Vector Regression without normalized features, and Decision Tree Regression are the four learning models picked by the authors. And they use the Root Mean Square Error instead of accuracy.

In their result, authors figure that no matter which combination of features he uses, Linear Regression always performs the best and RMSE drops to almost 0.6014 when they select general frequent words, which implies that there is linear correlation between their features and target.

Also, the best performance for different models appears while top 50 to 100 frequent words are selected.

### Yelp Dataset Challenge: Review Rating Prediction

Link [10]: https://arxiv.org/pdf/1605.05362.pdf

Summary:

The author believes that online reviews have significant influence on consumer shopping behavior, so Review Rating Prediction, which is the problem of predicting a user's star rating for a product by giving the user's text review, is becoming a hard but popular problem for review websites and machine learning. The author creates sixteen different models by four feature extraction with four machine learning algorithms.

The author states that selected features are all rely on semantic analysis of the text. She chooses unigrams as her features of first model, next she adds bigrams to make her second feature group and then adds trigrams as her third feature groups. She also uses Latent Semantic Indexing to find topics in the review as her last feature group.

As for machine learning algorithms, the author selects Logistic regression, Naive Bayes classification, Perceptron, and Linear Support Vector Classification. Author provides both RMSE and accuracy to tell the performance of model. Comparing with all groups of features, Logistic Regression achieved the highest accuracy of 64% and lowest RMSE of 0.78 using the top 10,000 Unigrams and Bigrams as features.

## Data Pre-processing

### Business Data

#### Sub-setting

As discussed in the Data Exploration section, considering the domain of our research and the limitations of our computers to handle large data files, we only take a subset of the original data files. From our EDA, we decided to choose only the open restaurant businesses (category contains 'Restaurant' and is_open variable equals 1) in two states: Pennsylvania (PA) and Florida (FL). The total number of businesses is nearly 14,000. Among the 14 columns in the original business dataset, we only keep 9 columns that we think are useful to use in our later processing steps. The required columns are business ID, business name, categories, city, state, postal code, business star ratings, review count, and attributes. After getting the subset of the business data, we checked for null (missing) values of all the columns (Table 3). We had to drop all rows with null values in any column, resulting in a total of 13,844 rows.

| Column name | Number of nulls | Null ratio (%) |
|---|---|---|
| **attributes** | 148 | 1.0577 |
| **postal_code** | 1 | 0.0071 |
| **business_id** | 0 | 0 |
| **business_name** | 0 | 0 |
| **categories** | 0 | 0 |
| **city** | 0 | 0 |
| **state** | 0 | 0 |

| | | |
|---|---|---|
| **stars** | 0 | 0 |
| **review_count** | 0 | 0 |

*Table 3. Null count summary of variables in the business sub-dataset*

### Merging Business and Census Income Datasets

Next, the business sub-dataset is then merged with the Income dataset that contains the estimated total household and the average household income in PA and FL. To do this, we obtained the 5-digit postal ZIP codes from the Geographic Area Name column of the Income dataset to join the two datasets on the postal code column. After merging, we observed that there are some zip codes with no household, so the average income values of these areas are either missing or in an invalid format. We removed these rows, which left 13,724 rows remaining in the merged dataset.

### Processing the 'Attribute' Column

The value of the 'attributes' column is originally in a string format that looks like Figure 7. Consequently, we need to breakdown the column to extract the attributes into separate columns for each restaurant. We converted the string format into a dictionary then to a Pandas' Series format to get a Data Frame column for each of the attributes. There are 39 attributes extracted from this step.


```
'{\'RestaurantsDelivery\': \'False\', \'OutdoorSeating\': \'False\', \'BusinessAcceptsCreditCards\': \'False\', \'BusinessParking\': "{\'garage\': False,
\'street\': True, \'validated\': False, \'lot\': False, \'valet\': False}", \'BikeParking\': \'True\', \'RestaurantsPriceRange2\': \'1\',
\'RestaurantsTakeOut\': \'True\', \'ByAppointmentOnly\': \'False\', \'WiFi\': "u\'free\'", \'Alcohol\': "u\'none\'", \'Caters\': \'True\'}'
```


*Figure 7. An example value of the 'attributes' column in the business dataset*

| colnames | null | null_ratio% |
|---|---|---|
| HairSpecializesIn | 13723 | 99.9927 |
| Open24Hours | 13720 | 99.9709 |
| DietaryRestrictions | 13720 | 99.9709 |
| RestaurantsCounterService | 13719 | 99.9636 |
| AcceptsInsurance | 13719 | 99.9636 |
| AgesAllowed | 13683 | 99.7013 |
| BYOBCorkage | 13300 | 96.9105 |
| Smoking | 12741 | 92.8374 |
| GoodForDancing | 12728 | 92.7426 |
| ByAppointmentOnly | 12661 | 92.2544 |
| CoatCheck | 12503 | 91.1032 |
| Corkage | 12422 | 90.5130 |
| BestNights | 12320 | 89.7697 |
| BYOB | 12135 | 88.4217 |
| Music | 12044 | 87.7587 |
| DriveThru | 11982 | 87.3069 |
| BusinessAcceptsBitcoin | 11631 | 84.7493 |
| DogsAllowed | 10148 | 73.9435 |
| HappyHour | 9943 | 72.4497 |
| WheelchairAccessible | 9571 | 69.7391 |
| RestaurantsTableService | 7815 | 56.9440 |

*Figure 8. Null count summary of attribute columns*

We then examined the null count of each attribute column and decided to remove 20 attributes with more than 50% null values, indicating the features are not represented by at least half of the

sample size. Figure 8 shows that the attributes that will be dropped due to having high null ratio. Some of these removed attributes are in fact irrelevant or unrealistic features that do not apply to 'Restaurant' businesses (e.g., Hair Specializes In, Accepts Insurance, or Open 24 Hours).

Moreover, some of the extracted attribute columns from the step above still have sub-attributes (e.g., Business Parking, Ambience, and Good For Meal) and categorical values such as Alcohol, Noise Level, Restaurant Attire, WiFi, and Restaurant Price Range. For instance, inside the Business Parking attribute, we need to have separate Data Frame columns for garage, street, validated, lot, and valet parking. Since these columns need further splitting, to avoid code errors, we dropped all rows with nulls from the dataset prior to applying the same technique above to split the sub-attributes into different columns. As a result, we have more than 8,000 businesses remained in the dataset. Regarding the categorical attributes, we converted each category of the attribute into a dummy variable. For each category, one of the created dummy variables is dropped to make it the reference category in our models.

### Final Processed Business Dataset

After all the processing steps on the 'attributes' column above, we get a total of 45 attributes and sub-attributes, which we merge to the business sub-dataset. By checking the data types and unique values of each column, we could identify invalid data values in each feature so that we can either convert the invalid values to the appropriate datatype or drop rows that contain any invalid values. Most of the extracted attributes and sub-attributes have string values ('True', 'False', 'None', or 'none'). True and False strings are converted to Python's Boolean values; and 'None' and 'none' strings are treated as Numpy's nan values. All null or nan values of the attributes and sub-attributes are considered as False Boolean data value because the missing attributes of a restaurant simply indicate that the restaurant does not have these attributes and/or sub-attributes.

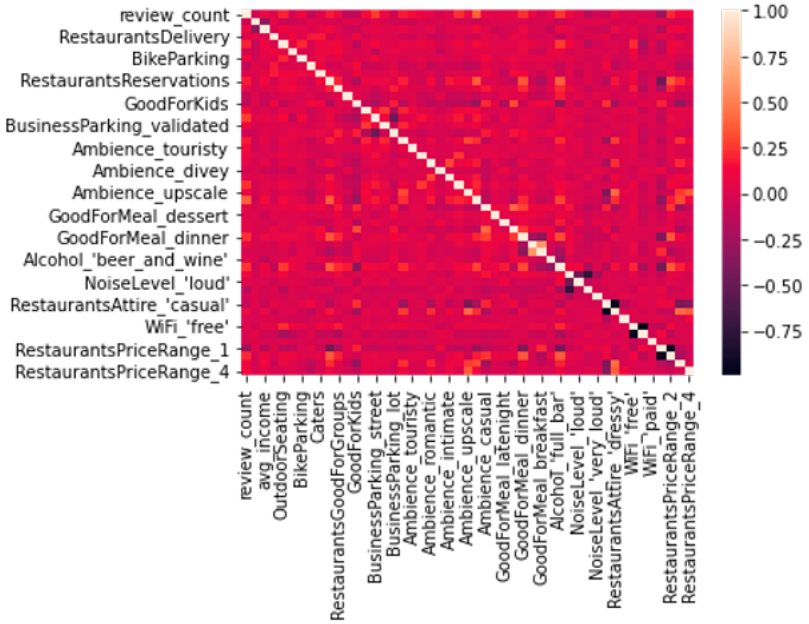



*Figure 9. Correlation Matrix Heatmap of all features in processed business dataset*

Furthermore, we conducted the correlation analysis (Figure 9) to remove highly correlated features (features with an absolute value of correlation coefficient of more than 0.5) to avoid multi-collinearity in our regression models. We excluded 5 variables: "GoodForMeal_breakfast", "NoiseLevel_'quiet'", "RestaurantAttire_'dressy'", "RestaurantPriceRange_2", and "WiFi_'no'".

The final processed business dataset has 6,265 rows (businesses) with 44 relevant features to be used in our predictive models. Finally, we exported this dataset into a CSV file to be used in the later steps of this project.

## Review Data

### Sub-setting

After getting our final processed business dataset, we subset the original review dataset to select 1,194,951 reviews for the 6,265 chosen business IDs by performing Binary Search in JSON objects. This helped reduce the size of the review data file significantly so that further text cleaning processes can be done on the review data file more easily.

### Text Review Pre-processing

First, from the reduced review dataset, we selected necessary columns which are 'business_id' and 'review_id' to identify the reviews, as well as 'text' to get text reviews. Then, we utilized the Natural Language Toolkit (nltk) module in Python to clean text reviews by converting all words to lowercase, removing stop words, numbers, url, special characters, and punctuations from text reviews, as well as handling contraction expressions, word tokenization, and stemming and lemmatizing words. Table 4 below demonstrates an example of a text review before and after all of the preprocessing steps mentioned above.

| Text WITHOUT Preprocessing | Text WITH Preprocessing |
|---|---|
| If you decide to eat here, just be aware it is going to take about 2 hours from beginning to end. We have tried it multiple times, because I want to like it! I have been to it's other locations in NJ and never had a bad experience. \n\nThe food is good, but it takes a very long time to come out. The waitstaff is very young, but usually pleasant. We have just had too many experiences where we spent way too long waiting. We usually opt for another diner or restaurant on the weekends, in order to be done quicker. | decide eat aware going take hour beginning end tried multiple time want like location nj never bad experience food good take long time come waitstaff young usually pleasant many experience spent way long waiting usually opt another diner restaurant weekend order done quicker |

*Table 4. A text review before and after preprocessing*

### Sentiment Analysis

Next, we generate polarity scores of the cleaned text. In this case, we import Sentiment Intensity Analyzer from the Natural Language Toolkit (nltk) to analyze if the text expresses a positive, negative, or neutral opinion; and at the same time, we will also receive a positive, negative, neutral, and compound sentiment score (VADER score) for each review. Since there are several

reviews for each restaurant, we choose to use the average numbers of each sentiment score (positive, negative, and neutral) to represent the overall sentiment description that the restaurant received from all of its reviews.

### Merging Review with Business Data

Finally, we merge the processed review dataset that has 4 columns (business ID and 3 sentiment scores of text reviews) with our final business dataset, and the output is a dataset (final_business_sentiment.csv) with thousands of restaurants with their information and 47 features (including review count, total household, average income, 41 attributes, and 3 sentiments) that may help us to predict those restaurants' star ratings.

## Models and Evaluation

Using Python Scikit-learn module, we have built thirteen models using five algorithms (Decision Tree Regressor, Random Forest Regressor, Gradient Boosting Regressor, Support Vector Regression (SVR), and Neural Network Multi-Layer Perceptron (MLP) Regressor) with three datasets (All Features Dataset, Reduced Features Dataset, and PCA Dataset). We divided our final processed dataset (final_business_sentiment.csv) into 80% training and 20% testing. Next, we performed k-fold cross validation for the thirteen models and conducted hyperparameter tuning on the best performance model to determine the best parameters that generate the best results. Additionally, we also used the clustering technique to group the similar datapoints into different clusters on the model that has the best performance to validate the model performance in various clusters. For the evaluation of models, we are using R-squared since it is the most popular metric to evaluate how well the regression models fit the features. Table 5 below shows the summary of the all the models we built and their performance.

| Model | Dataset | K-fold Score | MSE | RMSE | R-Squared |
|---|---|---|---|---|---|
| **Decision Tree** | All Features | 0.6440 | 0.1520 | 0.3899 | 0.6610 |
| **Gradient Boosting** | All Features | 0.8361 | 0.0731 | 0.2703 | 0.8371 |
| **Neural Network** | All Features | 0.7321 | 0.1226 | 0.3502 | 0.7266 |
| **Random Forest** | All Features | 0.8257 | 0.0766 | 0.2768 | 0.8291 |
| **SVR** | All Features | 0.8042 | 0.0843 | 0.2903 | 0.8121 |
| **Decision Tree** | PCA | 0.5045 | 0.2261 | 0.4755 | 0.4960 |
| **Gradient Boosting** | PCA | 0.7805 | 0.0954 | 0.3088 | 0.7874 |
| **Neutral Network** | PCA | 0.7171 | 0.1114 | 0.3338 | 0.7515 |
| **Random Forest** | PCA | 0.7693 | 0.1041 | 0.3226 | 0.7679 |

| | | | | | |
|---|---|---|---|---|---|
| **SVR** | PCA | 0.7915 | 0.0909 | 0.3016 | 0.7972 |
| **Decision Tree** | Reduced Features | 0.8097 | 0.0837 | 0.2893 | 0.8133 |
| **Gradient Boosting** | Reduced Features | 0.8219 | 0.0793 | 0.2815 | 0.8233 |
| **Random Forest** | Reduced Features | 0.7992 | 0.0890 | 0.2983 | 0.8017 |

*Table 5. Models and Evaluation Summary*

## All Features Models

There are five models built using all features dataset (final_business_sentiment.csv) with 47 features that consist of business attributes data, business review data, income data, and user reviews' sentiment scores. We have used the default parameters on all the five models using the five algorithms mentioned above. Our results showed that the Gradient Boosting model generated the best performance with 83.71% of R-squared. This means that the Gradient Boosting model explained about 84% of the variability of all features. The worst model is the Decision Tree model with 66.1% R-squared, which is the lowest among all models. Random Forest and SVR models both have greater than 80% of R-squared while Neural Network, which is the second worst model, has 72.65% of R-squared.

## Reduced Features Models

We have also applied the feature importance algorithm to the optimal features for the tree-based models which are the Decision Tree Regressor, Random Forest Regressor, and Gradient Boosting Regressor using the default parameters. We selected the features based on the importance threshold that delivered the best results. The number of selected features for the three models are in the range between 5 to 7 features out of 47 features. The best model using reduced features dataset is Gradient Boosting model with an R-squared of 82.33%, and the worst model is Random Forest with 80.17% R-squared. Overall, the tree-based models using the reduced features dataset have greater than 80% of R-squared.

## PCA Models

The other technique we used to identify the important features is Principal Component Analysis (PCA). This method also helps reduce the dataset dimensionality without changing the dataset pattern. In PCA, we have selected 30 number of components / features that explained greater than 80% of the variability of the dataset based on the scree plot (Figures 10 & 11). With the PCA dataset, we have built five models using the default parameters with the five algorithms previously mentioned. The best model the ran on the PCA dataset is the SVR model with 79.72% of R-squared, whereas the worst model is Decision Tree with only 49.6% of R-squared. The overall R-squared from PCA models is shown to have weakest performance as compared to all features models and reduced features models.

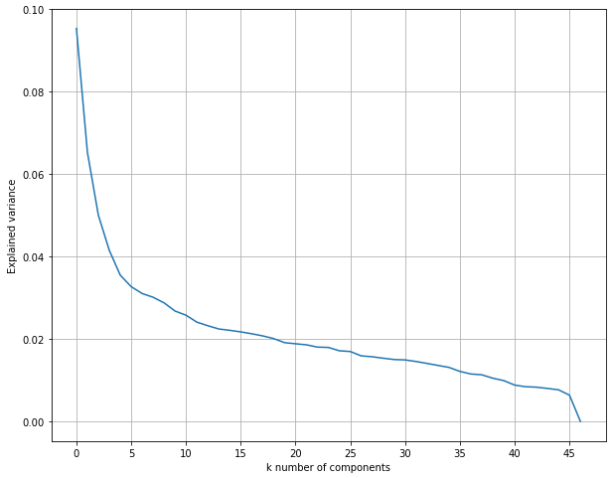



*Figure 10. PCA scree plot #1*

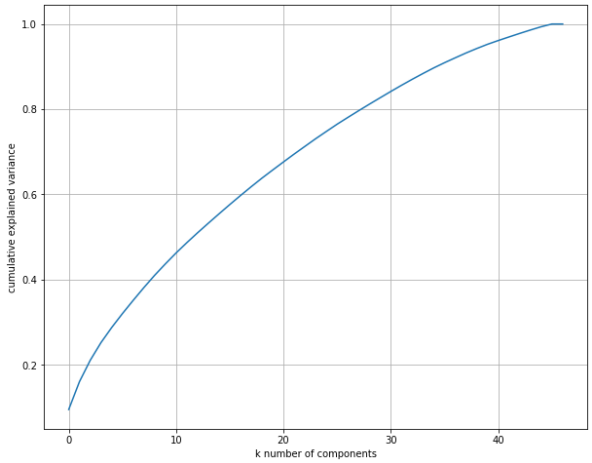



*Figure 11. PCA scree plot #2*

## K-Fold Cross Validation

K-fold cross validation is a re-sampling technique to determine how effective the model can apply to a new or unseen dataset. This technique splits the dataset into k number of subsets and evaluates the overall performance metrics. In our project, we have tested on five number of subsets (k=5) for all algorithms on all features, reduced features, and PCA dataset. Besides, we also tested ten (k=10) and fifteen (k=15) number of subsets for Gradient Boosting all features model, which is the model yielding the highest R-squared. Note that we are using the default parameters on all models. As a result, the best model after applying k-fold cross validation is Gradient Boosting all features model with 83.61% of R-squared, slightly lower than without applying this technique. On the other hand, the worst model is Decision Tree PCA model with 50.45% of R-squared.

## Hyperparameter Tuning

In order to improve the model to generate an optimal result, we have applied the hyperparameter tuning technique on Gradient Boosting all features dataset that has previously resulted in the highest R-squared. We used GridSearchCV and RandomizedSearchCV to find the combination of parameters that resulted in the best performance among a total of 300 possible combinations. We wanted to try more parameter, which means searching through a larger number of models, but due to time constraint given to conduct this project, we could not succeed. The best parameters after performing GridSearchCV are {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 100, 'subsample': 1.0}, which are also the default parameters. The R-Squared for Gradient Boosting all features model after GridSearchCV is 83.6%.

## Clustering

In this project, we also applied the K-means Clustering algorithm to aggregate datapoints together based on their characteristics and similarities. Before the clustering, datapoints are being normalized since each feature has varying scales and units. This would help to standardize the datapoints and ensure that the distance and weight are measured equally when applying K-means Clustering. We have selected 3 clusters on All Feature dataset based on the elbow method (Figure 12) that showed us the optimal number of clusters. Then, we run Gradient Boosting Regressor with the parameters determined from the hyperparameter tuning. Our results showed

that the R-squared for all three clusters did not give us better results as compared to the original All Feature dataset on Gradient Boosting algorithm. The best R-squared was from Cluster 2 with 78.47% while the worst R-squared was from Cluster 1 with 68.31%.

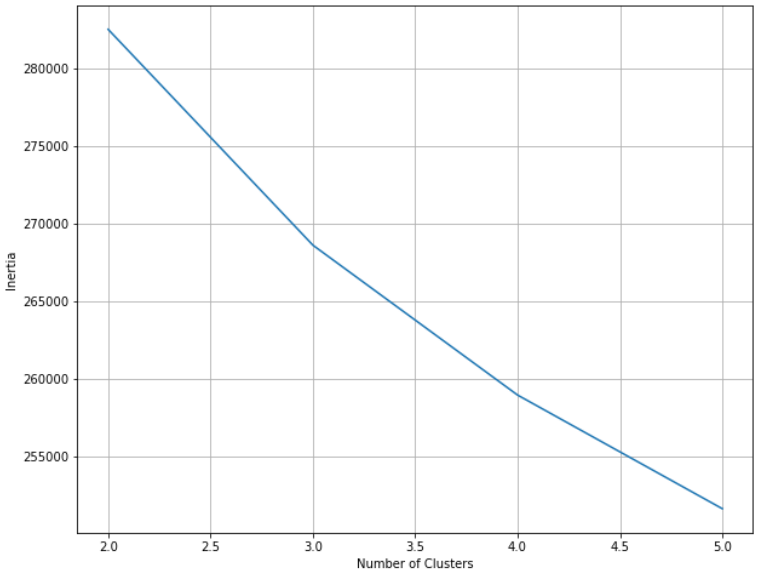



*Figure 12. K-Means Clustering Plot*

## Interpretations

After running all five algorithms on the three datasets using various machine learning techniques, we have determined that Gradient Boosting algorithm is the best model since it consistently showed good results on all three datasets. Gradient Boosting all features and reduced features models have ranked as the first and third out of all models in terms of R-squared (both are greater than 82%). Gradient Boosting PCA model is ranked as the second-best model out of all PCA models.

On the other hand, Decision Tree Regressor yields the worst results on all features and PCA models, in which R-squared are 51.86% and 66.1% respectively, being the lowest among all. Surprisingly, Decision Tree reduced features model based on the feature importance helps to increase the R-squared significantly to 81.33% with seven features selected. This also showed that the algorithm is unstable when a small change can lead to a major change in performance. Additionally, Neural Network MLP Regressor has the second worst performance after both all features and PCA dataset's Decision Tree.

Besides, we realized that PCA technique is not helpful in improving the model performance in our project. More specifically, the models using PCA dataset generally are at the bottom in performance ranking with lower than 80% R-squared. The feature importance technique on tree-based models also told us that the most frequently selected features are the sentiment scores (positive, negative, and neutral) and review counts. This could imply that the user reviews can significantly influence the business star ratings.

In terms of computation time, Random Forest PCA model generally took the longest to run (about 11 seconds) while the worst models – all features and PCA Decision Tree models and

Gradient Boosting reduced features model took less than a second to run. In conclusion, Gradient Boosting Regressor seems to have both good results and quick run time.

# Discussion

## Domain Knowledge

With most of the models' R-squared being greater than 70%, it has proven that our features have high predictive power in predicting the business star ratings. Specifically, the feature importance technique applied on tree-based models showed that **sentiment scores and review counts** can significantly influence the business star rating. Sentiment scores allow business owners to understand about customer's behavior and feedback.

With these insights, Yelp business owners could use the user reviews to predict their business's star ratings by understanding how users expressing on their products & services without having user to give star rating for each review. To have more reviews for better prediction, Yelp business owners could encourage user to leave review after purchasing such as giving out promotions or having referral program.

In our study, we also have business attributes such as Alcohol Full Bar, Business Parking Street, Business Price Range, etc. Having some specific business attributes could help in predicting business star rating. For example, Alcohol Full Bar is shown as one of the most important features. The reason could be because our dataset includes Florida restaurants, and Florida is famous with bars and nightlife. Thus, Yelp business owners can update the business attributes accordingly in order to generate a higher rating prediction.

Same as the average household income, business star rating can be higher or lower in different locations with different income. Our study is beneficial for Yelp business owners in better understanding the features that can greatly influence the business star rating.

## Methodological Contributions

The original Yelp datasets are stored in JSON format since the size of the files is large. Considering the huge number of rows and columns, especially the review file, it still took a lot of time to read the files in the program. Therefore, we had to do deep research on how to handle the file more efficiently. We found out that, instead of reading the files directly into Pandas' Data Frame, we read them into JSON items, which is stored as a list of dictionaries in the memory, so it is smaller in size than Data Frame. Besides, our Binary Search in JSON objects reduced the time to subset the data files significantly compared to iterating rows of a Data Frame to find and select the reviews for the chosen business IDs. Moreover, the 'attributes' column in the raw data file is stored as a nested string format. This also required thorough processing and took a long time to treat the format and split them into columns. Our code is generalized so that it can be used to process any variable that have a similar structure as the 'attributes' column. For example, we reused the code to get the sub-attributes as discussed in Data Pre-processing section.

When pre-processing the review text, it took us about an hour or more to clean the review text and generate the sentiment polarity scores. After filtering out the open restaurants in Pennsylvania and Florida, we have almost 1.2 million text reviews. It is time consuming to process the review text as the computer must run for each of the review text. Besides, there are

also many contraction expressions for the auxiliary verbs such as “wasn’t”, “can’t”, and “didn’t” that are the most common contraction in the non-formal conversation. By transforming the contraction expressions into formal auxiliary verbs such as “was not”, “cannot”, and “did not” could help to the computer to better analyze the polarity of review text in our research but we must sacrifice the run time in text pre-processing stage.

Furthermore, PCA is one of the statistical techniques to identify important features from the large dataset. However, this method does not meet our expectation to see improvements in models’ performance. From our research, the R-squared of PCA models are ranked at the bottom and performed worse than other models using all features and reduced features set. In addition, the results after applying various statistical techniques such as feature importance, k-fold cross validation, clustering, and PCA are not as good as the original all features dataset. This can be due to the dataset provided by Yelp since Yelp might already have processed and selected the useful features for public research.

# Conclusion

## Summary

Throughout our research, we have experimented with the prediction of business star rating with various features using five different machine learning algorithms on three datasets. We learned that the user review text, specifically the sentiment polarity scores are the most important features in predicting the business star rating. There are also other features such as business attributes and average household income that contribute to the prediction but may vary depending on the business category and location. As for the machine learning models, our evaluation metric R-squared has been consistently high (>70%) for most of the models. The best performance is the Gradient Boosting All Features model while the worst performance is the Decision Tree PCA model. We also do not recommend Decision Tree algorithms for this research since it yields very unstable results in different dataset. Lastly, the models with all features dataset somehow produced better results than the models after selecting important features. We believe by further enhancing the machine learning models using other statistical techniques and modifications may help to improve the performance in the future.

## Limitations

The first limitation in our study is that we only included open restaurants in Pennsylvania and Florida in the dataset. The data points’ correlations and models’ prediction are derived from a particular set of restaurants only. Thus, the high model performance is only beneficial to these restaurants and may not have the same results in other states and other business categories.
The second limitation is that we thought that it could be more accurate and interesting to predict the business star rating using the review text vectors and to identify the words that are important in the prediction. By vectorizing the user review text, we have generated more than 300,000 word vectors. Our ideal approach was to merge the processed business dataset (final_business.csv) with the vectors.

To do this, we must transform the vectors from matrix to Pandas’ Data Frame then aggregate the TFIDF scores using group by function on business_id. However, we have encountered memory error when converting from matrix to Pandas Dataframe due to the huge number of vectors. The error continues to occur even when we reduced the max_features within the vectorizer function

to 10,000 and 1,000. Therefore, we decided to do sentiment analysis and utilize the sentiment polarity scores to represent text reviews for our prediction models.

### Future Projects

We are still very interested in finding out the effect of review text vectors combined with business attributes on business star rating prediction. As discussed in the Limitations section, we were unable to handle the large number of vectors with our machines. Future extended research can be done to solve the vectorization problems by using a more powerful machine to handle the more than 300,000 vectors that may contribute to the crowdsource review platform industry. By doing this, we can also determine if any specific words from the reviews are more important and can significantly improve the model performance.

In addition, Yelp originally published five datasets and we only utilized business and review dataset for our research. To understand more about the effect of other features, other datasets (e.g., user, tip, and check-in) can be useful in providing more insights. For example, user dataset consists of the length of period since users joined Yelp, number of friends, fans, and total votes a user receives. And tips generally have shorter word count than user review and can reduce the number of word vectors. All these data can help the industry to understand more broadly about how these features can contribute to achieve the goal to predict business star rating. Moreover, the external datasets can also be included in the future projects such as events, holiday, promotions, and inflation rate across the U.S. since these factors can also influence consumers' opinions about restaurants or other types of businesses.

## References


[1] Yelp, Inc. and C. Crawford (2022). *Yelp Dataset*. Kaggle. Retrieved October 4, 2022, from https://www.kaggle.com/datasets/yelp-dataset/yelp-dataset?select=yelp_academic_dataset_user.json

[2] Pitman, J. (2022). *Local consumer review survey 2022: Customer reviews and behavior*. BrightLocal. Retrieved December 6, 2022, from https://www.brightlocal.com/research/local-consumer-review-survey/

[3] Feldman, E. (2015, January 18). *Why the restaurant industry is the most important industry in today's America*. Medium. Retrieved October 11, 2022, from https://medium.com/@EliFeldman/why-the-restaurant-industry-is-the-most-important-industry-in-todays-america-6a819f8f0ac9

[4] Andre, L. (2022, November 5). *78 Yelp Statistics You Must Know: 2022 Market Share & User Profile Analysis*. Retrieved December 6, 2022, from https://financesonline.com/yelp-statistics/

[5] Sharma, D. (n.d.). *Identifying Influential Factors for Yelp Business Ratings*. Retrieved December 5, 2022, from https://cseweb.ucsd.edu/classes/wi15/cse255-a/reports/fa15/010.pdf

[6] Singh, R. and Woo, J. (2019). *Applications of Machine Learning Models on Yelp Data*. Retrieved October 10, 2022, from http://www.apjis.or.kr/pdf/02_%EC%9A%B0%EC%A2%85%EC%9A%B1(35-49).pdf

[7] Xu, Y., Wu, X., & Wang, Q. (2014). *Sentiment Analysis of Yelp's Ratings Based on Text*. Retrieved October 10, 2022, from http://cs229.stanford.edu/proj2014/Yun%20Xu,%20Xinhui%20Wu,%20Qinxia%20Wang,%20Sentiment%20Analysis%20of%20Yelp's%20Ratings%20Based%20on%20Text%20Reviews.pdf

[8] Channapragada, S. & Shivaswamy, R. (2015). *Prediction of Rating Based on Review Text of Yelp Reviews*. Retrieved December 5, 2022, from https://cseweb.ucsd.edu/classes/wi15/cse255-a/reports/fa15/031.pdf

[9] Fan, M. and Khademi, M. (2014, January 5) *Predicting a Business Star in Yelp from Its Reviews Text Alone*. Retrieved October 10, 2022, from https://arxiv.org/ftp/arxiv/papers/1401/1401.0864.pdf

[10] Asghar, N. (2016, May 17) *Yelp Dataset Challenge: Review Rating Prediction*. Retrieved October 10, 2022, from https://arxiv.org/pdf/1605.05362.pdf

[11] United States Census Bureau (2020). *Income and Poverty Data in Pennsylvania and Florida*. Retrieved October 31, 2022, from https://data.census.gov/table?t=Income+and+Poverty&g=0400000US12$8600000,42$8600000&y=2020&d=ACS+5-Year+Estimates+Subject+Tables&tid=ACSST5Y2020.S1901

# Appendices

## A. Data dictionary

***Attached file:* Data Dictionary.xlsx**

### A-1. Yelp Business Dataset

| Name | Description | Valid Domain Values | Data Type | Length | Example Values | Data Attributes | Null Ratio |
|---|---|---|---|---|---|---|---|
| **business_id** | Business ID | | String | 22 | Pns2l4eNsfO8kk83dixA6A | Nominal | 0.00 % |
| **name** | Business Name | | String | 66 | Abby Rappoport, LAC, CMQ | Nominal | 0.00 % |
| **address** | Business Address Street | | String | 120 | 1616 Chapala St, Ste 2 | Nominal | 0.00 % |
| **city** | Business Address City | | String | 50 | Santa Barbara | Nominal | 0.00 % |
| **state** | Business Address State | State Abbreviations: XX | String | 2 | CA | Nominal | 0.00 % |
| **postal_code** | Business Address Postal Code | US: ##### Canada: X#X #X# | String | 5<br><br>7 | 93101<br><br>T5J 5A3 | Nominal | 0.00 % |
| **latitude** | Business Address Latitude | | Float | 14 | 34.426679 | Interval | 0.00 % |
| **longitude** | Business Address Longitude | | Float | 14 | -119.711197 | Interval | 0.00 % |
| **stars** | Star rating, rounded to half-stars | Ranges: 1.0 - 5.0 | Float | 3 | 5.0 | Ratio | 0.00 % |
| **review_count** | Number of reviews | | Integer | 4 | 7 | Ratio | 0.00 % |
| **is_open** | 0 or 1 for closed or open, respectively | 1 = Open 0 = Close | Integer | 1 | 1, 0 | Binary | 0.00 % |
| **attributes** | Business attributes to values. Note: some | {Attribute 1:True/False, Attribute | String | 33 | {'ByAppointmentOnly': 'True'} | Nominal | 9.14 % |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | attribute values might be objects | 2:True/False, etc.} | | | | | |
| **categories** | Business Categories | Category 1, Category 2, etc. | String | 503 | Doctors, Traditional Chinese Medicine, Naturopathic/Holistic, Acupuncture, Health & Medical, Nutritionists | Nominal | 0.07 % |
| **hours** | Key day to value hours, hours are using a 24hr clock | {Day of week: HH:MM-HH:MM} | String | 7 | {'Monday': '0:0-0:0', 'Tuesday': '8:0-18:30', 'Wednesday': '8:0-18:30', 'Thursday': '8:0-18:30', 'Friday': '8:0-18:30', 'Saturday': '8:0-14:0'} | Interval | 15.45% |

### A-2. Yelp Review Dataset

| Name | Description | Valid Domain Values | Data Type | Length | Example Values | Data Attributes | Null Ratio |
|---|---|---|---|---|---|---|---|
| **review_id** | Review ID | | String | 22 | KU_O5udG6zpxOg-VcAEodg | Nominal | 0.0% |
| **user_id** | User ID | | String | 22 | mh_-eMZ6K5RLWhZyISBhwA | Nominal | 0.0% |
| **business_id** | Business ID | | String | 22 | XQfwVwDr-v0ZS3_CbbE5Xw | Nominal | 0.0% |
| **stars** | Star Rating | Ranges: 1 – 5<br><br>Increment = 0.5 | Float | 3 | 3.0 | Ratio | 0.0% |

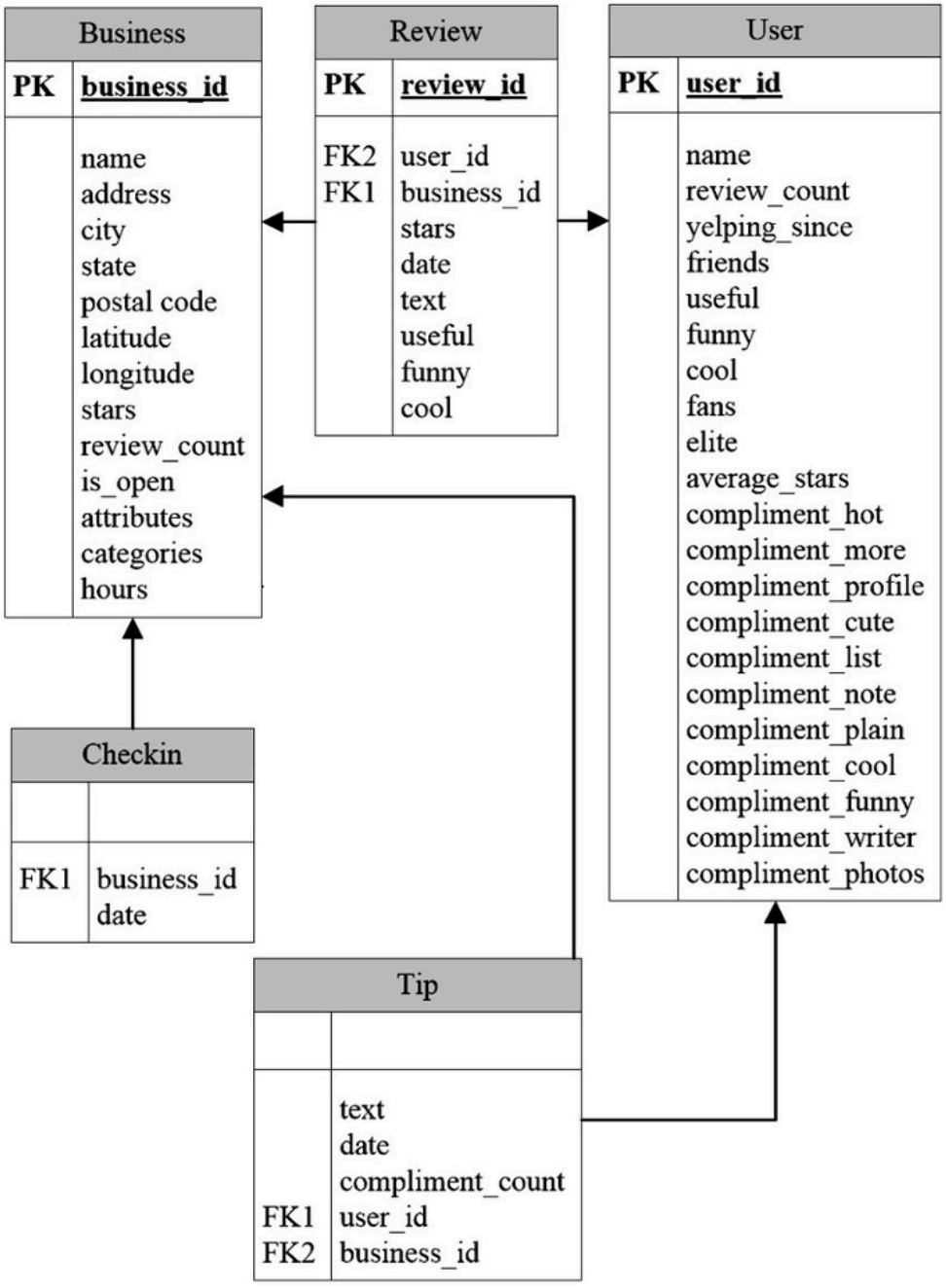
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **useful** | Number of useful votes received | 0 - 1182 | Integer | 4 | 3 | Ratio | 0.0% |
| **funny** | Number of useful funny received | 0 - 792 | Integer | 3 | 4 | Ratio | 0.0% |
| **cool** | Number of useful cool received | 0 - 404 | Integer | 3 | 5 | Ratio | 0.0% |
| **date** | Reviewed Date | YYYY-MM-DD HH:MM:SS | Date | 10 | 2018-07-07 22:09:11 | Interval | 0.0% |
| **text** | Review Text | | String | | Super simple place but amazing nonetheless. It's been around since the 30's and they still serve the same thing they started with: a bologna and salami sandwich with mustard. \n\nStaff was very helpful and friendly. | Nominal | 0.0% |

### A-3. U.S. Census Income Dataset

| Name | Description | Valid Domain Values | Data Type | Length | Example Values | Data Attributes | Null Ratio |
|---|---|---|---|---|---|---|---|
| **full_zip_code** | Geographic Area Name | ZCTA5 ##### | String | 11 | ZCTA5 32133 | Nominal | 0% |
| **total_hh** | Estimate Total House holds | 0 - 31260 | Integer | 5 | 11093 | Nominal | 0% |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **avg_income** | Estimate Mean Household Income | 13196 - 409434 | Integer | 6 | 119164 | Nominal | 4% |

## B. Entity Relationship Diagram (ERD) of Yelp Database



## C. Python Code

***Attached folder:* Team Yelp – All Code + Data**

NOTE: The attached folder is submitted under the compressed zip file format (**Team Yelp – All Code + Data.zip**)

### C-1. Business Pre-processing

```
# %% [markdown]
# ## business.json: Contains business data including location data, attributes, and
categories.

# %%
# Import relevant libraries:
import json
import csv
import pandas as pd
import numpy as np #linear algebra
import gc #garbage collection - keep track of objects in memory
import re #regular expressions

#plotting
import matplotlib.pyplot as plt
import seaborn as sns

pd.set_option('display.max_columns', None)

def ReadJSONtoDF(file_name): #read JSON file and convert to pandas DF
    data_file = open(file_name,encoding="utf8")
    data_list = []
    for line in data_file:
        data_list.append(json.loads(line))
    DataDF = pd.DataFrame(data_list)
    data_file.close()
    return DataDF

def null(dataframe): #generate null count summary by descending number
    null_df = pd.DataFrame(dataframe.isnull().sum()).reset_index() #count null values
    null_df = null_df.rename(columns={'index':'colnames', 0:'null'})
    null_df['null_ratio%'] = round(null_df['null']/dataframe.shape[0]*100, 4)
#calculate null ratio
    null_df = null_df.sort_values(by='null_ratio%', ascending=False)
    return null_df

# %%
business_df = ReadJSONtoDF("yelp_academic_dataset_business.json")
business_df.head(5)
```

```
# %%
business_df.rename({"name": "business_name"}, axis = "columns", inplace = True) #
rename name column to be more explicit

# %%
# show number of rows & columns
business_df.shape

# %%
## Sub setting data to include data with restaurants and that are open.
## Source: https://www.kaggle.com/code/virajkadam/yelp-restaurants-eda-and-data-
preparation

# finding restaurants businesses
business_df['restaurant']= (business_df.categories.str.contains(pat='Restaurant',
flags=re.IGNORECASE, regex=True))
# business_df.loc[business_df.restaurant==True]

required_columns = ['business_id','business_name', 'city', 'state', 'postal_code',
'stars', 'categories', 'review_count', 'attributes']  # columns to use

# filtering open restaurants:
restaurants = business_df[(business_df.restaurant==1) &
(business_df.is_open==1)].reset_index(drop=True)

# keeping required cols
restaurants=restaurants.loc[:,required_columns]

restaurant_list = restaurants.business_id.unique()   # list of restaurants ids

print(f'There are {len(restaurants):,} Restaurants(That are Open) in the Dataset')

# %%
# Filter restaurants in PA & FL
restaurants_reduced_DF = restaurants[(restaurants['state'] == 'PA') |
(restaurants['state'] == 'FL')].reset_index(drop=True)
restaurants_reduced_DF

# %% [markdown]
# #### Save DFs as csv files

# %%
# Saving DF as csv
restaurants.to_csv('restaurants.csv',index=False)
restaurants_reduced_DF.to_csv('restaurants_reduced.csv',index=False)

# %% [markdown]
# ## Restaurants in PA and FL Data (Subset from EDA):
```

```
# %%
#### SUBSETTING
restaurants_reduced_DF = pd.read_csv("restaurants_reduced.csv", sep=',', header=0)

# %%
# check data types:
restaurants_reduced_DF.info() # re-run after processing nulls and change data type of
zip code below

# %%
# check number of null values:
null(restaurants_reduced_DF) # re-run after processing nulls and change data type of
zipcode below

# %%
restaurants_reduced_DF.dropna(axis=0, inplace=True) # dropping null values to avoid
errors

# %%
# postal_code values are in float; convert postal_code to 'object' type:
restaurants_reduced_DF['postal_code'] =
restaurants_reduced_DF['postal_code'].astype(int).astype(str)

# %% [markdown]
# ## Income Data (from Census):

# %%
income_DF = pd.read_csv("income data.csv", sep=',', header=0)

# %%
income_DF.info()

# %%
# rename mean columns to be shorter and easier to understand:
income_DF.rename({"Geographic Area Name": "full_zip_code",
                  "Estimate!!Households!!Mean income (dollars)": "avg_income",
                  "Estimate!!Households!!Total": "total_hh"}, axis = "columns",
inplace = True)

# %%
income_DF[(income_DF["avg_income"] == '-') | (income_DF["avg_income"] == 'N')] #find
rows where avg_income is not a number (106 rows total)
                                                                    #not
dropping them at the moment; drop after merged instead!

# %%
```

```
income_DF[['ZCTA', 'postal_code']] = income_DF['Geographic Area Name'].str.split(' ',
1, expand=True) #split area name to get postal codes

# %%
# drop unnecessary columns
income_DF = income_DF.drop(['Geographic Area Name', 'ZCTA'], axis=1)

# %%
income_DF

# %% [markdown]
# ## Merge income with restaurants:

# %%
# left outer join
restaurants_with_income = pd.merge(restaurants_reduced_DF, income_DF,
on='postal_code', how='left')

# %%
restaurants_with_income

# %%
restaurants_with_income.info()

# %%
# make sure a zipcode has similar avg_income:
restaurants_with_income[restaurants_with_income['postal_code'] == '19107']

# %%
# number of rows with income = '-':
print("There are {} lines with no total households, so no income (income = '-').". \
    format(len(restaurants_with_income[restaurants_with_income["avg_income"] == '-
']))))

# number of rows with income = 'N':
print("\n There are {} lines having income values that are not a number (income =
'N').". \
    format(len(restaurants_with_income[restaurants_with_income["avg_income"] ==
'N'])))

# Treat '-' and 'N' income values as nulls:
restaurants_with_income.avg_income = restaurants_with_income.avg_income.replace(['-
','N'], np.nan) #there will be 45 nulls added to avg_income column

restaurants_with_income.info() #avg_income non-nulls decreased by 45 (from 13,769 to
13,724)

# %%
```

```
# drop all rows that have missing avg_income:
restaurants_with_income.dropna(axis=0, inplace=True)

# %%
# convert total_hh to int type:
restaurants_with_income[['total_hh', 'avg_income']] =
restaurants_with_income[['total_hh', 'avg_income']].astype(int)

# %%
restaurants_with_income.info()

# %%
# save DF as csv:
restaurants_with_income.to_csv('restaurants_with_income.csv',index=False)

# %% [markdown]
# ## Processing 'attributes':

# %%
DataDF = pd.read_csv("restaurants_with_income.csv", sep=',', header=0)
DataDF

# %%
DataDF['attributes'][0]

# %% [markdown]
# #### Extract Attributes:

# %%
import ast

def extract_attributes(column):
    attribs = column.apply(lambda x: ast.literal_eval(x)) #convert string to
dictionary
    attribs = attribs.apply(pd.Series)
    return attribs

# %%
# get all attributes:
all_attributes = extract_attributes(DataDF['attributes'])
all_attributes

# %%
null(all_attributes) # check null values in all attributes

# %%
# drop columns with more than 50% null values:
```

```
all_attributes.dropna(thresh = int(0.5 * all_attributes.shape[0]), axis = 1, inplace =
True)

# %%
DataDF = pd.concat([DataDF, all_attributes], axis = 1)
DataDF.drop('attributes', axis=1, inplace=True) #drop attributes' columns that are
already split

# %% [markdown]
# #### Extract Sub-attributes:

# %%
# further split sub-attributes into their own columns:
cols_to_split = ['BusinessParking', 'Ambience', 'GoodForMeal']
DataDF = DataDF.dropna(axis=0, subset=cols_to_split).reset_index(drop=True) #drop null
values in the cols that need further splitting to avoid errors

# %%
# RUN THIS ONLY ONCE!!!! OTHERWISE IT WILL DOUBLE CONCAT COLUMNS!!!!
for col in cols_to_split:
   sub_attributes = extract_attributes(DataDF[col]) #split sub-attributes
   # rename sub-attributes cols to have their attribute prefixes:
   column_names = { sub_col : col + "_" + sub_col for sub_col in
sub_attributes.columns }
   sub_attributes = sub_attributes.rename(columns=column_names)
   DataDF = pd.concat([DataDF, sub_attributes], axis = 1)

DataDF.drop(cols_to_split, axis=1, inplace=True) #drop attributes' columns that are
already split

# Covnert string True/False to boolean values:
booleanDictionary = {'True': True, 'False': False}
DataDF = DataDF.replace(booleanDictionary)
DataDF

# %% [markdown]
# While most of these new columns we just made are boolean attributes, a number of
them are categorical, which need some special handling. We need to encode them so
models can use them as features.

# %%
# check DataDF column names:
DataDF.columns

# %%
DataDF.rename({"RestaurantsPriceRange2": "RestaurantsPriceRange"}, axis = "columns",
inplace = True)
```

```
# %% [markdown]
# #### Convert categorical columns to dummy variables:

# %%
# columns with non-boolean categorical values:
cat_cols_to_split = ['Alcohol', 'NoiseLevel', 'RestaurantsAttire', 'WiFi',
'RestaurantsPriceRange']

DataDF = DataDF.dropna(axis=0, subset=cat_cols_to_split).reset_index(drop=True) #drop
null values in the cols that need further splitting to avoid errors
DataDF

# %%
# Some str values contains "u"" at the beginning:
def clean_string(a_string):
    return a_string[1:] #remove the unwanted characters

for col in cat_cols_to_split:
    DataDF[col] = DataDF[col].apply(lambda x : clean_string(x) if x.startswith("u")
else x)

# %%
# check unique values:
for col in cat_cols_to_split:
    print({col: list(DataDF[col].unique())})

# %%
for col in cat_cols_to_split:
    DataDF[col] = DataDF[col].replace('None', "'none'",regex=True) #convert none
values to "'none'"

# %%
new_cat = pd.concat([pd.get_dummies(DataDF[col], prefix=col, prefix_sep='_') for col
in cat_cols_to_split], axis=1)
# dropping 1 dummy column of each category: (drop columns with names ends with
_'none')
new_cat = new_cat.loc[:,~new_cat.columns.str.endswith("_'none'")]

DataDF = pd.concat([DataDF, new_cat], axis=1) #merge to main data
DataDF.drop(cat_cols_to_split, inplace=True, axis=1)

# %%
DataDF

# %%
# Convert strings None or none to NaN values:
noneDictionary = {'None': np.nan, 'none': np.nan}
DataDF = DataDF.replace(noneDictionary)
```

```
DataDF

# %%
for col in list(DataDF.columns[10:]):
    print({col: list(DataDF[col].unique())})

# %%
# fill None or nan values as False (missing attributes of a restaurant indicate that
the restaurant does not have these attributes):
DataDF.fillna(value=False, inplace=True)

# %%
null(DataDF) #check number of nulls in the dataset

# %%
DataDF.columns

# %%
DataDF.info()

# %%
# save DF as csv:
DataDF.to_csv('processed_businesses.csv',index=False)

# write csv to json
with open('processed_businesses.csv') as f:
    reader = csv.DictReader(f)
    rows = list(reader)

with open('processed_businesses.json', 'w') as f:
    json.dump(rows, f)

# %% [markdown]
# ## Correlation Analysis

# %%
businesses = pd.read_csv("processed_businesses.csv", sep=',', header=0)
businesses

# %%
businesses.info() #check nulls and data types

# %%
all_raw_features = businesses.iloc[: , 7:]
corr_matrix = all_raw_features.corr()
abs_corr = corr_matrix.abs()
abs_corr.unstack().sort_values(ascending=False).drop_duplicates()
```

```
# %%
sns.heatmap(corr_matrix);

# %%
# Creating the Correlation matrix and Selecting the Upper trigular matrix:
upper_tri = abs_corr.where(np.triu(np.ones(abs_corr.shape),k=1).astype(np.bool))
print(upper_tri)

# %%
to_drop = [column for column in upper_tri.columns if any(upper_tri[column] > 0.5)]
print(); print(to_drop)

# %%
businesses = businesses.drop(to_drop, axis=1) #drop high correlated columns

# %%
businesses

# %%
intDictionary = {True: 1, False: 0} #convert boolean values to binary values (0 and 1)
businesses = businesses.replace(intDictionary)

# %%
# save DF as csv:
businesses.to_csv('final_businesses.csv',index=False)

# write csv to json
with open('final_businesses.csv') as f:
    reader = csv.DictReader(f)
    rows = list(reader)

with open('final_businesses.json', 'w') as f:
    json.dump(rows, f)
```

### C-2. Review Pre-processing

```
# %% [markdown]
# #### review.json: Contains full review text data including the user_id that wrote
the review and the business_id the review is written for.

# %% [markdown]
# ## Subsetting:

# %%
import json
import csv
import pandas as pd
```

```
import numpy as np
import gc
import re

#plotting
import matplotlib.pyplot as plt
import seaborn as sns

pd.set_option('display.max_columns', None)
pd.set_option('display.max_colwidth', None)

def ReadJSONitems(file_name, within_list=False):
    items = []
    with open(file_name, encoding='utf8') as read_file:
        if not within_list:
            for row in read_file:
                items.append(json.loads(row))
        else:
            items = json.load(read_file)
    return items

def WriteJSON(file_name, data):
    with open(file_name, "w") as out_file:
        json.dump(data, out_file)

def PropFilter(items, prop, approved_values):
    new_list = []
    for item in items:
        if item[prop] in approved_values:
            new_list.append(item)
    return new_list

def GetPropList(items, prop):
    return [item[prop] for item in items]

def binary_search(x, lyst, l, r):
    if l > r:
        return False
    while l <= r:
        mid = (l+r) >> 1
        if lyst[mid] == x:
            return True
        elif lyst[mid] < x:
            l = mid + 1
        else:
            r = mid - 1
    return False
```

```
# %%
reviews = ReadJSONitems('yelp_academic_dataset_review.json')
reviews

# %%
restaurants_reduced_list = ReadJSONitems('processed_businesses.json')[0]
restaurants_reduced_list

# %%
# Get business_id of restaurants in the chosen list
reduced_restaurants_ids = sorted(GetPropList(restaurants_reduced_list, 'business_id'))
#sort ids
reduced_restaurants_ids

# %%
len(reduced_restaurants_ids)

# %%
# get all reviews for the restaurants in the chosen list:
reduced_reviews_list = []
for review in reviews:
    if binary_search(review['business_id'], reduced_restaurants_ids, 0,
len(reduced_restaurants_ids)-1):
        reduced_reviews_list.append(review)

# %%
print(f'Nuber of reduced reviews: {len(reduced_reviews_list):,}'.format())

# %%
# Write to new file:
WriteJSON("review_restaurants_reduced.json", reduced_reviews_list)

##############################################################################

# %% [markdown]
# ## Reading the reduced review dataset and cleaning:

# %%
reduced_reviews_list = ReadJSONitems('review_restaurants_reduced.json')[0]

# %%
reduced_review_df = pd.DataFrame(reduced_reviews_list)

# %%
review_text.head()

# %% [markdown]
# #### Text Preprocessing
```

```
# %%
import string
import nltk
import re
nltk.download('stopwords')
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from wordcloud import WordCloud, STOPWORDS
from sklearn.feature_extraction.text import TfidfVectorizer
from nltk.sentiment import SentimentIntensityAnalyzer
from tqdm.notebook import tqdm
from sklearn.model_selection import train_test_split

# %%
def clean_text(text):

    text = text.lower() # convert to lower case

    text = text.replace('\n', ' ')
    text = text.replace('wasn\'t', 'was not')
    text = text.replace('weren\'t', 'were not')
    text = text.replace('won\'t', 'will not')
    text = text.replace('wouldn\'t', 'would not')
    text = text.replace('can\'t', 'cannot')
    text = text.replace('couldn\'t', 'could not')
    text = text.replace('don\'t', 'do not')
    text = text.replace('didn\'t', 'did not')
    text = text.replace('doesn\'t', 'does not')
    text = text.replace('isn\'t', 'is not')
    text = text.replace('hasn\'t', 'has not')
    text = text.replace('haven\'t', 'have not')
    text = text.replace('hadn\'t', 'had not')
    text = text.replace('shouldn\'t', 'should not')
    text = text.replace('aren\'t', 'are not')
    text = text.replace('ain\'t', 'am not')
    text = text.replace('\'s', '')
    text = text.replace('i\'m', 'i am')
    text = text.replace('i\'ve', 'i have')
    text = text.replace('i\'ll', 'i will')
    text = text.replace('we\'ve', 'we have')
    text = text.replace('i\'d', 'i would')
    text = text.replace('you\'re', 'you are')

    text = re.sub(r'http\S+', '', text) ## remove url
    text = re.sub('\w*\d\w*', '', str(text))  ## remove numbers
    text = re.sub('[^\w\s]', '', text)  ## remove punctuation
```

```
    tokens = word_tokenize(text) ## tokenize words

    stop_words = stopwords.words('english')
    stop_words = set(stop_words)
    words = [w for w in tokens if not w in stop_words] ## remove stop words

    lemmatizer = WordNetLemmatizer()
    text = [lemmatizer.lemmatize(w) for w in words] ## lemmatize words

    text = " ".join(w for w in text)

    text = text.replace('wa ', ' ')
    text = text.replace('ha ', ' ')

    return text


clean_df = review_text.copy()
clean_df['new_text'] = clean_df.text.apply(lambda x: clean_text(x))

# %%
clean_df.head()

# %%
# generate polarity scores

def sentiment(dataframe, review):

    sia = SentimentIntensityAnalyzer()

    res = {}

    for i, row in tqdm(dataframe.iterrows(), total = len(dataframe)):
        text = row[review]
        myid = row['review_id']
        res[myid] = sia.polarity_scores(text)

    vaders = pd.DataFrame(res).T
    vaders = vaders.reset_index().rename(columns={'index':'review_id'})
    vaders = vaders.merge(dataframe, how='left')

    return vaders

# %%
clean_sent = sentiment(clean_df, 'new_text')

# %%
```

```
clean_sent.head()

# %% [markdown]
# #### Average Polarity Scores Group by Business Id

# %%
polarity_scores = clean_sent.groupby('business_id')[['pos', 'neg', 'neu']].mean()
polarity_scores

# %% [markdown]
# #### Merge with business dataset

# %%
final_business = ReadJSONitems('final_businesses.json')[0]

# %%
business_df = pd.DataFrame(final_business)

# %%
business_df.shape

# %%
business_df.head()

# %%
combined_df = business_df.merge(polarity_scores, how='left', on='business_id')
combined_df

# %%
combined_df.isnull().sum()

# %%
combined_df.shape

# %% [markdown]
# #### Export to csv

# %%
combined_df.to_csv('final_business_sentiment.csv', index=False)
```

### C-3. Modeling

```
## Modeling (Attributes and Sentiment)

import pandas as pd
import numpy as np
import re
```

```
import datetime as dt
import os
import matplotlib.pyplot as plt
import seaborn as sns


pd.set_option('display.max_columns', None)
pd.set_option('display.max_colwidth', None)


final_business = pd.read_csv('final_business_sentiment.csv')
final_business.shape
final_business.head()

# Split X and y dataset

cols_to_drop = ['business_id', 'business_name', 'city', 'state', 'postal_code',
'categories', 'stars']

X = final_business.drop(cols_to_drop, axis=1)
y = final_business['stars']

X.head()


## Split training and testing dataset
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y,
random_state=1234)
X_train.shape, X_test.shape, y_train.shape, y_test.shape



#### PCA
from sklearn.preprocessing import scale

Xn = scale(X)


from sklearn.decomposition import PCA

#Create primary components
pca = PCA()
X_pca = pca.fit_transform(Xn)


ev = pca.explained_variance_ratio_
```

```
print(ev)



from matplotlib.pyplot import figure
figure(figsize=(10, 8))
plt.plot(ev) # Create a scree plot.
plt.xlabel('k number of components')
plt.ylabel('Explained variance')
plt.grid(True)
plt.xticks(np.arange(0, X.shape[1]+1, 5.0))
plt.show


figure(figsize=(10, 8))
plt.plot(np.cumsum(ev))
plt.xlabel('k number of components')
plt.ylabel('cumulative explained variance')
plt.grid(True)
plt.show()


# Create a scree plot to determine the number of primary components to use.
pca = PCA(n_components=30)

X_pca=pca.fit_transform(Xn)

X_pca[0:5]



# Create a scree plot to determine the number of primary components to use.
pca = PCA(n_components=30)

X_pca=pca.fit_transform(Xn)

X_pca[0:5]



X_pca_train, X_pca_test, y_pca_train, y_pca_test = train_test_split(X_pca, y,
test_size =.2, random_state=1234, stratify=y)
X_pca_train.shape, X_pca_test.shape, y_pca_train.shape, y_pca_test.shape


##########################################################################
##########################################################################
```

```
#### Modeling
from sklearn.metrics import mean_squared_error,mean_absolute_error,
make_scorer,classification_report,confusion_matrix,accuracy_score,roc_auc_score,roc_cu
rve
from sklearn.model_selection import train_test_split,cross_val_score,KFold
from sklearn.metrics import f1_score
from sklearn import metrics
from sklearn.preprocessing import scale
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.svm import SVR
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.neural_network import MLPRegressor

import time

# #### Random Forest (Original)
t_start = time.time()
rfrm = RandomForestRegressor()
rfrm.fit(X_train, y_train)
y_pred = rfrm.predict(X_test)
t_end = time.time()
execution_time = t_end - t_start

print (f'Execution time is {execution_time} seconds')



rfrm_mse = metrics.mean_squared_error(y_test,y_pred)
rfrm_rmse = np.sqrt(rfrm_mse)
rfrm_rsquared = rfrm.score(X_test, y_test)
print('\n', '**Evaluation of Errors**')
print (' mse: ', rfrm_mse,'\n','rmse:', rfrm_rmse)
print (' R^2: ', rfrm_rsquared)


#### Random Forest (Feature Selection)

fn = X.columns
print(f'Originally, we have {len(fn)} features.')


importances = rfrm.feature_importances_


# Draw a bar chart to see the sorted importance values with feature names.
```

```
df_importances = pd.DataFrame(data=importances, index=fn,
                              columns=['importance_value'])

df_importances.sort_values(by = 'importance_value', ascending=False,
                           inplace=True)

df_importances


figure(figsize=(8, 15))
plt.barh(df_importances.index,df_importances.importance_value)


from sklearn.feature_selection import SelectFromModel

selector = SelectFromModel(estimator=RandomForestRegressor(),threshold=0.010692)
X_reduced_rfrm = selector.fit_transform(X,y)
selector.threshold_

selected_TF = selector.get_support()
print(f'\n** {selected_TF.sum()} features are selected.')
#This shows how many features are selected.


# Show those selected features.
selected_features = []

for i,j in zip(selected_TF, fn):
    if i: selected_features.append(j)
print(f'Selected Features: {selected_features}')


# Now, we are ready to build a model using those reduced number of features.
X_reduced_train, X_reduced_test, y_reduced_train, y_reduced_test =
train_test_split(X_reduced_rfrm, y, test_size =.2, random_state=1234, stratify=y)
X_reduced_train.shape, X_reduced_test.shape, y_reduced_train.shape,
y_reduced_test.shape


# Build a model with the reduced number of features.
t_start = time.time()
rfrm2 = RandomForestRegressor()
rfrm2.fit(X_reduced_train, y_reduced_train)
y_pred2 = rfrm2.predict(X_reduced_test)
t_end = time.time()
execution_time = t_end - t_start
```

```
print (f'Execution time is {execution_time} seconds')


rfrm_reduced_mse = metrics.mean_squared_error(y_reduced_test,y_pred2)
rfrm_reduced_rmse = np.sqrt(rfrm_reduced_mse)
rfrm_reduced_rsquared = rfrm2.score(X_reduced_test, y_reduced_test)
print('\n', '**Evaluation of Errors**')
print (' mse: ', rfrm_reduced_mse,'\n','rmse:', rfrm_reduced_rmse)
print (' R^2: ', rfrm_reduced_rsquared)

#### Random Forest (PCA)
t_start = time.time()
rfrmPCA = RandomForestRegressor()
rfrmPCA.fit(X_pca_train, y_pca_train)
y_predpca = rfrmPCA.predict(X_pca_test)
t_end = time.time()
execution_time = t_end - t_start

print (f'Execution time is {execution_time} seconds')



rfrm_pca_mse = metrics.mean_squared_error(y_pca_test,y_predpca)
rfrm_pca_rmse = np.sqrt(rfrm_pca_mse)
rfrm_pca_rsquared = rfrmPCA.score(X_pca_test, y_pca_test)
print('\n', '**Evaluation of Errors**')
print (' mse: ', rfrm_pca_mse,'\n','rmse:', rfrm_pca_rmse)
print (' R^2: ', rfrm_pca_rsquared)


##########################################################################


#### Decision Tree (Original)
t_start = time.time()
dtm = DecisionTreeRegressor()
dtm.fit(X_train,y_train)
y_predDT = dtm.predict(X_test)
t_end = time.time()
execution_time = t_end - t_start

print (f'Execution time is {execution_time} seconds')


dtm_mse = metrics.mean_squared_error(y_test,y_predDT)
dtm_rmse = np.sqrt(dtm_mse)
dtm_rsquared = dtm.score(X_test, y_test)
print('\n', '**Evaluation of Errors**')
```

```
print (' mse: ', dtm_mse,'\n','rmse:', dtm_rmse)
print (' R^2: ', dtm_rsquared)


#### Decision Tree (Feature Selection)

fn = X.columns
print(f'Originally, we have {len(fn)} features.')

importances = dtm.feature_importances_

# Draw a bar chart to see the sorted importance values with feature names.

df_importances = pd.DataFrame(data=importances, index=fn,
                              columns=['importance_value'])

df_importances.sort_values(by = 'importance_value', ascending=False,
                           inplace=True)

df_importances


figure(figsize=(8, 15))
plt.barh(df_importances.index,df_importances.importance_value)


selector = SelectFromModel(estimator=DecisionTreeRegressor(),threshold=0.01)
X_reduced_dtm = selector.fit_transform(X,y)
selector.threshold_
selected_TF = selector.get_support()
print(f'\n** {selected_TF.sum()} features are selected.')
#This shows how many features are selected.



# Show those selected features.
selected_features = []
for i,j in zip(selected_TF, fn):
    if i: selected_features.append(j)
print(f'Selected Features: {selected_features}')



# Now, we are ready to build a model using those reduced number of features.
X_reduced_train, X_reduced_test, y_reduced_train, y_reduced_test =
train_test_split(X_reduced_dtm, y, test_size =.2, random_state=1234, stratify=y)
X_reduced_train.shape, X_reduced_test.shape, y_reduced_train.shape,
y_reduced_test.shape
```

```
# Build a model with the reduced number of features.
t_start = time.time()
dtm2 = RandomForestRegressor()
dtm2.fit(X_reduced_train, y_reduced_train)
y_pred_dtm2 = dtm2.predict(X_reduced_test)
t_end = time.time()
execution_time = t_end - t_start

print (f'Execution time is {execution_time} seconds')


dtm_reduced_mse = metrics.mean_squared_error(y_reduced_test,y_pred_dtm2)
dtm_reduced_rmse = np.sqrt(dtm_reduced_mse)
dtm_reduced_rsquared = dtm2.score(X_reduced_test, y_reduced_test)
print('\n', '**Evaluation of Errors**')
print (' mse: ', dtm_reduced_mse,'\n','rmse:', dtm_reduced_rmse)
print (' R^2: ', dtm_reduced_rsquared)


#### Decision Tree (PCA)
t_start = time.time()
dtmPCA = DecisionTreeRegressor()
dtmPCA.fit(X_pca_train,y_pca_train)
y_predDTPCA = dtmPCA.predict(X_pca_test)
t_end = time.time()
execution_time = t_end - t_start

print (f'Execution time is {execution_time} seconds')


dtm_pca_mse = metrics.mean_squared_error(y_pca_test,y_predDTPCA)
dtm_pca_rmse = np.sqrt(dtm_pca_mse)
dtm_pca_rsquared = dtmPCA.score(X_pca_test, y_pca_test)
print('\n', '**Evaluation of Errors**')
print (' mse: ', dtm_pca_mse,'\n','rmse:', dtm_pca_rmse)
print (' R^2: ', dtm_pca_rsquared)


##########################################################################


#### Gradient Boosting (Original)
t_start = time.time()
gbr = GradientBoostingRegressor()
gbr.fit(X_train, y_train)
```

```
y_predGB = gbr.predict(X_test)
t_end = time.time()
execution_time = t_end - t_start
print (f'Execution time is {execution_time} seconds')


gbr_mse = metrics.mean_squared_error(y_test,y_predGB)
gbr_rmse = np.sqrt(gbr_mse)
gbr_rsquared = gbr.score(X_test, y_test)
print('\n', '**Evaluation of Errors**')
print (' mse: ', gbr_mse,'\n','rmse:', gbr_rmse)
print (' R^2: ', gbr_rsquared)


#### Gradient Boosting (Feature Selection)
importances = gbr.feature_importances_

# Draw a bar chart to see the sorted importance values with feature names.
pd.set_option('display.float_format', lambda x: '%.6f' % x)

df_importances = pd.DataFrame(data=importances, index=fn,
                              columns=['importance_value'])

df_importances.sort_values(by = 'importance_value', ascending=False,
                           inplace=True)

df_importances


figure(figsize=(8, 15))
plt.barh(df_importances.index,df_importances.importance_value)



selector = SelectFromModel(estimator=GradientBoostingRegressor(),threshold=0.004)
X_reduced_gbr = selector.fit_transform(X,y)
selector.threshold_
selected_TF = selector.get_support()
print(f'\n** {selected_TF.sum()} features are selected.')
#This shows how many features are selected.


# Show those selected features.
selected_features = []
for i,j in zip(selected_TF, fn):
    if i: selected_features.append(j)
print(f'Selected Features: {selected_features}')
```

```
# Now, we are ready to build a model using those reduced number of features.

X_reduced_train, X_reduced_test, y_reduced_train, y_reduced_test =
train_test_split(X_reduced_gbr, y, test_size =.2, random_state=1234, stratify=y)
X_reduced_train.shape, X_reduced_test.shape, y_reduced_train.shape,
y_reduced_test.shape



# Build a model with the reduced number of features.
t_start = time.time()
gbr2 = GradientBoostingRegressor()
gbr2.fit(X_reduced_train, y_reduced_train)
y_pred_gbr2 = gbr2.predict(X_reduced_test)
t_end = time.time()
execution_time = t_end - t_start

print (f'Execution time is {execution_time} seconds')


gbr_reduced_mse = metrics.mean_squared_error(y_reduced_test,y_pred_gbr2)
gbr_reduced_rmse = np.sqrt(gbr_reduced_mse)
gbr_reduced_rsquared = gbr2.score(X_reduced_test, y_reduced_test)
print('\n', '**Evaluation of Errors**')
print (' mse: ', gbr_reduced_mse,'\n','rmse:', gbr_reduced_rmse)
print (' R^2: ', gbr_reduced_rsquared)


#### Gradient Boosting (PCA)
t_start = time.time()
gbrPCA = GradientBoostingRegressor()
gbrPCA.fit(X_pca_train,y_pca_train)
y_predGBRPCA = gbrPCA.predict(X_pca_test)
t_end = time.time()
execution_time = t_end - t_start

print (f'Execution time is {execution_time} seconds')


gbr_pca_mse = metrics.mean_squared_error(y_pca_test,y_predGBRPCA)
gbr_pca_rmse = np.sqrt(gbr_pca_mse)
gbr_pca_rsquared = gbrPCA.score(X_pca_test, y_pca_test)
print('\n', '**Evaluation of Errors**')
print (' mse: ', gbr_pca_mse,'\n','rmse:', gbr_pca_rmse)
print (' R^2: ', gbr_pca_rsquared)


##########################################################################
```

```
#### SVR (Original)
scaler = StandardScaler()
Xn = scaler.fit_transform(X)


## Split training and testing dataset
X_train, X_test, y_train, y_test = train_test_split(Xn, y, test_size=0.2, stratify=y,
random_state=1234)
X_train.shape, X_test.shape, y_train.shape, y_test.shape


t_start = time.time()
svr = SVR(kernel='rbf')
svr.fit(X_train, y_train)
y_predSVR = svr.predict(X_test)
t_end = time.time()
execution_time = t_end - t_start

print (f'Execution time is {execution_time} seconds')


svr_mse = metrics.mean_squared_error(y_test,y_predSVR)
svr_rmse = np.sqrt(svr_mse)
svr_rsquared = svr.score(X_test, y_test)
print('\n', '**Evaluation of Errors**')
print (' mse: ', svr_mse,'\n','rmse:', svr_rmse)
print (' R^2: ', svr_rsquared)



#### SVR (Feature Selection)
importances = svr.feature_importances_

# Draw a bar chart to see the sorted importance values with feature names.
# pd.set_option('display.float_format', lambda x: '%.6f' % x)
df_importances = pd.DataFrame(data=importances, index=fn,
                              columns=['importance_value'])

df_importances.sort_values(by = 'importance_value', ascending=False,
                           inplace=True)

df_importances



#### SVR (PCA)
```

```
t_start = time.time()
svrPCA = SVR(kernel='rbf')
svrPCA.fit(X_pca_train,y_pca_train)
y_predSVRPCA = svrPCA.predict(X_pca_test)
t_end = time.time()
execution_time = t_end - t_start

print (f'Execution time is {execution_time} seconds')


svr_pca_mse = metrics.mean_squared_error(y_pca_test,y_predSVRPCA)
svr_pca_rmse = np.sqrt(svr_pca_mse)
svr_pca_rsquared = svrPCA.score(X_pca_test, y_pca_test)
print('\n', '**Evaluation of Errors**')
print (' mse: ', svr_pca_mse,'\n','rmse:', svr_pca_rmse)
print (' R^2: ', svr_pca_rsquared)



# #### Neural Network (Original)
scaler = StandardScaler()
Xn = scaler.fit_transform(X)

## Split training and testing dataset
X_train, X_test, y_train, y_test = train_test_split(Xn, y, test_size=0.2, stratify=y,
random_state=1234)
X_train.shape, X_test.shape, y_train.shape, y_test.shape



t_start = time.time()
nnr = MLPRegressor(hidden_layer_sizes=(100), max_iter=1000, random_state=1234)
nnr.fit(X_train, y_train)
y_predNNR = nnr.predict(X_test)
t_end = time.time()
execution_time = t_end - t_start

print (f'Execution time is {execution_time} seconds')


nnr_mse = metrics.mean_squared_error(y_test,y_predNNR)
nnr_rmse = np.sqrt(nnr_mse)
nnr_rsquared = nnr.score(X_test, y_test)
print('\n', '**Evaluation of Errors**')
print (' mse: ', nnr_mse,'\n','rmse:', nnr_rmse)
print (' R^2: ', nnr_rsquared)

#########################################################################
```

```
#### Neural Network (Feature Selection)
importances = nnr.feature_importances_

# Draw a bar chart to see the sorted importance values with feature names.
pd.set_option('display.float_format', lambda x: '%.6f' % x)

df_importances = pd.DataFrame(data=importances, index=fn,
                              columns=['importance_value'])

df_importances.sort_values(by = 'importance_value', ascending=False,
                           inplace=True)

df_importances



#### Neural Network (PCA)
t_start = time.time()
nnrPCA = MLPRegressor(hidden_layer_sizes=(100), max_iter=1000, random_state=1234)
nnrPCA.fit(X_pca_train,y_pca_train)
y_predNNRPCA = nnrPCA.predict(X_pca_test)
t_end = time.time()
execution_time = t_end - t_start

print (f'Execution time is {execution_time} seconds')


nnr_pca_mse = metrics.mean_squared_error(y_pca_test,y_predNNRPCA)
nnr_pca_rmse = np.sqrt(nnr_pca_mse)
nnr_pca_rsquared = nnrPCA.score(X_pca_test, y_pca_test)
print('\n', '**Evaluation of Errors**')
print (' mse: ', nnr_pca_mse,'\n','rmse:', nnr_pca_rmse)
print (' R^2: ', nnr_pca_rsquared)


##########################################################################


#### Summary
rsquared_dict = {
    "Decision Tree Original": dtm_rsquared,
    "Decision Tree Feature Selection": dtm_reduced_rsquared,
    "Decision Tree PCA": dtm_pca_rsquared,
    "Random Forest Original": rfrm_rsquared,
    "Random Forest Feature Selection": rfrm_reduced_rsquared,
    "Random Forest PCA": rfrm_pca_rsquared,
```

```
    "Gradient Boosting Original": gbr_rsquared,
    "Gradient Boosting Feature Selection": gbr_reduced_rsquared,
    "Gradient Boosting  PCA": gbr_pca_rsquared,
    "SVR Original": svr_rsquared,
    "SVR PCA": svr_pca_rsquared,
    "Neural Network Original": nnr_rsquared,
    "Neutral Network PCA": nnr_pca_rsquared}



rsquared = pd.DataFrame(rsquared_dict.items(), columns=['Models', 'RSquared'])
rsquared.sort_values(by="RSquared", ascending=False, inplace=True)
rsquared.reset_index(drop=True)


#### K-Fold Cross Validation
from sklearn.model_selection import cross_val_score

dtm_mean_score = np.mean(cross_val_score(dtm,X,y,cv=5))
rfrm_mean_score = np.mean(cross_val_score(rfrm,X,y,cv=5))
gbr_mean_score = np.mean(cross_val_score(gbr,X,y,cv=5))
svr_mean_score = np.mean(cross_val_score(svr,Xn,y,cv=5))
nnr_mean_score = np.mean(cross_val_score(nnr,Xn,y,cv=5))


# Print the scores (R²)
print('\n  ** Mean Scores (R²) **')
print(f'Mean Score for Decision Tree: {dtm_mean_score:.4f}')
print(f'Mean Score for Random Forest: {rfrm_mean_score:.4f}')
print(f'Mean Score for Gradient Boosting: {gbr_mean_score:.4f}')
print(f'Mean Score for Support Vector Regression: {svr_mean_score:.4f}')
print(f'Mean Score for Neural Network: {nnr_mean_score:.4f}')


dtm_reduced_mean_score = np.mean(cross_val_score(dtm2,X_reduced_dtm,y,cv=5))
rfrm_reduced_mean_score = np.mean(cross_val_score(rfrm2,X_reduced_rfrm,y,cv=5))
gbr_reduced_mean_score = np.mean(cross_val_score(gbr2,X_reduced_gbr,y,cv=5))


# Print the scores (R²)
print('\n  ** Mean Scores (R²) **')
print(f'Mean Score for Decision Tree: {dtm_reduced_mean_score:.4f}')
print(f'Mean Score for Random Forest: {rfrm_reduced_mean_score:.4f}')
print(f'Mean Score for Gradient Boosting: {gbr_reduced_mean_score:.4f}')

dtm_PCA_mean_score = np.mean(cross_val_score(dtmPCA,X_pca,y,cv=5))
rfrm_PCA_mean_score = np.mean(cross_val_score(rfrmPCA,X_pca,y,cv=5))
gbr_PCA_mean_score = np.mean(cross_val_score(gbrPCA,X_pca,y,cv=5))
```

```
svr_PCA_mean_score = np.mean(cross_val_score(svrPCA,X_pca,y,cv=5))
nnr_PCA_mean_score = np.mean(cross_val_score(nnrPCA,X_pca,y,cv=5))


# Print the scores (R²)
print('\n  ** Mean Scores (R²) **')
print(f'Mean Score for Decision Tree: {dtm_PCA_mean_score:.4f}')
print(f'Mean Score for Random Forest: {rfrm_PCA_mean_score:.4f}')
print(f'Mean Score for Gradient Boosting: {gbr_PCA_mean_score:.4f}')
print(f'Mean Score for Support Vector Regression: {svr_PCA_mean_score:.4f}')
print(f'Mean Score for Neural Network: {nnr_PCA_mean_score:.4f}')


gbr_mean_score_10 = np.mean(cross_val_score(gbr,X,y,cv=10))
print(f'Mean Score for Gradient Boosting: {gbr_mean_score_10:.4f}')


gbr_mean_score_15 = np.mean(cross_val_score(gbr,X,y,cv=15))
print(f'Mean Score for Gradient Boosting: {gbr_mean_score_15:.4f}')
```

### C-4. Hyperparameter Tuning

```
# ## Hyperparameter Tuning

import pandas as pd
import numpy as np
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.neural_network import MLPRegressor
from datetime import time
import os
from sklearn import metrics


pd.set_option('display.max_columns', None)
pd.set_option('display.max_colwidth', None)


final_business = pd.read_csv('final_business_sentiment.csv')


# Split X and y dataset
cols_to_drop = ['business_id', 'business_name', 'city', 'state', 'postal_code',
'categories', 'stars']

X = final_business.drop(cols_to_drop, axis=1)
y = final_business['stars']
```

```
## Split training and testing dataset
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y,
random_state=1234)
X_train.shape, X_test.shape, y_train.shape, y_test.shape



#### GridSearchCV - Gradient Boosting
from sklearn.model_selection import GridSearchCV
import numpy as np

gbr = GradientBoostingRegressor()

params = {
        'learning_rate': list(np.arange(0.1, 0.6, 0.1)),
        'n_estimators' : list(np.arange(100, 600, 100)),
        'max_depth'    : [3, 5, 10, 15],
        'subsample'    : [1.0, 2.0, 3.0]
        }


import time

start = time.time()
src = GridSearchCV(estimator= gbr, param_grid= params)
src.fit(X, y)
end = time.time()


print('\n\n   **Report**')
print(f'The best estimator: {src.best_estimator_}')
print(f'The best parameters:\n {src.best_params_}')
print(f'The best score: {src.best_score_:.4f}')
print(f'Total run time for GridSearchCV: {(end - start):.2f} seconds')



results_gs = pd.DataFrame(src.cv_results_)
results_gs



#### RandomizedSearchCV - Gradient Boosting
from sklearn.model_selection import RandomizedSearchCV
import numpy as np
```

```
gbr = GradientBoostingRegressor()

params = {
        'learning_rate': list(np.arange(0.1, 0.6, 0.1)),
        'n_estimators' : list(np.arange(100, 600, 100)),
        'max_depth'    : [3, 5, 10, 15],
        'subsample'    : [1.0, 2.0, 3.0]
                }

import time

start_r = time.time()
rand_src = RandomizedSearchCV(estimator= gbr, param_distributions = params, n_iter=6)
rand_src.fit(X,y)
end_r = time.time()


print('\n\n   **Report**')
print(f'The best estimator: {rand_src.best_estimator_}')
print(f'The best parameters:\n {rand_src.best_params_}')
print(f'The best score: {rand_src.best_score_:.4f}')
print(f'Total run time for RandomizedSearchCV: {(end_r - start_r):.2f} seconds')


# Check the details of search
results_rgs = pd.DataFrame(rand_src.cv_results_)
results_rgs


#### Gradient Boosting Original
import time

t_start = time.time()
gbr = GradientBoostingRegressor(learning_rate=0.1, max_depth=3, n_estimators=100,
subsample=1.0)
gbr.fit(X_train, y_train)
y_predGB = gbr.predict(X_test)
t_end = time.time()
execution_time = t_end - t_start

print (f'Execution time is {execution_time} seconds')


gbr_mse = metrics.mean_squared_error(y_test,y_predGB)
gbr_rmse = np.sqrt(gbr_mse)
gbr_rsquared = gbr.score(X_test, y_test)
print('\n', '**Evaluation of Errors**')
print (' mse: ', gbr_mse,'\n','rmse:', gbr_rmse)
```

```
print (' R^2: ', gbr_rsquared)
```

### C-5. Clustering

```
## Clustering

import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

pd.set_option('display.max_columns', None)
pd.set_option('display.max_colwidth', None)

# os.chdir('C:/Users/zzlen/OneDrive - Seattle University/MSBA/5. Fall 2022/BUAN
5510/Capstone Project/Yelp Review/JSON')


# load the dataset
final_business = pd.read_csv('final_business_sentiment.csv')


# Split X and y dataset
cols_to_drop = ['business_id', 'business_name', 'city', 'state', 'postal_code',
'categories']

# X = final_business.drop(cols_to_drop, axis=1)
# y = final_business['stars']

df = final_business.drop(cols_to_drop, axis=1)


# Normalize the data
scaler = StandardScaler()
dfn = scaler.fit_transform(df)


## Finding K:

# Initialize the list for inertia values - sum of squared distances
inertia_list = []

# Calculate the inertia for the number of clusters.
for i in range(2,6):
    km = KMeans(n_clusters=i, random_state=1234)
    km.fit(dfn)
```

```
    inertia_list.append(km.inertia_)


# Check the inertia values.
for i in range(len(inertia_list)):
    print('{0}: {1:.2f}'.format(i+2, inertia_list[i]))


# Draw the plot to find the elbow
from matplotlib.pyplot import figure

figure(figsize=(10, 8))
plt.plot(range(2,6), inertia_list)
plt.grid(True)
plt.xlabel('Number of Clusters')
plt.ylabel('Inertia')
plt.show()


# Selected k = 3
km = KMeans(n_clusters=3, random_state=1234)


# Build a model.
km.fit(dfn)


# Show the cluster numbers and assign them to a variable.
labels = km.labels_
labels


km.inertia_


km.n_clusters


# Add the cluster numbers to the original data.
X_clusters = np.column_stack((labels,df))



df_1 = pd.DataFrame(data=df,columns=df.columns)

# Add the cluster numbers to df
df_1['Cluster_No'] = km.labels_
df_1.head()
```

```
df_1.info()


# Create a dataframe for each cluster.
Cluster_0 = df_1.loc[df_1.Cluster_No==0]
Cluster_1 = df_1.loc[df_1.Cluster_No==1]
Cluster_2 = df_1.loc[df_1.Cluster_No==2]


# Produce some descriptive information.
df_1['Cluster_No'].value_counts()
df_1.describe().T

## Modeling ##

from sklearn.metrics import mean_squared_error,mean_absolute_error,
make_scorer,classification_report,confusion_matrix,accuracy_score,roc_auc_score,roc_cu
rve
from sklearn.model_selection import train_test_split,cross_val_score,KFold
from sklearn.metrics import f1_score
from sklearn import metrics
from sklearn.preprocessing import scale
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.svm import SVR
from sklearn.ensemble import GradientBoostingRegressor

import time



## Cluster 0:
#############

# Split X and y dataset
X0 = Cluster_0.drop('stars', axis=1)
y0 = Cluster_0['stars']

y0.value_counts()


## Split training and testing dataset
from sklearn.model_selection import train_test_split

X0_train, X0_test, y0_train, y0_test = train_test_split(X0, y0, test_size=0.2,
stratify=y0, random_state=1234)
X0_train.shape, X0_test.shape, y0_train.shape, y0_test.shape
```

```
Cluster_0_dropped = Cluster_0[Cluster_0['stars'] != 5.0]


# Split X and y dataset
X0_dropped = Cluster_0_dropped.drop('stars', axis=1)
y0_dropped = Cluster_0_dropped['stars']
y0_dropped.value_counts()



## Split training and testing dataset
from sklearn.model_selection import train_test_split

X0_train, X0_test, y0_train, y0_test = train_test_split(X0_dropped, y0_dropped,
test_size=0.2, stratify=y0_dropped, random_state=1234)

X0_train.shape, X0_test.shape, y0_train.shape, y0_test.shape


#### Cluster 0_Gradient Boosting (Original)
t_start = time.time()
gbr0 = GradientBoostingRegressor(learning_rate=0.1, max_depth=3, n_estimators=100,
subsample=1.0)
gbr0.fit(X0_train, y0_train)
y0_predGB = gbr0.predict(X0_test)
t_end = time.time()
execution_time = t_end - t_start

print (f'Execution time is {execution_time} seconds')


gbr0_mse = metrics.mean_squared_error(y0_test,y0_predGB)
gbr0_rmse = np.sqrt(gbr0_mse)
gbr0_rsquared = gbr0.score(X0_test, y0_test)
print('\n', '**Evaluation of Errors**')
print (' mse: ', gbr0_mse,'\n','rmse:', gbr0_rmse)
print (' R^2: ', gbr0_rsquared)



## Cluster 1
#############

# Split X and y dataset
X1 = Cluster_1.drop('stars', axis=1)
y1 = Cluster_1['stars']
```

```
y1.value_counts()


## Split training and testing dataset
X1_train, X1_test, y1_train, y1_test = train_test_split(X1, y1, test_size=0.2,
stratify=y1, random_state=1234)
X1_train.shape, X1_test.shape, y1_train.shape, y1_test.shape


#### Cluster 1_Gradient Boosting (Original)
t_start = time.time()
gbr1 = GradientBoostingRegressor(learning_rate=0.1, max_depth=3, n_estimators=100,
subsample=1.0)
gbr1.fit(X1_train, y1_train)
y1_predGB = gbr1.predict(X1_test)
t_end = time.time()
execution_time = t_end - t_start

print (f'Execution time is {execution_time} seconds')


gbr1_mse = metrics.mean_squared_error(y1_test,y1_predGB)
gbr1_rmse = np.sqrt(gbr1_mse)
gbr1_rsquared = gbr1.score(X1_test, y1_test)
print('\n', '**Evaluation of Errors**')
print (' mse: ', gbr1_mse,'\n','rmse:', gbr1_rmse)
print (' R^2: ', gbr1_rsquared)



## Cluster 2:
#############

# Split X and y dataset
X2 = Cluster_2.drop('stars', axis=1)
y2 = Cluster_2['stars']
y2.value_counts()


## Split training and testing dataset
X2_train, X2_test, y2_train, y2_test = train_test_split(X2, y2, test_size=0.2,
stratify=y2, random_state=1234)
X2_train.shape, X2_test.shape, y2_train.shape, y2_test.shape

Cluster_2_dropped = Cluster_2[Cluster_2['stars'] != 2.0]


# Split X and y dataset
```

```
X2_dropped = Cluster_2_dropped.drop('stars', axis=1)
y2_dropped = Cluster_2_dropped['stars']
y2_dropped.value_counts()


## Split training and testing dataset
X2_train, X2_test, y2_train, y2_test = train_test_split(X2_dropped, y2_dropped,
test_size=0.2, stratify=y2_dropped, random_state=1234)
X2_train.shape, X2_test.shape, y2_train.shape, y2_test.shape


#### Cluster 2_Gradient Boosting (Original)

t_start = time.time()
gbr2 = GradientBoostingRegressor(learning_rate=0.1, max_depth=3, n_estimators=100,
subsample=1.0)
gbr2.fit(X2_train, y2_train)
y2_predGB = gbr2.predict(X2_test)
t_end = time.time()
execution_time = t_end - t_start

print (f'Execution time is {execution_time} seconds')


gbr2_mse = metrics.mean_squared_error(y2_test,y2_predGB)
gbr2_rmse = np.sqrt(gbr2_mse)
gbr2_rsquared = gbr2.score(X2_test, y2_test)
print('\n', '**Evaluation of Errors**')
print (' mse: ', gbr2_mse,'\n','rmse:', gbr2_rmse)
print (' R^2: ', gbr2_rsquared)
```