IEOR 142

Team 12

Sentiment Analysis on Yelp Reviews

Kenny Johng, Kevin Pham, Daniel Hwang, Amanda Wu

I. Background

Yelp is a crowd-sourced review platform whose primary purpose is to provide recommendations and information regarding local businesses such as restaurants, auto shops, hair salons, etc. Regarding the overall experience of reviewing a business, we can think of it in two parts:

A. User Level

Users can submit a rating and a review text for businesses they have experienced. These individual reviews can also be upvoted by other reviewers in terms of three categories: Useful, Funny, and Cool. For example, if a user is reading another user's review and finds it practical or useful, they can give it a "Useful" upvote. The review will then display how many votes it has for each of these three categories.

B. Business Level

Each business has an average rating that is calculated from all the individual reviews submitted for that business. Another important attribute on the business level are its "features". These features give binary (Yes/No) information pertaining to the services offered and business characteristics such as Alcohol Served, Delivery, Good for Kids, Good for Groups, etc.

II. The Problem/Goal

Given the plethora of a data available on Yelp's platform (overall/individual reviews and business features), we think it could be very useful and feasible to find patterns in a business' features and reviews and how they relate to a reviewer's individual experience. Thus, our primary objective is to provide business owners with insight as to what parts of their business they should focus on to improve their business model and consequently their Yelp ratings.

III. Data Collection

We imported and joined multiple datasets from Yelp's open source API: user review data, user data, and business data. One record/row of the resulting, joined data set represents one instance of a user's review. The final dataset includes the following columns:

Column Name	Data Type	Description
attributes	String	Various features/services of a business
business_id	String	Unique identifier for a business branch
categories	String	Type of business
city	String	City of the business branch
hours	String	Operating hours throughout the week
is_open	Binary	If a business branch is still operating
name	String	Name of business
postal_code	Integer	Postal code of business branch
review_count	Integer	# of reviews left for this branch
stars_x	Integer	Overall rating of the branch
cool	Integer	# of "Cool" votes
date	String	Date the review was posted
funny	Integer	# of "Funny" votes
review_id	String	Unique identifier for the review
stars_y	Integer	User's rating for the branch
text	String	Actual text of the user's review
useful	Integer	# of "Useful" votes
user_id	String	Unique identifier for the user

IV. Data Preprocessing

After collecting all of the data, there were a few steps that we took in order to make the data better fit our needs.

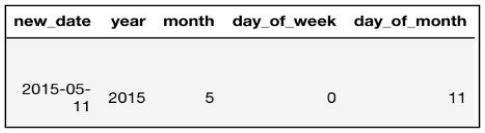
1. First we dropped all data points where the business was not located in Phoenix,AZ.
2. Then we dropped the 'city', 'postal_code', 'neighborhood', and 'state' columns from out data set, because we deemed them redundant given that we are only looking at businesses in Phoenix, AZ.
3. We then made all of our text data lowercase.
4. Each restaurant on Yelp has attributes, and our dataset had all the attributes as a list. In order to make this information incorporable into future models, we created separate columns for each attribute present in our dataset. Then we assigned binary values to each attribute for each restaurant: 1 if that attribute is true for that restaurant, and 0 if false.
5. We also noticed that the number of business names in our dataset and the number of business_id's did not match. We determined that this is because if a restaurant has multiple branches, each branch has a unique business_id but keeps the same business name.

6. Finally we dropped rows where there were less then 15 reviews, to remove outliers. For example a business with one review of 5 stars would have an overall rating of 5 star, and would give a skewed view of what features determine a high rating.

V. Feature Engineering

From the data given we determined that there are four features that we determined could be helpful in predicting the Yelp rating of a business. 1. Datetime Objects 2. Length of Reviews 3. Sentiment Scores 4. Most Commonly Used Words

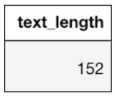
1. Datetime Objects: We changed 'date' to a datetime object in order to separate into year, month, and day columns for the data the review was added.



new_date	year	month	day_of_week	day_of_month
2015-05-11	2015	5	0	11

Figure 5.1: Datetime Objects

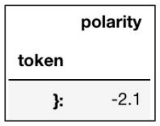
2. Length of Reviews: Calculated the length of reviews with respect to the number of characters.



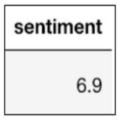
text_length
152

Figure 5.2: Number of characters in a review

3. Sentiment Scores: Used VADER to calculate the sentiments of reviews by assigning unique values to different words and emoticons.



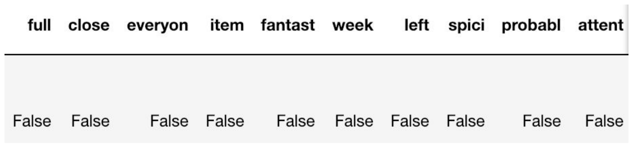
token	polarity
}:	-2.1



sentiment
6.9

Figure 5.3: Sentiment scores for different tokens **Figure 5.4: Calculated sentiment score of a review**

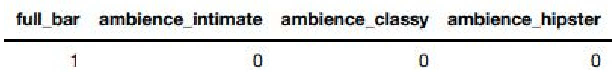
4. Most Commonly Used Words: Selected top 300 words appeared in all reviews, checked whether each word appears in a review. Before selecting the top 300 words, we first stemmed the words and removed all stopwords. As a result, we had 300 additional, binary features corresponding to a word in the top 300.



full	close	everyon	item	fantast	week	left	spici	probabl	attent
False	False	False	False	False	False	False	False	False	False

Figure 5.5: One hot encoded most commonly used words attributes

5. One Hot Encoding of Attributes: Took a single string of different attributes of a business (full bar, outdoor seating, good for groups, etc) and one hot encoded it to make a series of separate, categorical variables that are of the binary data type. This provides us with 40 more features and also allows us to capture the relationship between each attribute and the given rating. We believe the existence of these features are useful to business owners because they can understand how different features and services of their business relate to their customers' experience.



full_bar	ambience_intimate	ambience_classy	ambience_hipster
1	0	0	0

Figure 5.6: One hot encoded attributes

VI. Exploratory Data Analysis

Next we will look at some exploratory data analysis using our current dataset.

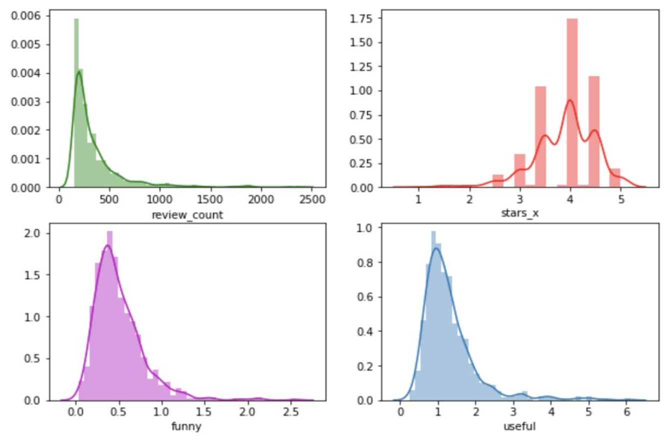


Figure 6.1 Sample Distributions of Different Variables

Key Observations:

- The average review_count for the different business is ~200
- The distribution of average funny votes and average useful votes are almost identical
- The overall rating for a business in the dataset is 4 stars

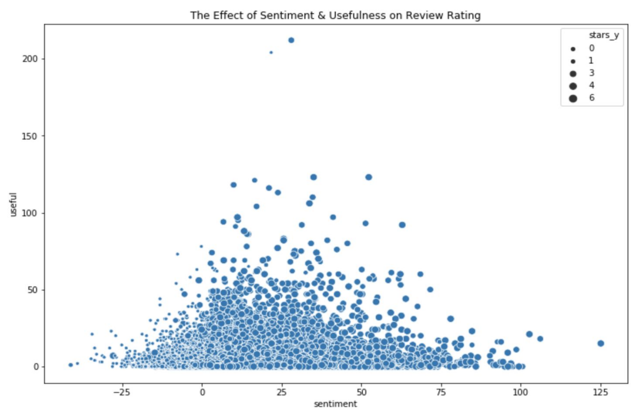


Figure 6.2: Sentiment score mapped against individual ratings & usefulness ratings

Key Observations:

- Individual ratings (stars_y) is highly correlated with the sentiment scores as well as the usefulness scores of the review rating rated by other users
- Usefulness scores tend to be highest when sentiment scores are less extreme

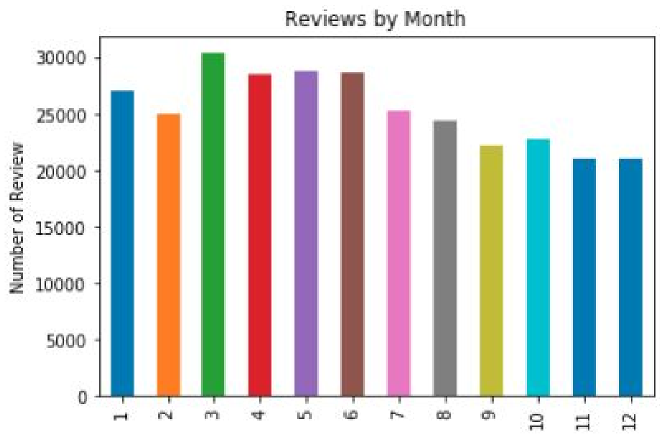


Figure 6.3: Total # of reviews across all business mapped against months of the year

Key Observations:

- More reviews are given during the spring months(March - June)
- Less reviews are given during the fall and winter (September - December)
- Likely that more people are travelling or participating in recreational activities more during the spring and summer times (vacations, end of school, etc)

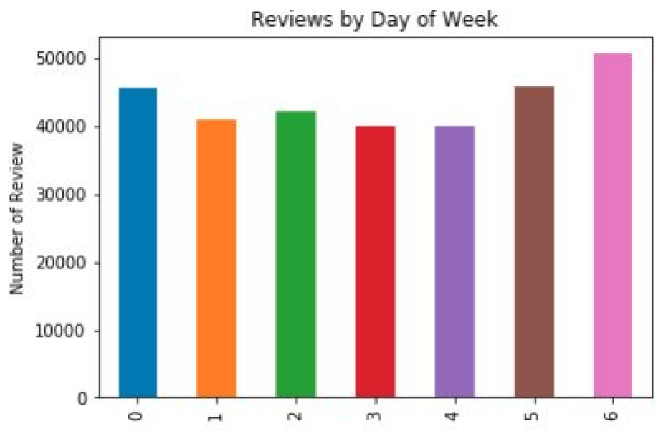


Figure 6.4: Total # of reviews across all business mapped against months of the year

Key Observations:

- Reviews are more likely given during the weekend which makes sense because that is when people are more likely to go out to eat, drink, or play
- Business owners can capitalize on the data displayed in Fig 6.3 and 6.4 because they can put more emphasis on their customer experience during these peak times of reviews

VII. Models

We split our data into a training (66%) and test (33%) set. After the feature engineering and data preprocessing, we are left with 343 features. A majority of those features are the 300 most common words but another chunk of the features are the one-hot encoded attributes of the business which are also binary data types and features we obtained from feature engineering. There were two possible approaches/response variables that we considered: predicting the overall rating for a business by looking at the reviews on an aggregate level OR predicting a single rating by looking at the reviews on an individual level. We decided on the latter for three reasons:

Assumptions:

1. Extreme sentiment scores for individual reviews would skew the aggregated sentiment score. For instance, if one review is particularly long and has a lot of negative sentiments, that would drive the overall sentiment score down and consequently the overall predicted rating as opposed to having this extreme sentiment score isolated in one row to predict a single, low rating that would be averaged with the other predicted values
2. If we were looking at an aggregate level, each word would have a smaller weight when influencing the overall sentiment score. Therefore, looking at each individual review would better allow us to determine the influence of certain words.
3. Our intuition is that business owners who would actually utilize our model would tend to be owners of smaller businesses (i.e. Walmart corporate probably does not care about their Yelp reviews to the same level a small, family owned, Berkeley restaurant would). In that sense, we think it is more useful for these owners to understand what aspects of their business appeal the most/least to an individual.

Note: Without the added 300 features (300 most common words), our model's accuracy was around 31% which, as shown below, was significantly increased with the adding of these features. This is intuitive because, as shown above, the sentiment score of the review text was highly correlated with the actual ratings so when we added features that capture the review texts' sentiments (the 300 common words), we can better capture the relationship between the user's review and his/her rating.

Models:

A. Linear Regression/OLS
   a. OSR^2= 0.51
      Although this might seem low, we are satisfied with this value. Our model is predicting the ratings on a continuous scale as opposed to predicting only the discrete rating values: 1,2,3,4,5. Continuous response variables are good in this instance because we believe business owners would find more utility from predicting a 4.7 star rating as opposed to a 5 star rating because it would give them more opportunities to "fine tune" their business features
   b. MSE = 0.83
      Significance: Since we are predicting a continuous response variable, an MSE value of **0.83** represents a fairly low error for our model. This can be interpreted as having an error value that is less than 1 in relative to the actual ratings.

B. Ridge Regression with cross validation
   We achieved the results discussed below by using cross validation on the hyper parameter lambda which resulted in the best model (both Ridge and Lasso regression)
   a. OSR^2= 0.51
      We observe the same OSR^2 as our linear regression model. This is because Ridge regression serves to treat multicollinearity - something our model doesn't really exhibit because a majority of our features (i.e. 300 common words and the encoded attributes) are of the binary data type.

C. Lasso Regression with cross validation
   a. OSR^2 = 0.31
      As in the case of Ridge regression, it helps us reduce number of features in the model down to only 4, and our model doesn't exhibit multicollinearity which would explain why the OSR^2 here is lower. Lasso regression penalizes the absolute size of regression coefficients to reduce multicollinearity but in our case we are left with inaccurate estimations of the coefficients that are decreasing our OSR^2

D. Logistic regression (Not considered)
   We decided not to use a logistic regression model. As mentioned above, we think there is value in an owner understanding what contributes to a 4.7 review as opposed to a 4.9 review. Therefore, a linear regression model works best for our purposes. However, if we were to have used a logistic regression model to predict ratings, we would have to have some sort of a rating benchmark (4 star for example) and tailor all of our predictions relative to that benchmark. For example, a business owner would only know if a review is or is not a 4 star rating. This implies that an incorrectly predicted value of 5 would be treated the same as an incorrectly predicted value of 1 which is not nearly as useful to an owner as a continuous response variable.

VIII. Conclusion

Amongst our models, we find that a crossed validated Ridge regression does a good job interpreting the relationship between the business' feature and a customer's review and ultimately their rating. If a business user were to use our model, we hope they would be able to preempt a customer's experience and iteratively improve their business based on customer feedback.

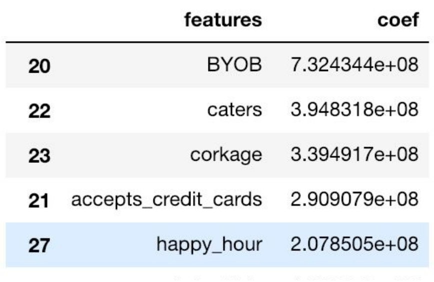
OSR^2

One important thing to emphasize is our OSR^2 value. Again, this value is seemingly low but since a user can only give discrete values for the ratings and our model predicts continuous variables, we would get an error value, for instance, if we predicted a 4.1 rating when the actual was a 4 star rating. Given this, we still achieve an OSR^2 of **0.51.** We predict if we rounded our predicted values, our OSR^2 would be much higher but we would sacrifice granularity in our prediction since we think understanding the difference between a 4.1 rating and a 4 rating is important for a business owner.

Notable Features

Our primary purpose of the model is to find which features of a business are highly related to the rating so we can look at the final coefficient values corresponding to our 343 features and look at the ones with the largest absolute value. Expectedly, words such as "excellent" and "fast" have larger coefficient values than most. From this, we can deduce that a user is more likely to give a higher rating if they use these words, but since this is pretty intuitive, we want to look at words with large coefficients that might give us some actual predictive insight. For example, the feature with the highest coefficient value was BYOB, shown below. This is not only intuitive, that a BYOB policy induces positive reactions from the customer, but also backed by our data as having a BYOB policy contributes very largely to a positive rating. We can see that the coefficient value is extremely high which we have two possible explanations for:

1. Not only is this an intuitively high value, but it could be the case that in Phoenix, AZ, where there are a lot of bars/restaurants, having a BYOB policy is extremely popular with customers
2. We are using 343 predictors. Since this number is so high, it would not be surprising to find that some features have little weight in the final rating (small coefficient) while other features might have significant weight to the rating, such as BYOB.



	features	coef
20	BYOB	7.324344e+08
22	caters	3.948318e+08
23	corkage	3.394917e+08
21	accepts_credit_cards	2.909079e+08
27	happy_hour	2.078505e+08

3. Multicollinearity with other values is potentially resulting in a high VIF. Some of these features with high coefficients may be commonly associated with other features, which balances out the coefficient value, but there are also possibly a small number of datapoints where that association is not present, leading to anomalies.

Appendix

All code was done in Jupyter notebook. The code is provided below but we recommend referencing the submitted document for our “Project Code Submission” as that is a PDF file with all of the below code with the Jupyter notebook output.

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

df = pd.read_csv('final_df.csv')
df.head()

df = df.drop(["Unnamed: 0", "city", "postal_code"], axis = 1)
df.attributes = df.attributes.str.lower()
df.business_id = df.business_id.str.lower()
df.review_id = df.review_id.str.lower()
df.user_id = df.user_id.str.lower()
df.head()

dfNoReview = df.iloc[:,0:8]
dfNoReview = dfNoReview.drop_duplicates()
dfNoReview

string = dfNoReview.attributes[0]
# test = string.split("\',")
# display(dfNoReview.head())
# type(string)
dfNoReview['full_bar'] = np.where(dfNoReview.attributes.str.contains("full_bar"),
1, 0)

dfNoReview['ambience_intimate'] =
np.where(dfNoReview.attributes.str.contains("intimate\': true"), 1, 0)
dfNoReview['ambience_classy'] =
np.where(dfNoReview.attributes.str.contains("classy\': true"), 1, 0)
dfNoReview['ambience_hipster'] =
np.where(dfNoReview.attributes.str.contains("hipster\': true"), 1, 0)
dfNoReview['ambience_divey'] =
np.where(dfNoReview.attributes.str.contains("divey\': true"), 1, 0)
dfNoReview['ambience_touristy'] =
np.where(dfNoReview.attributes.str.contains("touristy\': true"), 1, 0)
dfNoReview['ambience_trendy'] =
np.where(dfNoReview.attributes.str.contains("trendy\': true"), 1, 0)
dfNoReview['ambience_upscale'] =
np.where(dfNoReview.attributes.str.contains("upscale\': true"), 1, 0)
dfNoReview['ambience_casual'] =
np.where(dfNoReview.attributes.str.contains("casual\': true"), 1, 0)

dfNoReview['BYOB'] = np.where(dfNoReview.attributes.str.contains("BYOB\':
\'true"), 1, 0)
dfNoReview['accepts_credit_cards'] =
np.where(dfNoReview.attributes.str.contains("BusinessAcceptsCreditCards\':
\'true"), 1, 0)
```

```
dfNoReview['caters'] = np.where(dfNoReview.attributes.str.contains("Caters\':
\'true"), 1, 0)
dfNoReview['corkage'] = np.where(dfNoReview.attributes.str.contains("Corkage\':
\'true"), 1, 0)
dfNoReview['drivethru'] =
np.where(dfNoReview.attributes.str.contains("DriveThru\': \'true"), 1, 0)
dfNoReview['good_for_dancing'] =
np.where(dfNoReview.attributes.str.contains("GoodForDancing\': \'true"), 1, 0)
dfNoReview['good_for_kids'] =
np.where(dfNoReview.attributes.str.contains("GoodForKids\': \'true"), 1, 0)
dfNoReview['happy_hour'] =
np.where(dfNoReview.attributes.str.contains("HappyHour\': \'true"), 1, 0)
dfNoReview['has_tv'] = np.where(dfNoReview.attributes.str.contains("HasTV\':
\'true"), 1, 0)
dfNoReview['outdoor_seating'] =
np.where(dfNoReview.attributes.str.contains("OutdoorSeating\': \'true"), 1, 0)
dfNoReview['casual_attire'] =
np.where(dfNoReview.attributes.str.contains("RestaurantsAttire\': \'casual"), 1,
0)
dfNoReview['delivery'] =
np.where(dfNoReview.attributes.str.contains("RestaurantsDelivery\': \'True"), 1,
0)
dfNoReview['goodForGroups'] =
np.where(dfNoReview.attributes.str.contains("RestaurantsGoodForGroups\': \'true"),
1, 0)
dfNoReview['reservations'] =
np.where(dfNoReview.attributes.str.contains("RestaurantsReservations\': \'true"),
1, 0)
dfNoReview['tableservice'] =
np.where(dfNoReview.attributes.str.contains("RestaurantsTableService\': \'true"),
1, 0)
dfNoReview['takeout'] =
np.where(dfNoReview.attributes.str.contains("RestaurantsTakeOut\': \'true"), 1, 0)
dfNoReview['bike_parking'] =
np.where(dfNoReview.attributes.str.contains("BikeParking\': true"), 1, 0)
dfNoReview['freeWifi'] = np.where(dfNoReview.attributes.str.contains("WiFi\':
\'free"), 1, 0)
dfNoReview['wheelchair_access'] =
np.where(dfNoReview.attributes.str.contains("WheelchairAccessible\': \'true"), 1,
0)

dfNoReview['parking_garage'] =
np.where(dfNoReview.attributes.str.contains("garage\': true"), 1, 0)
dfNoReview['parking_street'] =
np.where(dfNoReview.attributes.str.contains("street\': true"), 1, 0)
dfNoReview['parking_validated'] =
np.where(dfNoReview.attributes.str.contains("validated\': true"), 1, 0)
dfNoReview['parking_lot'] = np.where(dfNoReview.attributes.str.contains("lot\':
true"), 1, 0)
dfNoReview['parking_valet'] =
np.where(dfNoReview.attributes.str.contains("valet\': true"), 1, 0)

dfNoReview['casual_attire'] =
np.where(dfNoReview.attributes.str.contains("RestaurantsAttire\': \'casual"), 1,
0)
#
```

```
dfNoReview['good_for_dessert'] =
np.where(dfNoReview.attributes.str.contains("dessert\': true"), 1, 0)
dfNoReview['good_for_latenight'] =
np.where(dfNoReview.attributes.str.contains("latenight\': true"), 1, 0)
dfNoReview['good_for_lunch'] =
np.where(dfNoReview.attributes.str.contains("lunch\': true"), 1, 0)
dfNoReview['good_for_dinner'] =
np.where(dfNoReview.attributes.str.contains("dinner\': true"), 1, 0)
dfNoReview['good_for_breakfast'] =
np.where(dfNoReview.attributes.str.contains("breakfast\': true"), 1, 0)
dfNoReview['good_for_brunch'] =
np.where(dfNoReview.attributes.str.contains("brunch\': true"), 1, 0)

dfNoReview.head()
```

```
len(df.name.unique())

len(df.business_id.unique())

df.groupby(["name", "business_id"]).count()

len(df.user_id.unique())

mean_score = df.groupby("name").mean().loc[:,["review_count", "stars_x", "cool", "funny", "stars_y", "useful"]]
mean_score.head()

fig, ax = plt.subplots(2, 2, figsize=(10, 7))
sns.distplot(mean_score.review_count, color="g",ax=ax[0,0])
sns.distplot(mean_score.stars_x, color="r",ax=ax[0,1])
sns.distplot(mean_score.funny, color="m",ax=ax[1,0])
sns.distplot(mean_score.useful, ax=ax[1,1])

g=sns.pairplot(mean_score)
g.fig.set_size_inches(9,9)
```

```
from datetime import datetime
df['new_date'] = pd.to_datetime(df['date'])
df['year'], df['month'] = df['new_date'].dt.year, df['new_date'].dt.month
df['day_of_month'] = df['new_date'].dt.day
df['day_of_week']= df['new_date'].dt.dayofweek
df["text_length"] = [len(df.text[i]) for i in np.arange(len(df))]
df.head()
```

```
df.year.value_counts().sort_index().plot(kind="bar")
plt.title("Reviews by Year")
plt.ylabel("Number of Review")

df.month.value_counts().sort_index().plot(kind="bar")
plt.title("Reviews by Month")
plt.ylabel("Number of Review")

df.day_of_week.value_counts().sort_index().plot(kind="bar")
plt.title("Reviews by Day of Week")
plt.ylabel("Number of Review")
```

```
df.day_of_month.value_counts().sort_index().plot(kind="bar")
plt.title("Reviews by Day of Month")
plt.ylabel("Number of Review")

plt.figure(figsize=(12,8))
ax = sns.scatterplot(x="funny", y="useful", size="stars_y",data=df)

df.text[0]

df_text = df.loc[:, ["business_id", "name","review_id", "stars_x", "stars_y","useful","text", "text_length"]]
df_text.head()

sent = pd.read_csv("vader_lexicon.txt", sep='\t', header=None).iloc[:, :2]
sent.columns = ["token", "polarity"]
sent.set_index("token", inplace=True)
sent.tail()

df.iloc[1,13]

sentiment = []
for review in np.arange(len(df)):
    sentiment.append(sum([sent.loc[i]["polarity"] for i in df.iloc[review,13].split() if i in sent.index]))

sentiment_series=[i for i in np.arange(len(sentiment)) if type(sentiment[i])!=
np.float64]

df_text["sentiment"]=sentiment
df_text = df_text.drop(index = sentiment_series)
df_text.head()

df_text.to_csv("df_sentiment.csv")

df_text = pd.read_csv("df_sentiment.csv")
df_text.head()

plt.figure(figsize=(12,8))
ax = sns.scatterplot(x="sentiment", y="useful", size="stars_y",data=df_text)
plt.title("The Effect of Sentiment & Usefulness on Review Rating")

plt.figure(figsize=(12,8))
ax = sns.scatterplot(x="sentiment", y="stars_y", data=df_text)
plt.title("Sentiment vs. Review Rating")

np.arange(8, len(dfNoReview.columns)+1)

one_hot = dfNoReview.iloc[:, [1,8,  9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24,25, 26, 27, 28, 29, 30, 31, 32,
33, 34, 35, 36, 37, 38, 39, 40, 41,42, 43, 44, 45, 46]]
one_hot.head()

merged=df.merge(one_hot, left_on="business_id", right_on="business_id", how="left")
final=merged.merge(df_text[["review_id","sentiment"]],left_on="review_id", right_on="review_id", how="right")
final.head()
```

```
final = pd.read_csv("final.csv")
final.year
```

```
from nltk.corpus import stopwords
s=set(stopwords.words("english"))
# temp = final.iloc[0:1, :]
# for txt in temp['text']:
#           print(txt.split())
#           print(filter(lambda w: not w in s, txt.split()))
#           txt = " ".join(a)
# final['text'][0]
```

```
from nltk.stem import PorterStemmer
import re
from nltk.tokenize import sent_tokenize, word_tokenize
ps = PorterStemmer()
# temp = final.iloc[0:1, :]
newtxt = []
for txt in final['text']:
    words = word_tokenize(txt)
    temp = []
    for w in words:
            if w in s:
            continue
            temp.append(ps.stem(w))
    temp=re.sub(r'[^\w\s]','',str(temp))
    newtxt.append("".join(temp))
# newtxt=re.sub(r'[^\w\s]','',str(newtxt))
newtxt[0]
```

```
final["newtext"]=newtxt
final.head()
```

```
from collections import Counter
mostFreq = Counter(" ".join(final["newtext"]).split()).most_common(300)
top300 = [tuple for tuple in mostFreq]
top300
# filter(lambda tuple: not tuple[0] in s, top300)
len(mostFreq)
```

```
newCommon = []
for tuple in mostFreq:
    if tuple[0] not in s:
            newCommon.append(tuple[0])
```

```
# newCommon
```

```
common_words.to_csv("common_words.csv")
final
```

```
for i in range(226,len(newCommon)):
       final['{}'.format(newCommon[i])] = final.apply(lambda row: newCommon[i] in
row.newtext, axis=1)
```

```
final = final.drop(["Unnamed: 0", "attributes", "categories", "hours", "name"], axis =1 )
final.head()
```

```
final.to_csv("final2.csv")
```

```
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

```
final=pd.read_csv("final2.csv")
```

```
y=final.stars_y
# y=final.groupby("business_id").mean()[["stars_y"]]
# x=temp
x=final.drop(['Unnamed: 0','business_id',"stars_y", "text", "newtext", "user_id",
"review_id", "date", "new_date"], axis =1)
# x = final.iloc[:, [5,7,8, 9,11,15,18,19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30,
31, 32, 33,
#     34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50,
#     51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61,62]]
```

In [14]:

```
x.columns[:100]
```

```
x.columns[100:200]
```

```
x.columns[200:300]
```

```
x.columns[300:]
```

```
vif = pd.DataFrame()
vif["VIF Factor"] = [variance_inflation_factor(x.values, i) for i in range(x.shape[1])]
vif["features"] = x.columns
```

```
vif.round(1)
```

```
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.33, random_state=42)
```

```
# Create linear regression object
regr = linear_model.LinearRegression()
# Train the model using the training sets
regr.fit(X_train, y_train)
# Make predictions using the testing set
```

```
y_pred = regr.predict(X_test)
y_pred_train = regr.predict(X_train)
# The coefficients
print('Coefficients: \n', regr.coef_)
# The mean squared error
print("Mean squared error: %.2f"
            % mean_squared_error(y_test, y_pred))
# R2 score: 1 is perfect prediction
print('R2 score: %.2f' % r2_score(y_test, y_pred))
# print(y_pred[:10])
# print(y_test[:10])

coeff = pd.DataFrame({"features": x.columns, "coef": regr.coef_})
coeff.sort_values(by="coef", ascending=False)
# for i in np.arange(len(regr.coef_)):
#           print(x.columns[i], regr.coef_[i])

import statsmodels.api as sm
model = sm.OLS(y_train, X_train)
results = model.fit()
print(results.summary())

## Out of Sample R2
y_ols = results.predict(X_test)
r2_score(y_test, y_ols)

from sklearn.linear_model import RidgeCV
clf = RidgeCV(alphas=[1e-3, 1e-2, 1e-1, 1]).fit(X_train, y_train)
clf.score(X_test, y_test)

coeff_used = np.sum(abs(clf.coef_)>0.05)
print("number of features used: ", coeff_used)

from sklearn.linear_model import LassoCV
reg = LassoCV(cv=5, random_state=0).fit(X_train, y_train)
reg.score(X_test, y_test)

coeff_used = np.sum(reg.coef_!=0)
print("number of features used: ", coeff_used)
```