Mining Tripadvisor Data Using Web Scraping and a Kaggle Data Set

By: Ronen Reouveni

Date: 12/06/2020

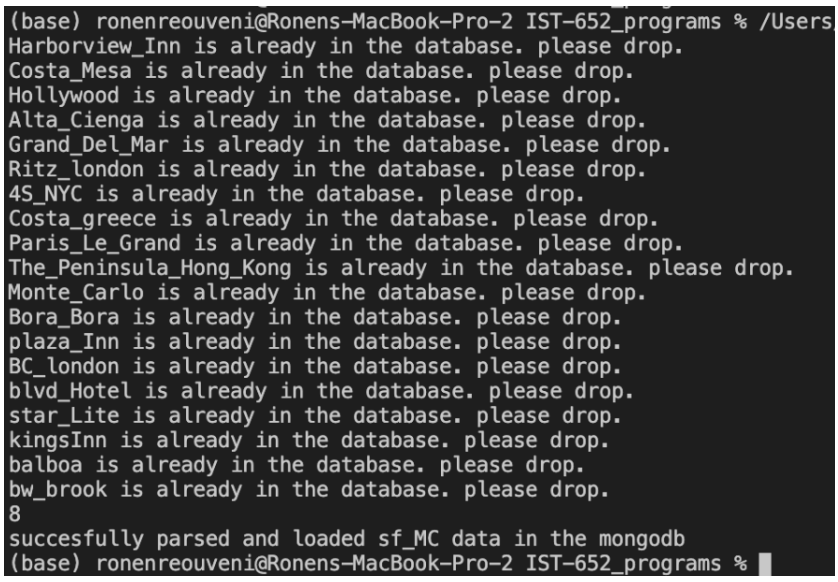
Data and its Source

The data in this report comes from tripadvisor.com and Kaggle. These two sources are combined to form the data used in analysis. The Kaggle dataset is simply 20k hotel reviews and their corresponding rating. These reviews however, come with stop words already removed. To use another source of data 20 hotels are selected and their data is scrapped using BeautifulSoup. Another 13,593 reviews are captured this way. Unlike the Kaggle dataset, the web scrapper is able to provide the average price, address, average review, and hotel name as well as the individual reviews and ratings. The hotels selected are from all over the world and have different price points and average reviews. I worked alone for this project.

Description of your data exploration and data cleaning steps

The web scrapper used to get the data from Tripadvisor uses mongo to store the data, and BeautifulSoup and urllib to extract the data from the website. A Tripadvisor hotel page has some properties worth mentioning. The home page of a hotel is what I will call a base link. When the next button is pressed to see the next page of reviews, a new link is generated. We call this the cycle link. Finally, each time the next page of reviews is loaded this cycle link has a number that simply increments by 5. These properties were used in building the web scrapper. Furthermore, the web scrapper is built as a function so it can be called on any hotel desired. The function takes 4 parameters; the base link, cycle link, number of pages to increment, and a title. When the function is called it checks if the title passed in matches any in the mongo DB. This way, we do not add data to the database more than once. The Tripadvisor website has html classes that define the structure of the site. BeautifulSoup is able to target html classes and take the necessary information. With this, get_text() is used to take just the written text. When get_text() does not work, I use .replace() to filter out the data needed. An issue arrises with the structure of the data when scrapped. The structure of the data becomes a list of lists. Where each inner list is the reviews from a single page. That is, each list within the main list is a page of reviews. We iterate through the list of lists and unpack them into a single structure. The data at the end is saved as a dictionary, it consists of individual reviews, ratings, hotel address, name, average price, and average rating. To keep things more organized, a separate script is made to handle calling this function to actually scrape and insert the data.

This is the output from calling the insertion script. Notice how it shows that the hotels are already in the database, except for one which it inserts.



From here, we can start to query the database to make sure we are returning the data that we need. This takes place in queries.py, the main script that does the analysis. To start out, I build helper functions that interact with the mongo database. Here is the code and the output from them.

```python
#function for showing all of a key in the db
def print_All_FromDB(target):
    for col in collections:
        mycol = mydb[col]
        docs = mycol.find()
        for doc in docs:
            print(doc[target])

#function for showing name and a target of choosing
def print_All_FromDB_withName(target):
    for col in collections:
        mycol = mydb[col]
        docs = mycol.find()
        for doc in docs:
            print(doc['Name'] , ',' ,doc[target])

#define another function
def print_price_ratings(target = 'Price', target_2 = 'Rating'):
    for col in collections:
        mycol = mydb[col]
        docs = mycol.find()
        for doc in docs:
            print(doc[target], ',',doc[target_2])

#call functions
print_All_FromDB('Address')
print_All_FromDB_withName('Price')
print_price_ratings()
```

Addresses

```
7110 Hollywood Blvd, Los Angeles, CA 90046-3203
5300 Grand Del Mar Court, San Diego, CA 92130-4901
22 Upper Berkeley Street Paddington, London W1H 7QG England
Salisbury Road, Tsim Sha Tsui, Hong Kong China
3751 6th Ave, San Diego, CA 92103-4337
1220 W Main St, Stroudsburg, PA 18360-1326
750 Ocean Dr, Miami Beach, FL 33139-6220
Navarino Dunes, Costa Navarino 24001 Greece
150 Piccadilly, London W1J 9BR England
Calle Luis Moya No. 11 Centro Historico, Mexico City 06000 Mexico
550 W Grape St, San Diego, CA 92101-2207
57 E 57th St, New York City, NY 10022-2081
Motu Tehotu BP 547, 98730 French Polynesia
Place Du Casino, Monte-Carlo 98000 Monaco
2 Rue Scribe, 75009 Paris France
1333 Hotel Circle South, San Diego, CA 92108-3408
2632 W 13th St, Brooklyn, NY 11223-5815
1005 N La Cienega Blvd, West Hollywood, CA 90069-4105
1951 Newport Blvd, Costa Mesa, CA 92627-2250
740 Ocean Dr, Miami Beach, FL 33139-6220
```

Name and price

```
Hollywood La Brea Motel , 121.5
Fairmont Grand Del Mar , 650.5
Berkeley Court Hotel , 111.5
The Peninsula Hong Kong , 576
ITH Colive Balboa Park , 62
Pocono Plaza Inn , 136.5
Starlite Hotel , 157
The Romanos, A Luxury Collection Resort, Costa Navarino , 748
The Ritz London , 784.5
Hotel San Francisco , 51.5
Harborview Inn and Suites , 106.5
Four Seasons Hotel New York , 1388
Four Seasons Resort Bora Bora , 1619
Hotel De Paris Monte-Carlo , 774.5
InterContinental Paris Le Grand , 496.5
Kings Inn San Diego , 126
Best Western Brooklyn-Coney Island Inn , 165
Alta Cienega Motel , 98.5
Travelodge by Wyndham Costa Mesa Newport Beach Hacienda , 95.5
Boulevard Hotel , 176
```

Price and average rating

```
121.5 , 3.235294117647059
650.5 , 4.7965
111.5 , 1.4263157894736842
576 , 4.732374100719425
62 , 4.793333333333333
136.5 , 2.1303030303030304
157 , 3.0050632911392405
748 , 4.628415300546448
784.5 , 4.7675
51.5 , 2.8
106.5 , 2.4955357142857144
1388 , 4.618181818181818
1619 , 4.779
774.5 , 4.515966386554622
496.5 , 4.3425
126 , 4.56
165 , 4.571428571428571
98.5 , 2.507042253521127
95.5 , 2.690307328605201
176 , 1.6896551724137931
```

```python
#very important function that takes data from mongodb and stores in a pandas frame
def createDF(data):
    mycol = mydb[str(data)]
    docs = mycol.find()
    myData = []
    for doc in docs:
        review = doc['individual_Reviews']['Reviews']
        rating = doc['individual_Reviews']['Ratings']
        myData.append([review, rating])
    structure = {'Reviews': myData[0][0], 'Ratings': myData[0][1]}
    return(pd.DataFrame(structure))

#get data for question hw question
def makeList_price_rating(target_1, target_2):
    global priceList
    priceList = []
    global ratingList
    ratingList = []
    for col in collections:
        mycol = mydb[col]
        docs = mycol.find()
        for doc in docs:
            priceList.append(doc[target_1])
            ratingList.append(doc[target_2])
```

The createDF function shown above is called on each collection in the mongoDB. Each collection is a different hotel. Then pd.concat is used to combine all of them into a single data frame. Furthermore, the Kaggle data is also used in the pd.concat to include data from both sources. This is a screenshot from calling head() on the data frame containing all the data, from the scrapper and Kaggle.

	Reviews	Ratings
0	nice hotel expensive parking got good deal sta...	4
1	ok nothing special charge diamond member hilto...	2
2	nice rooms not 4* experience hotel monaco seat...	3
3	unique, great stay, wonderful time hotel monac...	5
4	great stay great stay, went seahawk game aweso...	5

At this point we loop through all the reviews and we call the Vader sentiment analyzer from NLTK. This sentiment analyzer gives each review a negative score, positive score, neutral score, and compound score. The compound score is a combination of all other scores. We find these sentiment scores for each review and save them to new lists to be added to the data frame at a later point. Other preprocessing is needed before they can be added. This is the code that accomplishes this. We dive deeper into the sentiment scores later on in the preprocessing. The compound score ranges from -1 to 1, where 1 is good sentiment and -1 is bad sentiment. The other scores range from 0 to 1 and closer to 1 implies higher score in that category.

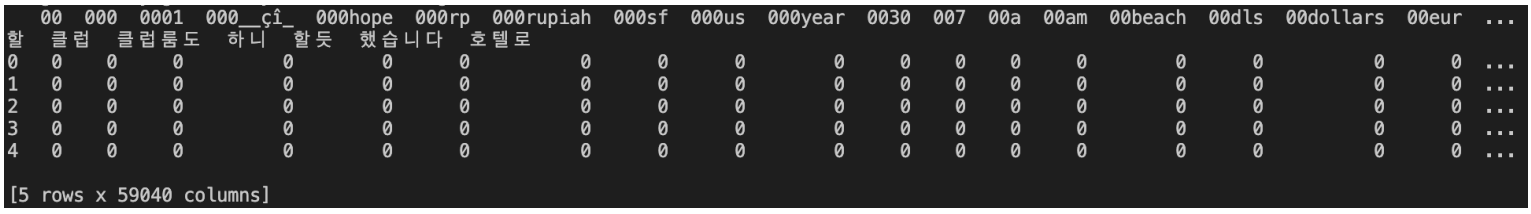
```
#append everything to master words
for elems in rawFile['Reviews']:
    masterWords.append(elems)

#create sentiment analysis
sid = SentimentIntensityAnalyzer()

negative = []
neutral = []
positive = []
compound = []

#loop through all reviews and append their sentiment scores to new lists
for review in masterWords:
    ss = sid.polarity_scores(review)
    negative.append(ss['neg'])
    neutral.append(ss['neu'])
    positive.append(ss['pos'])
    compound.append(ss['compound'])
```

Next, the data needs to be count vectorized. That is, we need a data frame where each column is a word and each row is a review, where each cell is the amount of times that word appears in that review. Thankfully, NLTK has an easy way of accomplishing this. We simply use the functions CountVectorizer and .fit_transform on the list of reviews. This will accomplish what we needed. Here is an image of the output.



	00	000	0001	000__çî_	000hope	000rp	000rupiah	000sf	000us	000year	0030	007	00a	00am	00beach	00dls	00dollars	00eur	...
할	클럽	클럽룸도	하니	할듯	했습니다	호텔로													
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	...

[5 rows x 59040 columns]

The main issue here can be seen in the bottom left. We have over 59k columns. This is far too many. In fact, we have more columns than we do observations which will cause a huge problem down the line. We need to find a heuristic to remove columns. A good way of doing

this is analyzing the variance of the columns. We can do this by looking at the standard deviation for each column. If it is very low, there is not much going on in the column and it can be deleted. Columns with virtually no standard deviation or variance will be useless in prediction and they can be removed. This is the code and output that accomplishes this.

```
#view shape and remove all columns whos std is below .025
print(df.shape)
df = df.loc[:, df.std() > 0.025]
print(df.shape)
```

```
(34084, 59040)
(34084, 8264)
```

With this method we have reduced the number of columns from 59,040 all the way down to 8,264. This is much more manageable and realistic. Next, stop words are removed. This is done by seeing if any of the column names, which are words, match the stop words, and they are removed if they do. Here is the code and output that executes this.

```
#remove remaining stopwords
colNames = df.columns
print(df.shape)
goodCols = []
for i in range(0, len(colNames),1):
    if colNames[i] not in stop_words:
        goodCols.append(i)

df = df.iloc[:,goodCols]
print(df.shape)
```

```
(34084, 8264)
(34084, 8129)
```

With removing stop words we lower the column count by 135. It is not a huge amount but still necessary to do. From here, we add the 4 sentiment categories to the data frame. We add the columns; compound score, negative score, positive score, and neutral score.

The Vader sentiment scores are one of the main focuses of the analysis. The first data question outlines much of this analysis with code and output. However, to summarize we use the Vader scores to compare rating groups. We can calculate the average sentiment score given rating group. Then use a t test, and cosine similarity to gauge how similar these rating scores actually are. This gives us intuition into how and why we can collapse categories together. The code and output for calculating these group means, running t tests, creating confidence intervals, and calculating cosine similarity is all shown in detail in data question 1 later on in this report. The results of this is that rating groups 4 and 5 should be combined into one rating group but the rest should be left separated.

Another major preprocessing step is removing outliers. With calculating the sentiment scores we can compare compound score to rating. This illuminates many reviews that do not make coherent sense. The code and output as well as deeper analysis of this issue is answered in data question 3 later on in this report. 3,404 observations are removed because they are defined as outliers. The strict definition and execution is shown later in the report in question 3.

The Kaggle dataset and the scrapped data skew heavily towards 4 and 5 ratings. Therefore, we need to rebalance the dataset. There are a few options for accomplishing this but the easiest is to simply build a sampling function where the threshold is the minimum count for the groups. This will make sure there is the same number of observations in each rating group. Although this means we are throwing out a huge portion of data it will make our results much more interpretable. Without doing this, our classification rate would be inflated and not show the true results. This is the code and output of sampling the data.

```
#sample data until the group sizes match
#we need to rebalance for better classification results
minGroup = df['Ratings'].value_counts().min()

def sampling_k_elements(group, k = minGroup):
    if len(group) < k:
        return group
    return group.sample(k)

print(df['Ratings'].value_counts())
balanced = df.groupby('Ratings').apply(sampling_k_elements).reset_index(drop=True)
print(balanced['Ratings'].value_counts())
```

```
5    23385
3    2658
1    2334
2    2303
Name: Ratings, dtype: int64
5    2303
3    2303
2    2303
1    2303
```

Finally, the data frame is split into a testing and training set. This is used to make sure that we do not overfit our model. Furthermore, we take the best performing model and use cross validation to assess any overfitting that may have occurred. A Random Forest, and an Extreme Gradient Boosting model are used. Confusion matrices are built for both and analyzed in the data questions below. We also run the Random Forest on the data with the Vader sentiment scores and the data without to see if it improves accuracy of the model.

Data questions

1. How similar is the sentiment between adjacent reviews? For example, how different is the sentiment between the reviews with a 5 vs a 4?

    Unit of analysis: Compound sentiment score from the Vader lexicon is used in multiple ways to answer this question.

    Comparison: Compare the sentiment scores between adjacent ratings. We can compare the ratings group sentiment means and use a statistical significance test to see if the difference in means is significant. We can build a confidence interval around these means as well. Finally, we can use cosine similarity to see how similar two vectors are. In this case, these vectors are the sentiment scores for each rating 1-5.

To answer this question we subset the data frame to get a vector of compound sentiment scores for each rating. Once we have these vectors we then use different tools to compare adjacent ratings. The first is a simple significance test. This tool tells us if the mean sentiment score is truly different for the various ratings. We can then wrap confidence intervals around some of the means to show the band of uncertainty around these sample means. Finally, we pass adjacent vectors into a cosine similarity function that returns a distance score. If the score is close to 1 the two vectors are exactly the same, and a score close to zero means they are exactly opposite each other. Depending on the results from this analysis we can justify collapsing ratings into fewer categories. If we found that there is no significant difference between ratings 4 and 5, we can mask them into a single group. To start, we can simply compare group means.

Below is the code and output for each method followed by a final analysis and answer to the question.

Group means code and output:

```
#find means and variance
groupMean = df.groupby('Ratings')['compScore'].mean()
print(groupMean)
```

```
Ratings
1   -0.191811
2    0.326952
3    0.698443
4    0.909260
5    0.936343
```

Significance tests code and output:

```
#run significance tests to understand differences between groups
#check if the composite sentiment score is truly different for adjacent rating groups
result_5_4 = stats.ttest_ind( df.loc[df['Ratings'] == 5, 'compScore'], df.loc[df['Ratings'] == 4, 'compScore'], equal_var = False)
print(result_5_4)

#is 4 different to 3
result_4_3 = stats.ttest_ind( df.loc[df['Ratings'] == 4, 'compScore'], df.loc[df['Ratings'] == 3, 'compScore'], equal_var = False)
print(result_4_3)

#is 3 different to 2
result_3_2 = stats.ttest_ind( df.loc[df['Ratings'] == 3, 'compScore'], df.loc[df['Ratings'] == 2, 'compScore'], equal_var = False)
print(result_3_2)

#is 2 different to 1
result_2_1 = stats.ttest_ind( df.loc[df['Ratings'] == 2, 'compScore'], df.loc[df['Ratings'] == 1, 'compScore'], equal_var = False)
print(result_2_1)
```

```
Ttest_indResult(statistic=9.976401441200576, pvalue=2.4156315918243773e-23)
Ttest_indResult(statistic=23.39762765122175, pvalue=7.59549754446309e-113)
Ttest_indResult(statistic=21.957695433813807, pvalue=8.31218186109203e-101)
Ttest_indResult(statistic=24.870724514406426, pvalue=2.863087900164405e-128)
```

Confidence intervals code and output:

```
#we can do some confidence intervals around means
conf = sms.DescrStatsW(df.loc[df['Ratings'] == 5, 'compScore']).tconfint_mean()
conf_2 = sms.DescrStatsW(df.loc[df['Ratings'] == 4, 'compScore']).tconfint_mean()
print(conf)
print(conf_2)
```

```
(0.9341223068894525, 0.938563294709769)
(0.9044236646841841, 0.9140957996015309)
```

Cosine similarity code and output:

```
#extract vectors
vector5 = df.loc[df['Ratings'] == 5, 'compScore']
vector4 = df.loc[df['Ratings'] == 4, 'compScore']
vector3 = df.loc[df['Ratings'] == 3, 'compScore']
vector2 = df.loc[df['Ratings'] == 2, 'compScore']
vector1 = df.loc[df['Ratings'] == 1, 'compScore']

#calculate the cosine similarity between two adjacent ratings
#-1 strong opposite vector
#0 orthogonal to each other
#1 strong similarity
sim_5_4 = cosine_similarity([vector5[:2000]], [vector4[:2000]])
print('similarity between ratings 5 and 4', sim_5_4)

sim_4_3 = cosine_similarity([vector4[:2000]], [vector3[:2000]])
print('similarity between ratings 4 and 3', sim_4_3)

sim_3_2 = cosine_similarity([vector3[:2000]], [vector2[:2000]])
print('similarity between ratings 3 and 2', sim_3_2)

sim_2_1 = cosine_similarity([vector2[:2000]], [vector1[:2000]])
print('similarity between ratings 2 and 1', sim_2_1)
```

```
similarity between ratings 5 and 4 [[0.97007296]]
similarity between ratings 4 and 3 [[0.85662252]]
similarity between ratings 3 and 2 [[0.39383321]]
similarity between ratings 2 and 1 [[-0.1201513]]
```

Answer to question one:

There were many methods used to compare the similarity between sentiment scores within each rating group. The average compound sentiment scores for ratings 1, 2, 3, 4 and 5 are -.19, .32, .69, .91, and .94 respectively. Scores close to -1 are the worst sentiment, scores close to 1 are the best sentiment. Based simply on these results, all the ratings have very different average sentiment, except for 4 and 5. However, we can use a t test to say with statistical significance if the group means are the same or not. The results from this however, are disappointing. All the t tests show with very high confidence that the groups are all different. This result was also true for comparing the means between group 4 and 5. The results are that they are different. To dig deeper into this we can look at the 95% confidence interval for the mean sentiment score of ratings 4 and 5. These confidence intervals are .934 to .938 for group 5 and .90 to .91 for group 4. This illuminates the problem with using a t test. There are so many observations that it is able to capture very small differences between the groups. The band of uncertainty defining the confidence intervals are very small. Thus, cosine similarity is probably the most accurate tool to make the comparison. Scores close to 1 are perfectly the same, 0 means they are orthogonal, and -1 means the scores are opposites. The cosine similarity between groups 5 and 4 is .97, between groups 4 and 3 is .85, between groups 3 and 2 is .39, and between groups 2 and 1 is -.12. When calculating cosine similarity we must limit the values passed in because each vector must be the same length and in its current state they are different lengths. Cosine similarity succeeded where the other tests failed. It showed that ratings 4 and 5 are actually very similar. 4 and 3 are also similar but not to the same degree. On the other hand, 3 and 2 as well as 2 and 1 are clearly different according to every test done. This is strong evidence that we should combine groups 4 and 5.

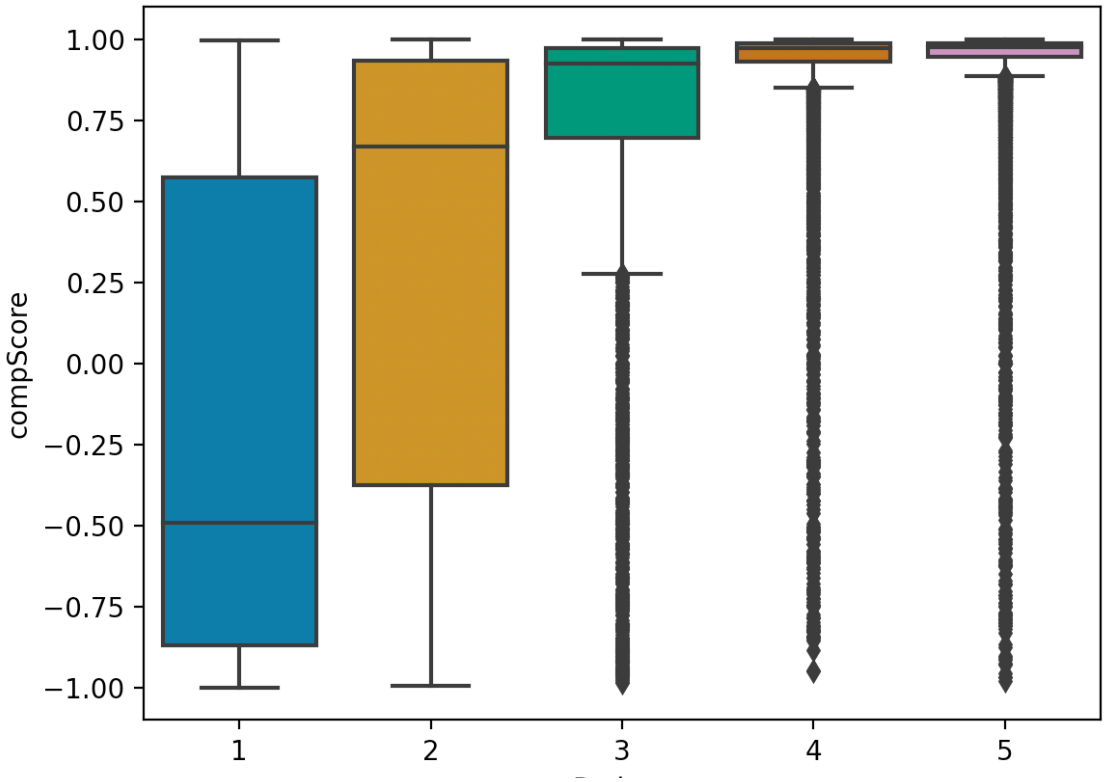
2. Can we visualize the similarity in compound sentiment between reviews?

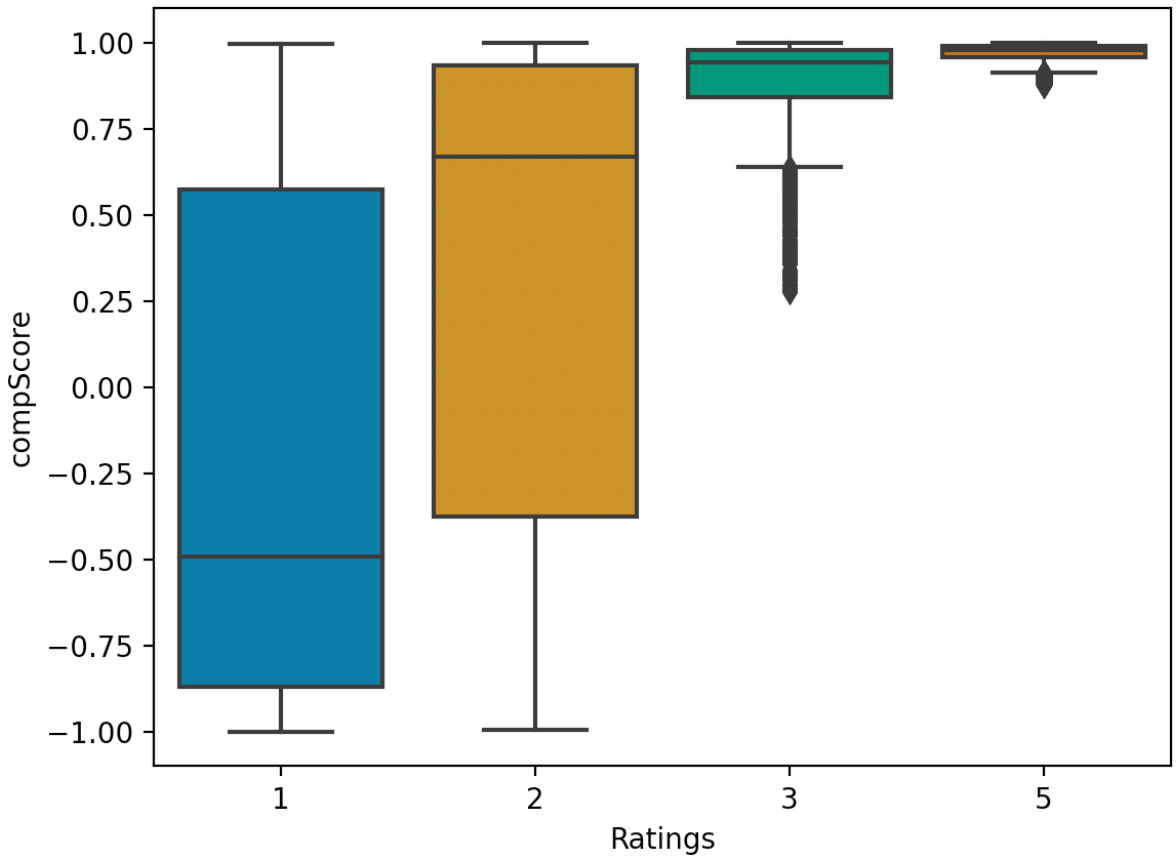
    Unit of Analysis: Boxplot and 3D scatter plots.

    Comparison: Use these visualizations to look at how similar the groups are. In the case of the 3D scatter plot we can also asses any sense of separability between the ratings.

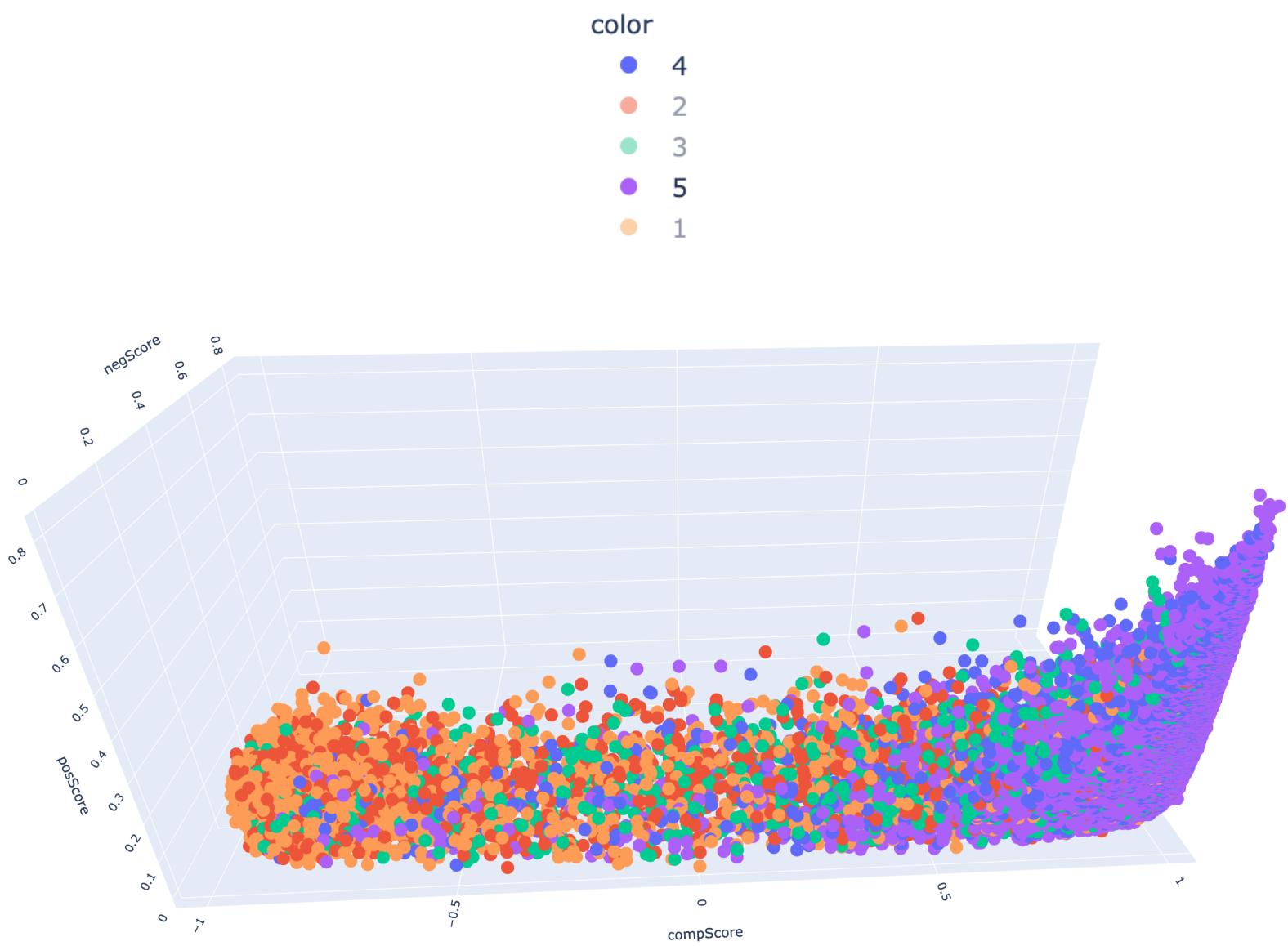
We can visualize the distribution and ranges of data easily with box plots. It is a simple matter of calling the box plot on the data frame with compound score along the y axis and the rating along the x axis. We then create a new data frame of just the sentiment scores. Compound, positive, and negative. Then we create a 3D scatter plot of the points. These visualizations are calculated multiple times. Based on the results from the previous question, I collapsed ratings 4 and 5 into one rating. Each visualization has a before and after from collapsing the groups as well as removing outliers. How outliers are removed is answered in the next question. Multiple 3D plots are shown with certain ratings turned on and off to simplify the results.

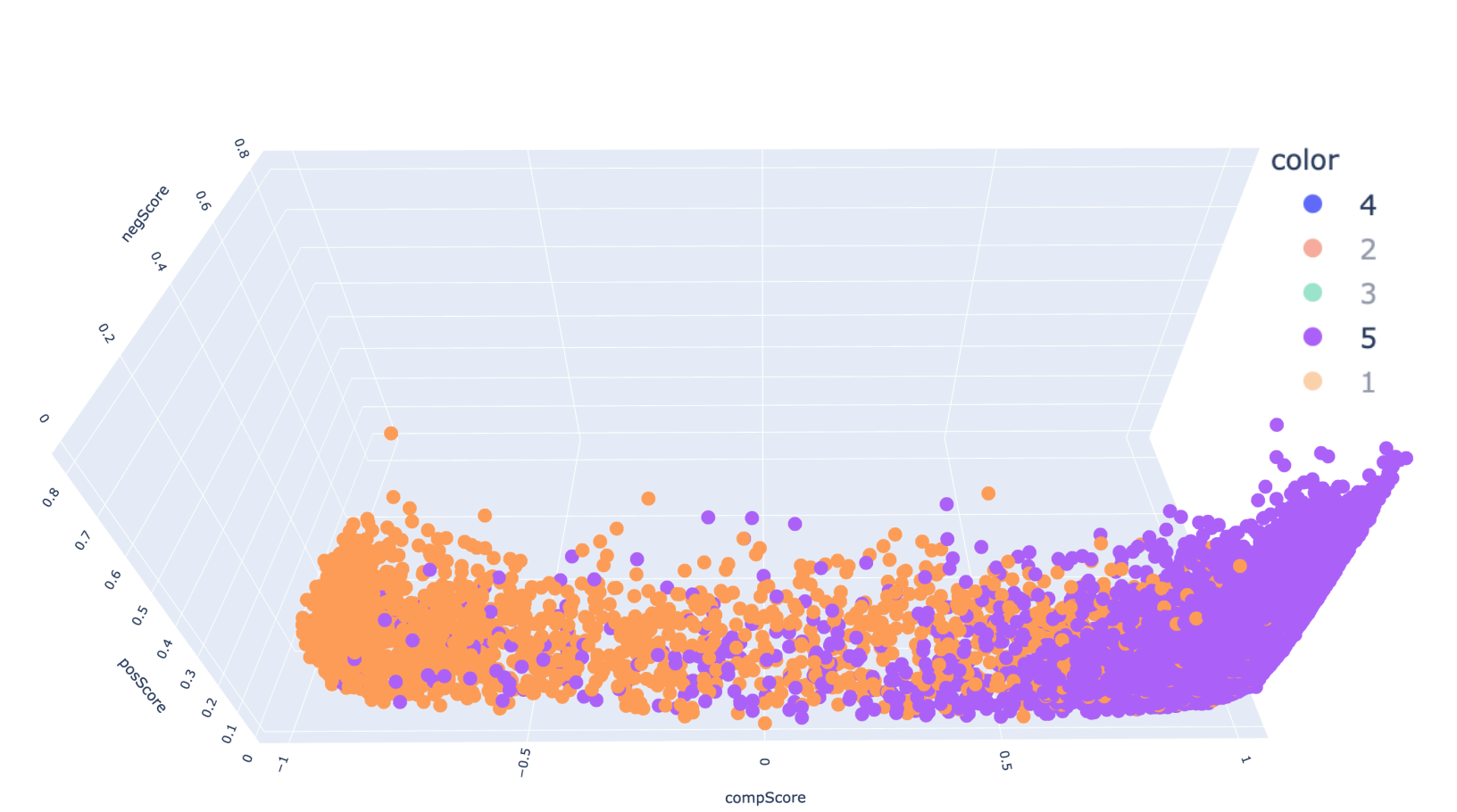
```
f3 = plt.figure(3)
sns.boxplot(y='compScore', x='Ratings',
                    data=df,
                    palette="colorblind")
```

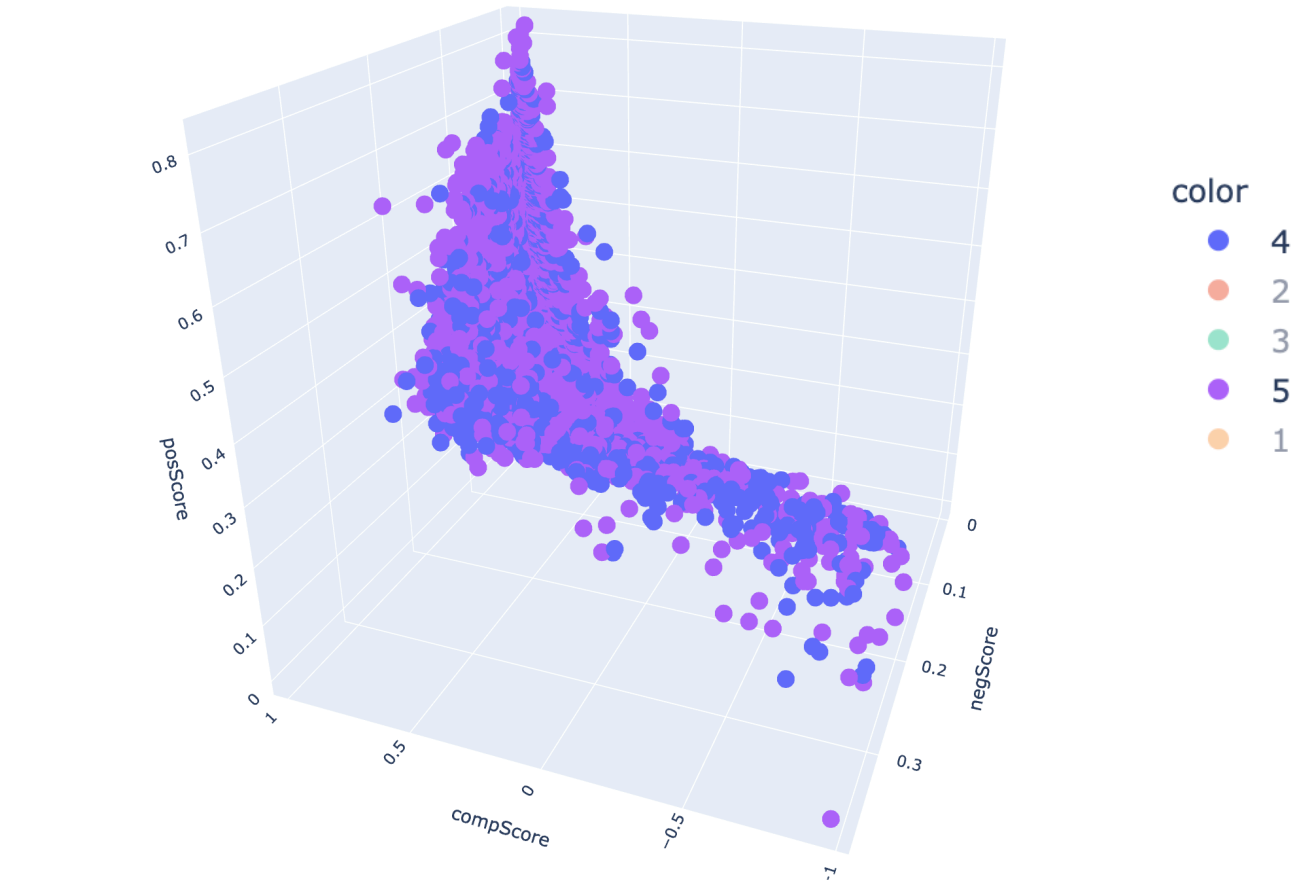




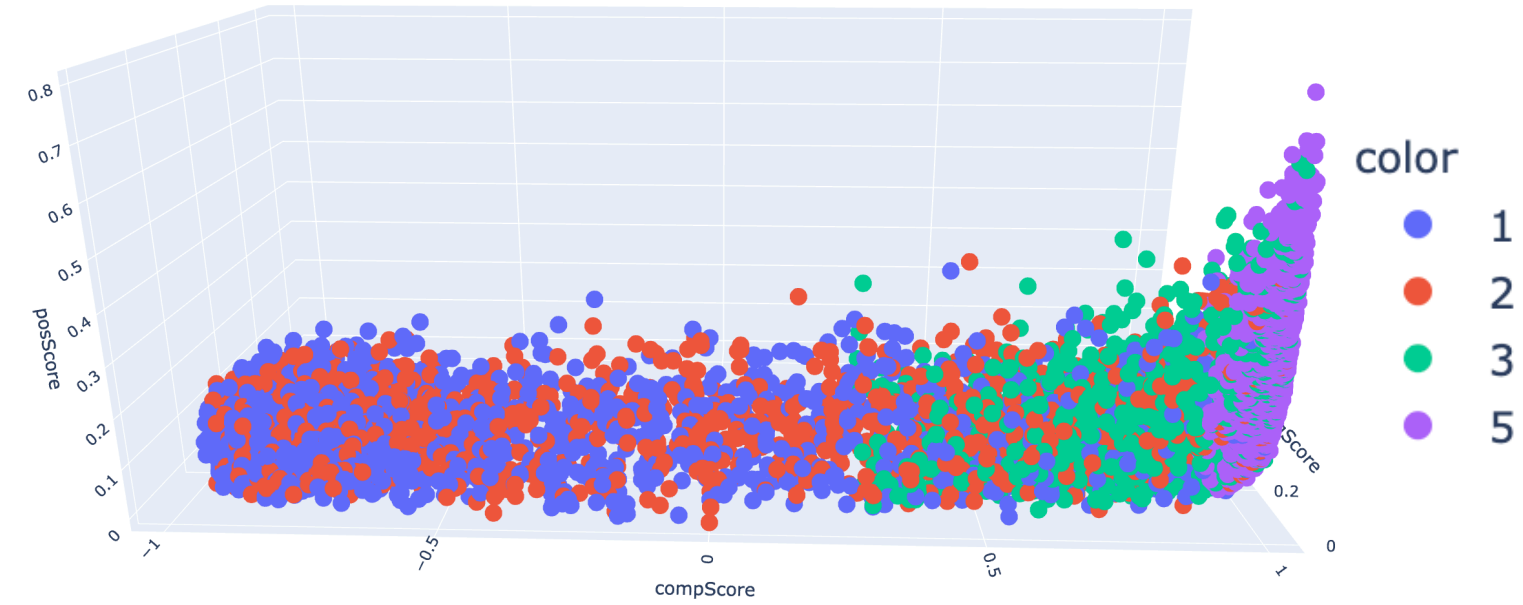
```
#visualize ratings given compScore, negScore, and posScore in three dimensions
X = df[['compScore', 'negScore', 'posScore']]
fig = px.scatter_3d(
    X, x='compScore', y='negScore', z='posScore', color= df['Ratings'].astype('category'),
    title=f'Ratings in 3D',
    labels={'0': 'compScore', '1': 'negScore', '2': 'posScore'}
)
```

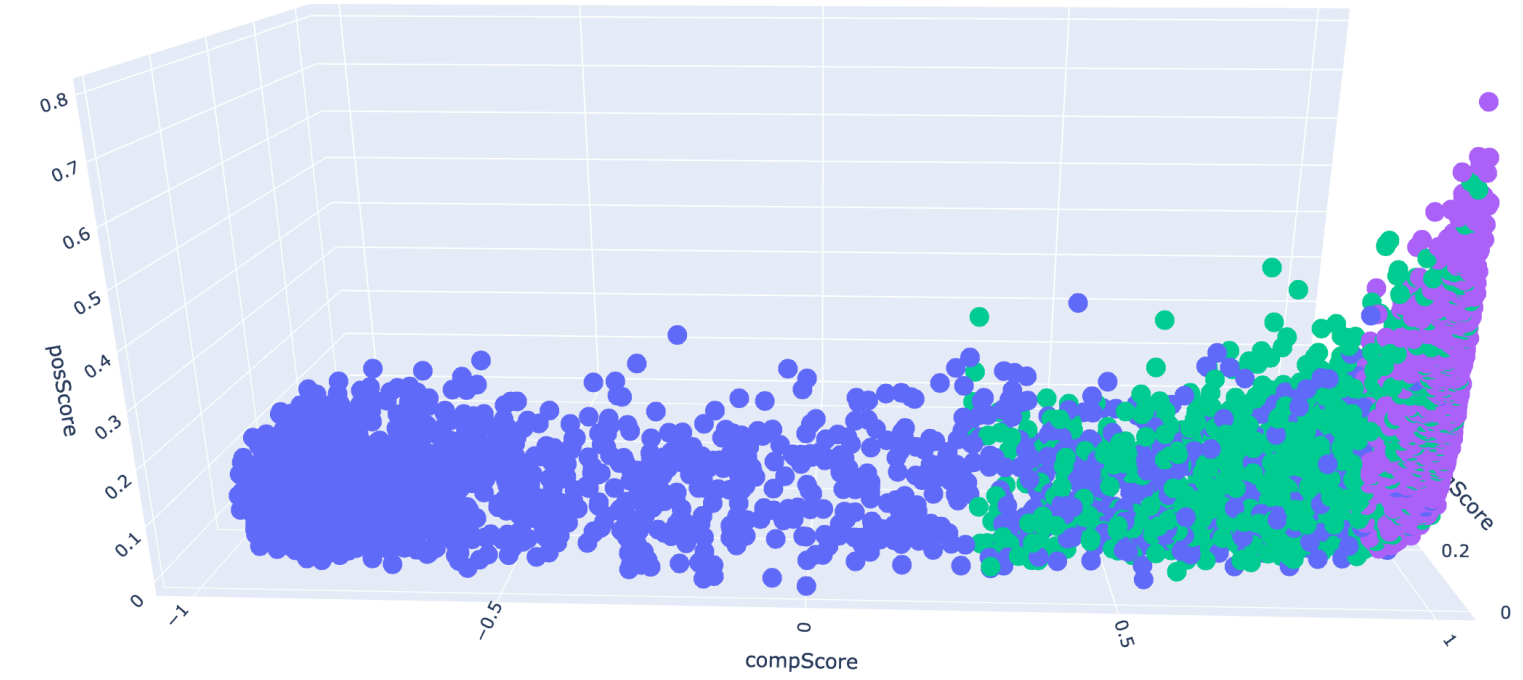


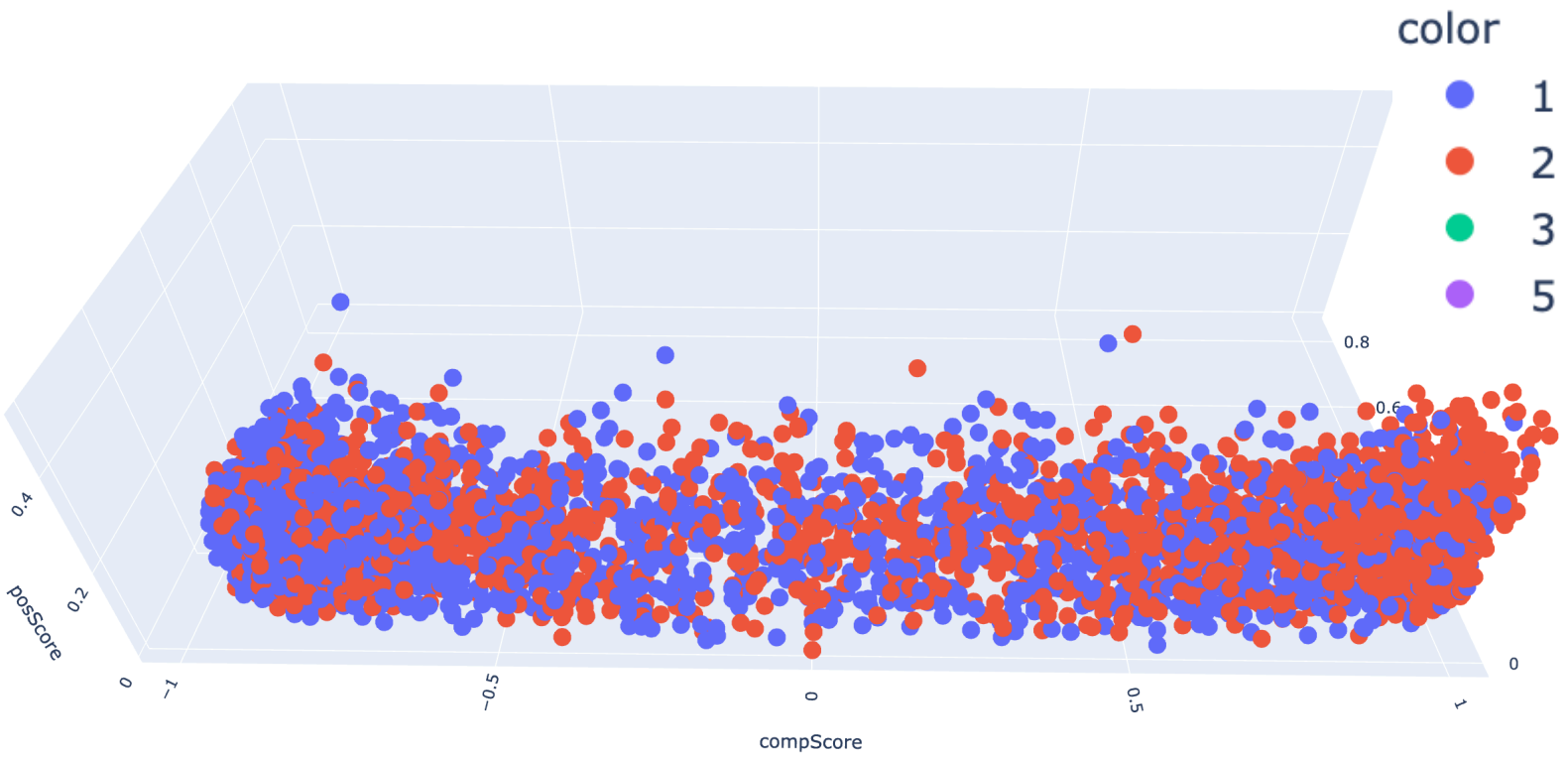

color
4
2
3
5
1
negScore
posScore
compScore


color
4
2
3
5
1
posScore
compScore
negScore

Below here the 3D plots have the outliers removed and groups 4 and 5 collapsed







Answer:

The box plots are very easy to interpret but they are very useful. They show the compound sentiment score along the y axis and the different rating group along the x axis. Ratings 1, 2, and 3 are very different. Their inner quartile ranges are clearly different to each other. However, it is also clear that ratings 4 and 5 are almost identical. They have the same inner quartile range. The box plots also make it clear that there are many outliers in rating groups 3, 4, and 5. The second box plot shows these outliers removed and rating groups 4 and 5 collapsed. As insightful as the box plot is, we can achieve very deep analysis with the 3D plots.

The 3D scatter plots have three axis. Compound score, positive score, and negative score. With these, we can directly visualize the space that the different ratings exist in. This can show the answer to the question of how similar sentiment scores are between ratings. At first glance, the initial 3D scatter plot looks like a mess. However, with a closer look there is a clear pattern and separation between ratings. We can view select ratings and then visualize again. The second 3D scatter plot shows only ratings 5 and 1. Here we can see a clear separation between these. The 3rd scatter plot shows group 4 and 5 together. Not to any surprise, there is no difference between these groups. They look completely intertwined with one another with no discernible separation. The next three scatter plots now have outliers removed and groups 4 and 5 combined. Even the first scatter plot in this second group shows a clear pattern 1-5. Only 1 and 2 seem similar to each other, the rest all clearly are clustered together. To reinforce this, the second scatter plot of this group shows only ratings 1, 3 and 5. It is very clear that each one of these ratings occupies a different area in this three dimensional space. Finally, the final group shows that ratings 1 and 2 are similar. However, not by enough to justify collapsing them into a single rating group.

3. Can we define an outlier in terms of its text and corresponding review? What is an example of an observation like this?

Unit of Analysis: Quantiles for sentiment reviews given rating, and the box plot definition of an outlier.

Comparison: Sentiment vs rating

The box plot shown above has many outliers. A box plot defines an outlier as being 1.5 times above or below the bounds of the inner quartile range. We can easily find the quantiles for each group and filter out all records that are defined as an outlier. These outliers are very interesting. For example, the box plot shows that there are reviews that had a sentiment at nearly -1 and yet the review was rated as a 5. A sentiment score of -1 is the worst possible sentiment. So how can an observation have terrible sentiment and the rating as a 5?

Here is the code and output. As well as the code to calculate a threshold and filter out all outliers. Notice the text of the output considered an outlier.

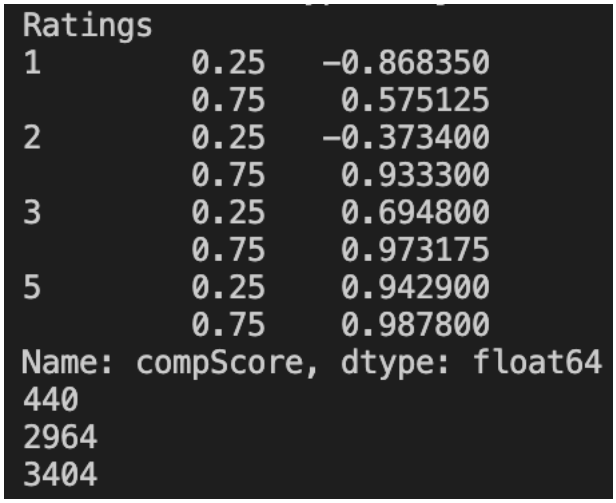
```
#bad sentiment but good review
max_1 = (df.loc[df['Ratings'] == 5]['compScore'].idxmin())
print(rawFile.iloc[max_1,:])
```

```
Reviews    Before I go ahead with my near perfect review of the Hotel de Paris, I absolutely have to tell anyo
ne reading this to AVOID going to the sister-hotel BEACH CLUB at all costs.  The Beach Club is part of the fam
ily of hotels that Hotel de Paris is part of...there is a shuttle that drives guests to and from the various m
embers of the hotel family.  You're tempted by the sound of "Beach Club," but I can't emphasize enough what a
complete waste of time and money this "Beach Club" is:  no beach, a pool scene straight out of Flamingo Kid-me
ets-Caddy Shack (and not in a good way), ridiculously expensive crappy food and completely obnoxious clientele
.  As my husband and I were paying our bill to hurridly escape the madness, I said "The only good thing about
us coming to the Beach Club is…
Ratings    5
```

```
#here I want to get all the reviews under the bottom line of 3 and 4
#we need to quants to calculate cutoff for outliers
quants = df.groupby('Ratings')['compScore'].quantile([.25,.75])
print(quants)

#calculation is the definition of an outlier as shown on the boxplot
#how many reviews are potential problems (considered outliers)
threshold_3 = 0.694800 - ((0.973100 - 0.694800)*1.5)
ids_3 = (df.index[(df['Ratings'] == 3) & (df['compScore'] < threshold_3)]).tolist()
print(len(ids_3))


threshold_5 = 0.942900 - ((0.987800 - 0.942900)*1.5)
ids_5 = (df.index[(df['Ratings'] == 5) & (df['compScore'] < threshold_5)]).tolist()
print(len(ids_5))
```



Answer:

As shown above, the review is very telling. To summarize, the review was amazing for the hotel in question, but then they go on to talk about how terrible a separate establishment in the city was. The negative aspects of the review had nothing to do with actual hotel itself. This complicates our model. Neither a sentiment analyzer nor a machine learning model can pick this nuance up easily. This is the intuition in removing these observations. It is an outlier because it's really not a review of the hotel but a review of something that went wrong on their trip outside of what they were supposed to be reviewing. To accomplish this we simply calculate the quantiles of each group and then calculate the cutoff point that defines an outlier. We also print the amount of observations removed, 3404. Furthermore, these are the observations removed that we can see in the visualizations in the previous question.

4. Can a machine learning algorithm predict if a review is rated a 1, 2, 3, or 5? Note, 4 was collapsed into five as shown in question 1.

    Unit of analysis: Classification rate and confusion matrix. We can calculate how many correct predictions we make. Furthermore, misclassifying a 2 as a 1 is not the same as misclassifying a 2 as a 5. With the confusion matrix we can see these outcomes clearly.

    Comparison: Compare the columns of confusion matrix and calculate the overall percentage of correctly classified observations.

The data frame used has all of the words used in a term document matrix and has sentiment scores added from the Vader sentiment lexicon. The dependent variable, Ratings, is removed and placed into a target object. We simply then split the data frame into training and testing sets, where the testing set is a holdout of 30% of the data. We train a Random Forest and an Extreme Gradient Boosting model. We will then use the better of the two models and test with cross validation. Previous iterations of the problem were attempted with only groups 1, 3, and 5. However, now with the added analysis of only combining 4 and 5 we are testing groups 1, 2, 3 and 5, where 5 is really 4 and 5. Close attention must be paid to the confusion matrix to really assess how the algorithm is performing.

First we show Random Forest result and output, followed by XGBoost, followed by the cross validation.

```
results of ML with sentiment scores
[[541  98  36  13]
 [228 271 181  33]
 [ 45  94 406 143]
 [  0   7 103 565]]
0.6450795947901592
```

	columns	importance
4250	lotions	0.000000
589	arii	0.000000
590	arizona	0.000000
592	armes	0.000000
6980	stung	0.000000
...	...	...
3253	great	0.006834
8131	neuScore	0.012603
8129	negScore	0.032784
8130	posScore	0.034311
8132	compScore	0.046482

```
results of XGBoost
[[493 133  54   8]
 [187 317 186  23]
 [ 19 118 425 126]
 [  0  10 103 562]]
0.6501447178002895
```

Cross validation, list of individual scores followed by the lists mean.

```
[0.63497232 0.64115923 0.63485342]
0.6369949911451422
```

Answer:

The classification rate for the Random Forest is .645. This means that 64.5% of the testing data was correctly classified. When using the XGBoost model the classification rate increases slightly to .65 or 65%. This however, may not be significant when taking a deeper look. Either way, we will analyze the confusion matrix for the XGBoost model. Although the classification rate is only 65% this does not tell the whole story. Column 1 of the confusion matrix is for ratings of 1 star. We can see that 493 of the observations are correctly classified as a rating of 1. It misclassifies 187 as a rating of 2, but only misclassifies 19 as a 3 and 0 as 4 and 5. This shows that we cannot really separate out 1 and 2. The algorithm knows that the review should be a 1 or a 2. Often times, the difference between a 1 and 2 is just a persons emotional choice. That is, two reviews could have the exact same words but someone may decide it's a 1 and others a 2. This theme is true as we look at the other groups. The misclassifications are centered around adjacent ratings. We can keep in mind the 3D visualizations. There was a clear trend in moving along sentiment score how the reviews were changing. However, the exact point where it switched was never clear. This can be seen in this confusion matrix. Finally, we use cross validation on the XGBoost model with 3 folds. Only 3 folds are used because of the immense time it takes to run these models. The cross fold validation has a mean score of .637. It has come down from .65. This may imply there is some overfitting, but not very much. These seem to be stable enough results to use a model like this.

5. Does including sentiment score with the term document matrix significantly improve the accuracy?

    Unit of analysis: Classification rate, the percent of correctly predicted observations.

    Comparison: Simply compare classification rate from the same model used on 2 different sets of data.

All we need to do is take our final data frame and copy it while deleting the Vader sentiment scores. These scores are compound score, negative score, positive score, and neutral score. Once we have these two data frames it is a matter of training a Random Forest on both. We use a Random Forest model because it is fast to run and traditionally stable.

```
results of ML with sentiment scores
[[541  98  36  13]
 [228 271 181  33]
 [ 45  94 406 143]
 [  0   7 103 565]]
0.6450795947901592
```

```
results of ML with no sentiment scores
[[536  93  34  25]
 [221 217 201  74]
 [ 65  98 359 166]
 [  4  14 118 539]]
0.5973227206946454
```

The classification rate is much higher when the model includes sentiment scores. It rises from .597 to .645. This is a massive increase and shows that even though the sentiment scores are based on the text, it adds value to the model.

Brief description of the program

This program is broken down into three scripts. First is the web scrapper used to pull additional data from the Tripadvisor website and store it in a mongo database. The scrapper is built as a function so it can be called on multiple hotels. It takes 4 parameters, the base link, cycle link, number of pages to parse, and a title. These are described in more detail at the beginning of this report. It is built in this way so it can be called on multiple hotels. A consequence of this is that we may add data to the database twice and introduce bad data. Therefore, an important component of the scraper is that before it executes and stores the data into the mongoDB, it checks if that title hotel is already in the database. If so, it exits out of the function and does not add any data. The second script is an insertion script that links to the web scraper and calls the function on select hotels. 13,593 reviews and ratings are captured this way, as well as the name, address, average price, and average review for the hotels. Finally, the query script is where the preprocessing and analysis happens. Here, we query the database and store the reviews and ratings into a data frame. We then combine this data frame with a pandas data frame. The query script cleans the data and converts it into a term document matrix. This is a data frame where each column is a word, each row is a review, and each cell is filled with a number for how many times that word appears in that review. We also use the Vader sentiment analyzer to add sentiment scores to each observation. These are compound score, negative score, positive score, and neutral score. This leaves us with over 59k columns. We therefore delete columns whose standard deviation is below .025. We also remove columns that are stop words. This now leaves us with 8,129 columns, a much more manageable amount of predictors. The similarity between ratings are compared and this leads to the conclusion that we should combine ratings 4 and 5 into the same group. The other groups however, are significantly different to one another. Next, outliers are removed. These are described in more detail above, but they are observations that fall outside the norm by an extreme amount. Finally, observations are sampled until all classes are balanced. Meaning, there are the same number of observations in each ratings group. At this point, the preprocessing is done and we pass these data frames into machine learning models. From here, the goal is to assess how well we can predict an observations rating. Random forest and extreme gradient boosting models are used.

Final Conclusion

The main take away from this project is the ability to scrape my own data and be able to run machine learning models on it. In the machine learning world, and data science in general, it is easy to grab a dataset from Kaggle. Although in these datasets cleaning and preprocessing is needed it is not the same thing as scraping your own data. This task elevates the sophistication of the project and allows flexibility in what to analyze. I am no longer bound by what datasets I can find online. I am confident in being able to scrape data from whatever website I want to and process it to the point where it can be used in a machine learning model. Working with text based data highlighted interesting points. When doing a project on home prices, an outlier is easy to understand, but when working with text the idea does not translate as easily. It takes comparisons and quantifying the ranges of sentiment score to understand their distribution and subsequently define an outlier. Extending basic statistical ideas to more sophisticated problems is essential to grow. The 3D visualizations highlighted another interesting issue. The problem is not perfectly defined. The difference between a rating of a 4 vs a 5 is often times not based on the review itself. It's the person typing it. Therefore, we cannot just look at the overall classification rate. We need to understand that misclassifying a 1 as a 2 is not the same as misclassifying a 1 as a 5. Either way, our classification rate of 65% is impressive for what it is. It can most likely be improved with deeper tuning and adding a deep learning model with Keras and TensorFlow.