Data Analytics Graduate Capstone: D214

Steven Oldenburg

Western Governors University

D214: Data Analytics Graduate Capstone

05/1/2024

Data Analytics Graduate Capstone: D214

Part A:

Project Name: Logistic Regression Analysis of Yelp Review Dataset

Project Topic: Utilizing Logistic Regression, Topic Modeling, and Frequency Analysis to Identify Common Themes in Restaurant 5-Star Yelp Reviews

Context:

In today's digital world, online reviews significantly influence customer decisions, particularly in the restaurant industry. This study examines Yelp as a vast database of restaurant reviews, offering both positive and negative impacts.

Research highlights:

Yelp's Discovery Power: Nielsen data (2017) shows that 55% of users discover restaurants on Yelp, often ordering takeout or delivery (47% were new customers). This emphasizes Yelp's ability to connect customers with new businesses.

Predicting Success: Building upon Hidayat et al. (2022), we explore using logistic regression to analyze review sentiment and potentially predict 5-star reviews based on keywords. This could empower restaurants to improve their online presence. This study produced a model with 87% accuracy and an F-1 Score of 81%.

Impacts:

This research explores how Yelp, with its mix of positive and negative reviews, can be a powerful tool for customers to discover new restaurants and businesses improving their online reputation.

Research Question:

Can a logistic regression model be constructed on a Yelp review dataset to predict the likelihood of a 5-star review based on text analysis?

Hypothesis:

Null Hypothesis: A logistic regression model constructed on the Yelp review dataset cannot achieve a statistically significant prediction of greater than 70% accuracy for 5-star reviews.

Alternative Hypothesis: A logistic regression model constructed on the Yelp review dataset can achieve statistically significant prediction of greater than 70% accuracy for 5-star reviews.

If possible, a final model is constructed with accuracy above 70% a second exploratory analysis will take place using word topics, themes, and frequencies for informational purposes only. The logical regression model will use these words to make predictions, if accuracy holds, the words can be extracted to give business insights and direction to future research.

Part B:

Data Description:

The raw data is 6.9 million rows of reviews of a variety of businesses across The United States. It is in the form of 5 JSON files, for this research, only 2 of the 4 will be used. This data is owned by Yelp but made freely available for research purposes (Yelp Open Dataset, n.d.). The raw dataset can be found here Yelp Dataset. The cleaned dataset can be found here.

Specific columns of interest are as follows:

Field	Type
Business_Id	Categorical
Categories	Categorical
Stars_x	Categorical
Text	Unstructured Data

Column Description:

Business_Id: A unique alphanumeric string tied to the business, an index.

Categories: A string that describes the type of business.

Stars_x: An integer that contains user star ratings 1-5.

Text: Unstructured text, reviews by users.

Below is one advantage of using this type of data and one disadvantage:

Advantage:

Time-Saving: This pre-made dataset saves significant time and effort. The data is already structured and cleaned by Yelp, ensuring consistency, and reducing the cleaning process.

Disadvantage:

Limited Control: The Dataset is massive, but still lacks enough information to dial in on specific restaurants or even restaurant areas as the number of businesses across The United States dwarfs the dataset. There are only so many 5-star restaurant reviews in the dataset which ultimately will limit the size of the trainable data.

Overcoming Challenges:

The dataset is large, combining all 5 JSON files is over 8 gigabytes of data. Just importing the data on my personal computer with a state-of-the-art processor takes over 4 minutes. To overcome this, I outlined what specific data I needed from each file, finding that I could import 2 JSON files instead of 5 and cut the processing time for all tasks in half.

Part C:

Data Extraction and Preparation:

The first step in the process of data extraction is to import the two JSON files from the Yelp dataset and merge them in the business_id column. The business_id column is the unique identifier to match these reviews with their businesses which will be used to group them.

```
business = pd.read_json("yelp_academic_dataset_business.json",lines=True)
review = pd.read_json("yelp_academic_dataset_review.json",lines=True)
merged_df = pd.merge(business, review, on='business_id')
print(merged_df.columns)
print(merged_df.head())

✓ 2m 2.7s
```

Next, the Pandas data frame is checked for correct data types in our 4 specific columns. It's also checked for null values, total shape, and data sparsity and the null values in the categories column are dropped.

```
# Check data structure and missing values
print(merged_df.dtypes)
print(merged_df.isnull().sum())
print(merged_df.shape)

total_cells = merged_df.size


missing_values = merged_df.isnull().sum().sum()

# Calculate the data sparsity (percentage of missing values)
data_sparsity = (missing_values / total_cells) * 100

# Print the data sparsity as a percentage
print(f"Overall data sparsity: {data_sparsity:.2f}%")

#Drop Missing Values
merged_df.dropna(subset=['categories'], inplace=True)
```

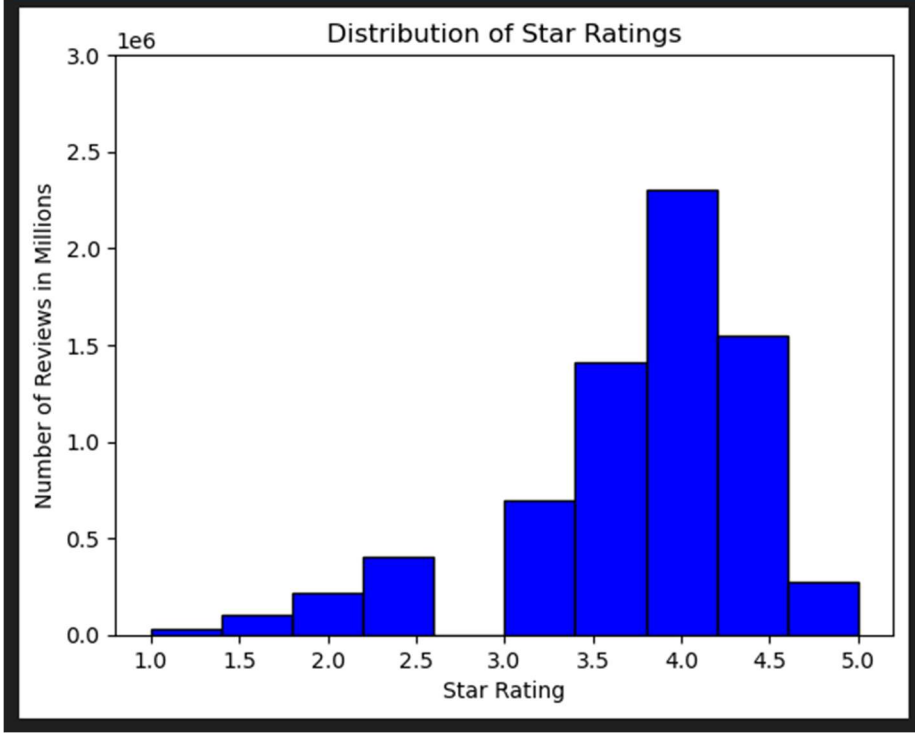
These queries returned:

- The columns 4 columns needed are the correct data types.
- There were 689 missing values in the categories column.
- The total rows are 6,990,280.
- Data sparsity is .38%.

The next step is to check for the normality of the star ratings.

```
#Plot histogram of star ratings to check for normality.
plt.hist(merged_df['stars_x'], color='blue', edgecolor='black')

plt.title("Distribution of Star Ratings")

plt.xlabel("Star Rating")

plt.ylabel("Number of Reviews in Millions")
plt.ylim(0, 3000000)

plt.show()
```

✓ 0.2s



Star ratings have a right-skewed distribution.

```
#Shapiro-Wilk test for normality
stat, p_value = stats.shapiro(merged_df["stars_x"])

print("Shapiro-Wilk test statistic:", stat)
print("p-value:", p_value)
```

✓ 0.1s

```
Shapiro-Wilk test statistic: 0.9034406830238736
p-value: 1.347272637198699e-163
```

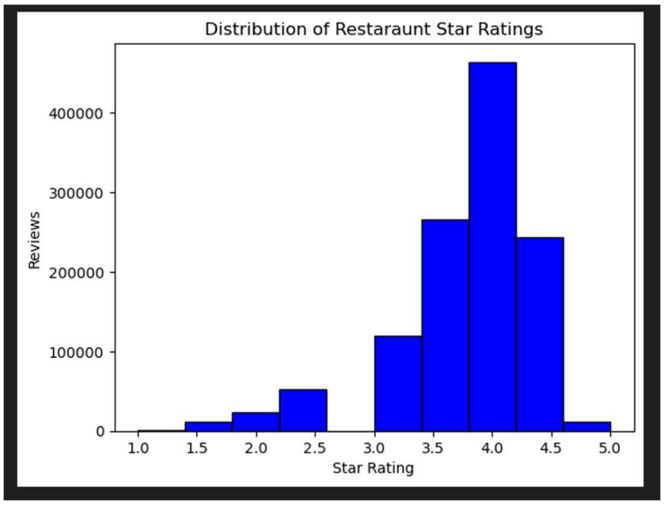
A Shapiro-Wilk test statistic and highly significant p-value suggest it is non-normal.

```
#Drop Extra columns
merged_df.drop(['date', 'name', 'review_id','address', 'attributes', 'hours', 'city','is_open', 'u

#this groups businesses by category ie. restaurant, shopping, etc
merged_df['categories'] = merged_df['categories'].apply(lambda x: x.split(',')[0] if x else 'NA')

merged_df['categories'] = merged_df['categories'].fillna('NA')
merged_df['categories'] = merged_df['categories'].str.strip()
#Filter for Restaurants
merged_df = merged_df[merged_df['categories'] == 'Restaurants']

print (merged_df.head(2))
```

✓ 3.6s

```
              business_id  stars_x   categories  \
46  MTSW4McQd7CbVtyjqoe9mw      4.0  Restaurants
47  MTSW4McQd7CbVtyjqoe9mw      4.0  Restaurants

                                                text
46  This is nice little Chinese bakery in the hear...
47  This is the bakery I usually go to in Chinatow...
```

Dropped the columns that are not needed, filter for restaurants by splitting the categories column by using the comma character. The data contains a business category grouping, comma, and then further explanation of the services of the business.



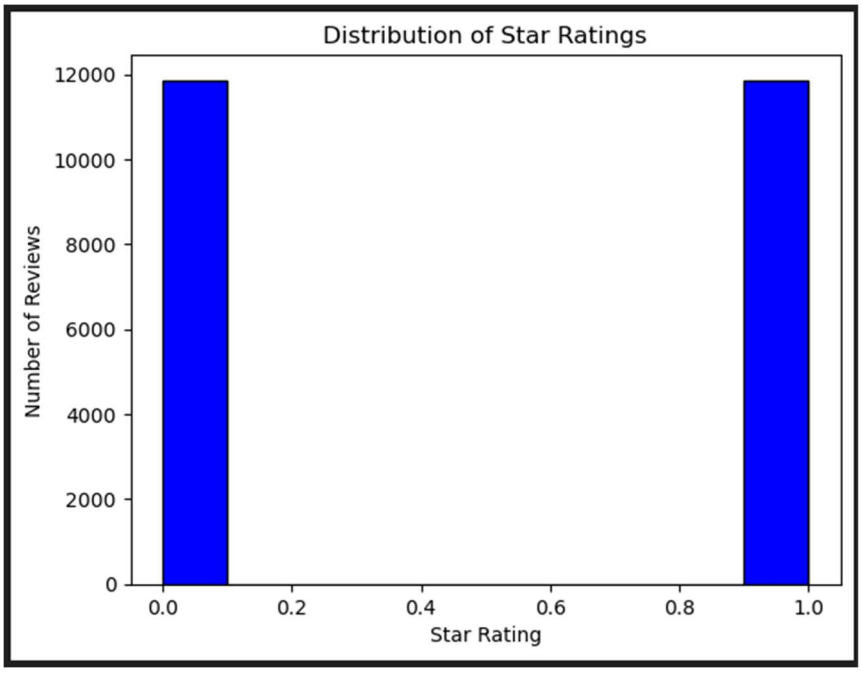
Rechecking for normalcy with only the restaurants shows a slight difference, fewer 5-star reviews, and a larger portion of 4-star reviews.

```
#Seperate 5 star ratings from the rest
stars = merged_df[merged_df['stars_x'] == 5]

filtered_reviews = merged_df[merged_df['stars_x'] < 5]


sample = [1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5]
big_list = pd.DataFrame()

#sample 1 star reviews seperately
sample_reviews = filtered_reviews[filtered_reviews['stars_x'] == 1.0].sample(1135)
big_list = pd.concat([big_list, sample_reviews], ignore_index=True)
big_list.shape

#sample the rest of the ratings
for i in sample:
    sample_reviews = filtered_reviews[filtered_reviews['stars_x'] == i].sample(1532)
    big_list = pd.concat([big_list, sample_reviews], ignore_index=True)

#add reviews to keep the data balanced
sample_reviews = filtered_reviews[filtered_reviews['stars_x'] == 1.5].sample(6)
big_list = pd.concat([big_list, sample_reviews], ignore_index=True)

#build a new dataframe with the sampled reviews and transform the ratings to 1 and 0's
sample_df = pd.concat([big_list, stars])
print(sample_df.shape)

sample_df["stars_x"] = transform_ratings(sample_df["stars_x"])
```

✓ 0.3s

```
(23730, 4)
```

The 5-star reviews are separated and an even sample balancing reviews against reviews that are not 5 stars is taken. The samples within each star grouping were also balanced to make sure bias was not introduced into the model. If the model is given the majority of one type of data it will predict those outcomes with higher accuracy which can artificially inflate the overall accuracy rating of the model, while not realistically accomplishing the model's goal. For example, a model that predicts ratings less than 5 stars while our goal is to predict 5-star ratings.




```
#Export dataset to csv
sample_df.to_csv ('yelp_sample_df.csv', index = None, header=True)
```
✓ 0.1s

This is the final sample of cleaned and balanced data, it's exported for continued use.

```
#standarize text, remove punctuation, stopwords, lemmatize and de-emoji
def transform_ratings(ratings):
    transformed_ratings = [1 if rating == 5 else 0 for rating in ratings]
    return transformed_ratings

def clean_text(text):
    text = text.lower()

    text = re.sub(r'[^\w\s]', '', text)

    stop_words = stopwords.words('english')
    text = [word for word in text.split() if word not in stop_words]

    lemmatizer = WordNetLemmatizer()
    text = [lemmatizer.lemmatize(word) for word in text]

    return " ".join(text)

def replace_emojis(text):
    if emoji:
        return emoji.demojize(text, delimiters=("", ""))
    return text
```

Create 3 functions that change the text to lowercase, remove punctuation, stop words, lemmatize, and de-emoji the text.

```
#clean the text and tokenize it
cleaned_text = []
for text in sample_df["text"]:
  cleaned_text.append(replace_emojis(clean_text(text)))

print(cleaned_text[:10])
text_data = cleaned_text

star_ratings = sample_df["stars_x"]

tokenized_data = []
for text in text_data:
  tokens = word_tokenize(text)
  tokenized_data.append(tokens)

print(tokenized_data[:10])
```

✓ 12.6s

```
'place horrible left gave friend someone else pizza pizza cold burnt explained worker
['place', 'horrible', 'left', 'gave', 'friend', 'someone', 'else', 'pizza', 'pizza',
```

Apply the functions cleaning the text and tokenizing the data.

```
#Create a dataframe with the 5 star predictions for word analysis.
predictions = pd.DataFrame(columns=['text', 'prediction'])
predictions['text'] = sample_df['text']
predictions['prediction'] = sample_df['stars_x']
predictions = predictions[predictions['prediction'] == 1]

#clean the text
cleaned_text = []
for text in predictions["text"]:
  cleaned_text.append(replace_emojis(clean_text(text)))

predictions["text"] = cleaned_text

print(predictions.head())
```

✓ 4.3s

	text	prediction
2269	amazing food great service luigi owner greeted...	1
2270	chose spot wife final date night since shes 8 ...	1
2271	excellent excellent excellent italian food bes...	1
2272	love placetucked elbow lower merion wonderful ...	1
2273	absolutely phenomenal takeout couple week ago ...	1

The final step in data preparation is to create a data frame consisting of only 5-star reviews for word and topic analysis.

Tools and packages used for the extraction process, including advantages and disadvantages:

1. Visual Studio Code – This is a GUI interface in a notebook environment for Python. This was chosen because it has much more support in the form of third-party extensions and fewer limitations than working directly in Jupyter Notebook out of the box.
2. Python – Python was chosen because of its ease of use, especially for cleaning and wrangling data.
    a. Pandas – Pandas was chosen because it easily renders data in 2-dimensional space, like an Excel spreadsheet. It has a built-in function for importing JSON files. The dataset was already in this format making it very easy to manipulate.
    b. Matplotlib – Matplotlib was chosen to create histograms of the data distribution of star ratings. It was selected because it is easy to use, especially for basic plotting. It can struggle with much more robust plotting, which is where other tools like Seaborn shine, however, it was not needed in this project.
    c. Scipy.stats – Scipy stats were used to run the Shapiro-Wilks Test. It was chosen to double-check the visuals of matplotlib and how easy the package is to use. It only needs the column to apply the test and it does the rest. One disadvantage is it becomes less reliable on larger datasets.

Part D:

Analysis:

The first step in the analysis is to see how words, unique words, and average unique words per review.

```
total_words = 0
unique_words = set()

for text in sample_df['text']:
    words = text.split()
    total_words += len(words)


    unique_words.update(words)
average_unique_words = len(unique_words) / len(sample_df['text'])

print("Total Words:", total_words)
print("Unique Words:", len(unique_words))
print("Average Unique Words per Review:", average_unique_words)
```

✓ 0.1s

```
Total Words: 2127518
Unique Words: 88401
Average Unique Words per Review: 3.725284450063211
```

The overall word analysis shows that the dataset contains a robust vocabulary of reviews and that reviewers will often use different words in their reviews. On average, each reviewer adds 3 to 4 new words per review, which may seem small but there are over 20,000 reviews which demonstrates the diversity of the reviewers.

```
#Training split

X_train, X_test, y_train, y_test = train_test_split(tokenized_data, star_ratings, test_size=0.2, random_state=99)


X_train_joined = [" ".join(text) for text in X_train]
X_test_joined = [" ".join(text) for text in X_test]

# Fit the TF-IDF vectorizer
vectorizer = TfidfVectorizer()
text_features_train = vectorizer.fit_transform(X_train_joined)

text_features_test = vectorizer.transform(X_test_joined)

#Build a logistic regression model
model = LogisticRegression(max_iter=10000)
model.fit(text_features_train, y_train)

predicted_probabilities = model.predict_proba(text_features_test)[:, 1]

#Classify the models threshold
threshold = 0.6
predicted_ratings = [1 if prob >= threshold else 0 for prob in predicted_probabilities]
```

✓ 0.7s

Next, the logistic regression model is built with a threshold of 60%, this means that any prediction above a 60% chance will be classified as a 5-star review.

```python
#Confusion Matrix, accuracy and classification report
cm = confusion_matrix(y_test, predicted_ratings)

print("Confusion Matrix:\n", cm)

accuracy = accuracy_score(y_test, predicted_ratings)
print("Classification Accuracy:", accuracy)

print(classification_report(predicted_ratings, y_test))
```

✓ 0.0s

```
Confusion Matrix:
 [[2103  276]
 [ 533 1834]]
Classification Accuracy: 0.8295406658238517
              precision    recall  f1-score   support

           0       0.88      0.80      0.84      2636
           1       0.77      0.87      0.82      2110

    accuracy                           0.83      4746
   macro avg       0.83      0.83      0.83      4746
weighted avg       0.84      0.83      0.83      4746
```

Our initial model is successful with the chosen metric of greater than 70% accuracy, however, it has quite a lot of false positives. The model incorrectly labeled 553 as false negatives, which shows there is some room for improvement particularly with positives being classified as negatives.

```python
#Iterate through different class weights
for i in range(30, 40):
    class_weight = {0: 1, 1: i * .1}

    model = LogisticRegression(class_weight=class_weight, max_iter=10000)
    model.fit(text_features_train, y_train)


    predicted_probabilities = model.predict_proba(text_features_test)[:, 1]

    threshold = 0.71
    predicted_ratings = [1 if prob >= threshold else 0 for prob in predicted_probabilities]

    print(i *.1)
    cm = confusion_matrix(y_test, predicted_ratings)

    print("Confusion Matrix:\n", cm)

    accuracy = accuracy_score(y_test, predicted_ratings)

    print("Classification Accuracy:", accuracy)

    print(classification_report(predicted_ratings, y_test))
```

The threshold for predictions was increased and an iteration to find the best weight to increase the accuracy and reliability of the 5-star predictions. The following is the output:

```
3.0
Confusion Matrix:
 [[1977  402]
 [ 355 2012]]
Classification Accuracy: 0.8404972608512431
              precision    recall  f1-score   support

           0       0.83      0.85      0.84      2332
           1       0.85      0.83      0.84      2414

    accuracy                           0.84      4746
   macro avg       0.84      0.84      0.84      4746
weighted avg       0.84      0.84      0.84      4746

3.1
Confusion Matrix:
 [[1972  407]
 [ 344 2023]]
Classification Accuracy: 0.8417614833544037
              precision    recall  f1-score   support

           0       0.83      0.85      0.84      2316
           1       0.85      0.83      0.84      2430

    accuracy                           0.84      4746
   macro avg       0.84      0.84      0.84      4746
weighted avg       0.84      0.84      0.84      4746

3.2
Confusion Matrix:
 [[1966  413]
 [ 336 2031]]
Classification Accuracy: 0.8421828908554573
              precision    recall  f1-score   support

           0       0.83      0.85      0.84      2302
           1       0.86      0.83      0.84      2444

    accuracy                           0.84      4746
   macro avg       0.84      0.84      0.84      4746
weighted avg       0.84      0.84      0.84      4746

3.3000000000000003
Confusion Matrix:
 [[1961  418]
 [ 320 2047]]
Classification Accuracy: 0.8445006321112516
              precision    recall  f1-score   support
```

```
           0       0.82      0.86      0.84      2281
           1       0.86      0.83      0.85      2465

    accuracy                           0.84      4746
   macro avg       0.84      0.85      0.84      4746
weighted avg       0.85      0.84      0.84      4746

3.4000000000000004
Confusion Matrix:
 [[1951  428]
 [ 310 2057]]
Classification Accuracy: 0.8445006321112516
              precision    recall  f1-score   support

           0       0.82      0.86      0.84      2261
           1       0.87      0.83      0.85      2485

    accuracy                           0.84      4746
   macro avg       0.84      0.85      0.84      4746
weighted avg       0.85      0.84      0.84      4746

3.5
Confusion Matrix:
 [[1946  433]
 [ 302 2065]]
Classification Accuracy: 0.8451327433628318
              precision    recall  f1-score   support

           0       0.82      0.87      0.84      2248
           1       0.87      0.83      0.85      2498

    accuracy                           0.85      4746
   macro avg       0.85      0.85      0.85      4746
weighted avg       0.85      0.85      0.85      4746

3.6
Confusion Matrix:
 [[1934  445]
 [ 295 2072]]
Classification Accuracy: 0.844079224610198
              precision    recall  f1-score   support

           0       0.81      0.87      0.84      2229
           1       0.88      0.82      0.85      2517

    accuracy                           0.84      4746
   macro avg       0.84      0.85      0.84      4746
weighted avg       0.85      0.84      0.84      4746
```

```
3.7
Confusion Matrix:
 [[1922  457]
 [ 287 2080]]
Classification Accuracy: 0.843236409608091
              precision    recall  f1-score   support

           0       0.81      0.87      0.84      2209
           1       0.88      0.82      0.85      2537

    accuracy                           0.84      4746
   macro avg       0.84      0.84      0.84      4746
weighted avg       0.85      0.84      0.84      4746

3.8000000000000003
Confusion Matrix:
 [[1917  462]
 [ 285 2082]]
Classification Accuracy: 0.8426042983565107
              precision    recall  f1-score   support

           0       0.81      0.87      0.84      2202
           1       0.88      0.82      0.85      2544

    accuracy                           0.84      4746
   macro avg       0.84      0.84      0.84      4746
weighted avg       0.85      0.84      0.84      4746

3.9000000000000004
Confusion Matrix:
 [[1909  470]
 [ 276 2091]]
Classification Accuracy: 0.8428150021070375
              precision    recall  f1-score   support

           0       0.80      0.87      0.84      2185
           1       0.88      0.82      0.85      2561

    accuracy                           0.84      4746
   macro avg       0.84      0.85      0.84      4746
weighted avg       0.85      0.84      0.84      4746
```

All these models are very similar, however, the highest accuracy which is the metric we selected is when 5-star predictions are weighted at 3.5 with an accuracy of 84.5%. The model is slightly

worse at predicting reviews that are not 5-star at 82% but succeeds at predicting 5-star reviews 87% of the time. This model shows a slight improvement in all categories except recall. The f-1 score which is a balance between precision and recall is slightly higher at 85% for predicting 5-star reviews over the initial 82%.

```
#Final model
class_weight = {0: 1, 1: 3.5}

model = LogisticRegression(class_weight=class_weight, max_iter=10000)
model.fit(text_features_train, y_train)

predicted_probabilities = model.predict_proba(text_features_test)[:, 1]

threshold = 0.71
predicted_ratings = [1 if prob >= threshold else 0 for prob in predicted_probabilities]

cm = confusion_matrix(y_test, predicted_ratings)

print("Confusion Matrix:\n", cm)

accuracy = accuracy_score(y_test, predicted_ratings)

print("Classification Accuracy:", accuracy)

print(classification_report(predicted_ratings, y_test))
```

✓ 0.3s

```
Confusion Matrix:
 [[1946  433]
 [ 302 2065]]
Classification Accuracy: 0.8451327433628318
              precision    recall  f1-score   support

           0       0.82      0.87      0.84      2248
           1       0.87      0.83      0.85      2498

    accuracy                           0.85      4746
   macro avg       0.85      0.85      0.85      4746
weighted avg       0.85      0.85      0.85      4746
```

Final logistic regression model: 5-star reviews weighted at 3.5, with a 71% threshold.

The final model is statistically sound, as such it gives merit to extracting topic and word information from the dataset. If the model can statistically predict 5-star reviews using relatively the same data, we will extract insights from the words themselves for informational purposes.

```
#Vectorize word counts
vectorizer = CountVectorizer()
document_term_matrix = vectorizer.fit_transform(predictions.text)

# Train the LDA model
num_topics = 100
lda_model = LatentDirichletAllocation(n_components=num_topics, random_state=0)
lda_model.fit(document_term_matrix)

topic_words = lda_model.components_.argsort(axis=1)[:, ::-1]

document_topic = lda_model.transform(document_term_matrix)


✓ 32.3s
```

A word vector count is created to count the appearance of each word in each review to build topics. 100 topics are created to ensure the diversity of the 11,000+ reviews.

```
vocab = vectorizer.get_feature_names_out()
num_top_words = 20  # Number of top words to display per topic

#Use the LDA model to find the top words in each topic
for topic_idx, words in enumerate(topic_words[:20]):
    topic_probs = lda_model.components_[topic_idx]
    sorted_word_indices = topic_probs.argsort()[::-1][:num_top_words]
    print(f"Topic {topic_idx+1}:", [vocab[i] for i in sorted_word_indices])
```

✓ 0.0s

```
Topic 1: ['really', 'place', 'food', 'another', 'way', 'like', 'lunch', 'star', 'sandwich', 'know', 'kyle', 'eat',
Topic 2: ['good', 'chicken', 'food', 'order', 'got', 'delicious', 'really', 'came', 'also', 'dip', 'meal', 'salad'
Topic 3: ['course', 'pairing', 'experience', 'drink', 'menu', 'meal', 'cupcake', 'unique', 'since', 'reservation',
Topic 4: ['great', 'smoothy', 'lot', 'love', 'work', 'grab', 'dog', 'also', 'food', 'friendly', 'juice', 'make', '
Topic 5: ['cream', 'cake', 'ice', 'chocolate', 'bakery', 'dessert', 'delicious', 'cooky', 'good', 'also', 'butter'
Topic 6: ['middle', 'great', 'eastern', 'delicious', 'salad', 'lasagna', 'food', 'crepe', 'try', 'place', 'cheese'
Topic 7: ['food', 'good', 'like', 'ordered', 'really', 'place', 'also', 'time', 'well', 'try', 'sauce', 'got', 'sa
Topic 8: ['waffle', 'empanadas', 'empanada', 'order', 'try', 'go', 'place', 'sweet', 'back', 'strawberry', 'good',
Topic 9: ['food', 'good', 'def', 'time', 'place', 'back', 'amazing', 'group', 'delicious', 'kurdish', 'came', 'wou
Topic 10: ['mi', 'quang', 'vietnamese', 'restaurant', 'back', 'banh', 'bun', 'place', 'food', 'get', 'dish', 'pork
Topic 11: ['drop', 'like', 'great', 'good', 'best', 'relaxed', 'restaurant', 'collard', 'mesa', 'another', 'super'
Topic 12: ['waiting', 'food', 'lot', 'may', 'two', 'brewery', 'get', 'ridge', 'want', 'anniversary', 'guide', 'gre
Topic 13: ['food', 'fair', 'yum', 'never', 'price', 'place', 'disappoints', 'delicious', 'stick', 'establishment',
Topic 14: ['crepe', 'good', 'meal', 'could', 'got', 'small', 'well', 'friend', 'like', 'group', 'chef', 'butter',
Topic 15: ['doughnut', 'place', 'home', 'one', 'best', 'good', 'go', 'love', 'even', 'like', 'life', 'tried', 'gem
Topic 16: ['cuban', 'flan', 'sandwich', 'good', 'cheesesteak', 'sauce', 'guava', 'cheese', 'ordered', 'really', 't
Topic 17: ['place', 'cheese', 'simply', 'marvelous', 'try', 'steph', 'food', 'great', 'go', 'sauce', 'corn', 'star
Topic 18: ['food', 'great', 'delicious', 'priced', 'service', 'definitely', 'good', 'friendly', 'place', 'back', '
Topic 19: ['coffee', 'tea', 'shop', 'drink', 'iced', 'good', 'great', 'place', 'little', 'latte', 'also', 'cup', '
Topic 20: ['best', 'amazing', 'go', 'flavor', 'make', 'daddy', 'place', 'kombucha', 'love', 'good', 'ever', 'kind'
```

Here is an example of 20 different topics created by the LDA model. It creates this list by using the most frequent or probable words to appear in the topic.

```
Document 1533: Topic 9: place legit filipino food friend came get something help hangoverthe chef made pork dinuguan
Document 3483: Topic 9: youve pho awesome youre adventurous youve bun bo hue reach next level vietnamese noodle soup
Document 888: Topic 9: ok start driving make sure paying attention pas step place transformed deli brooklyn hot smell
Document 10229: Topic 9: holy moly place truth know small drivethrough lattesfood joke excited try one smoothy next t
Document 3510: Topic 9: authentic viet food tucked unexpected neighborhood place kept showing recommendation decided
Document 5985: Topic 9: best plantbased meal restaurant considered popup khyber pas pub one would expect gourmet vega
Document 3523: Topic 9: restaurant one favorite vietnamese restaurant philadelphia area mom friend knew restaurant re
Document 9973: Topic 9: first trip el antojo poblano ordered huarache chicharron pretty good thought chicharron lot f
Document 5222: Topic 9: completed herbal detox cleanse hopefully heal m symptom knowledgeable vegan diet hope maintai
Document 3468: Topic 9: searching vietnamese restaurant way philadelphia yelp suggested place saw lot good review wen
```

Upon further review, topic 9 contains information mainly about customers who wrote reviews about Vietnamese food. While the data is from an array of customers across The United States due to the limitations of the dataset, if applied to a specific business location it would highlight specific topics for a particular business. For example, sentiment on thin crust or thick crust pizza at a pizzeria.

```
Top 10 topics (by document prevalence):
Topic 38 (Documents: 1115, Proportion: 9.4%)
Topic 70 (Documents: 961, Proportion: 8.1%)
Topic 74 (Documents: 378, Proportion: 3.19%)
Topic 55 (Documents: 359, Proportion: 3.03%)
Topic 84 (Documents: 309, Proportion: 2.6%)
Topic 19 (Documents: 286, Proportion: 2.41%)
Topic 18 (Documents: 283, Proportion: 2.39%)
Topic 61 (Documents: 278, Proportion: 2.34%)
Topic 87 (Documents: 266, Proportion: 2.24%)
Topic 66 (Documents: 239, Proportion: 2.01%)
```

Implementing code to calculate the prevalence of each of the top topics, reveals two major groups that account for 17.5% of the model's topics which were identified.

```
vocab = vectorizer.get_feature_names_out()
num_top_words = 20

for topic_idx in [37, 69, 73]:  # Access top topics
    topic_probs = lda_model.components_[topic_idx]
    sorted_word_indices = topic_probs.argsort()[::-1][:num_top_words]
    print(f"Topic {topic_idx+1}:", [vocab[i] for i in sorted_word_indices])
```

✓ 0.0s

```
Topic 38: ['food', 'recommend', 'place', 'great', 'amazing', 'service', 'highly', 'friendly', 'best', 'delicious', '
Topic 70: ['great', 'food', 'friendly', 'place', 'fresh', 'price', 'service', 'good', 'staff', 'lunch', 'delicious',
Topic 74: ['breakfast', 'sandwich', 'croissant', 'coffee', 'pastry', 'egg', 'cafe', 'delicious', 'cheese', 'place',
```

Analyzing topics 38,70 and 74 which are the top 3 topics reveals that they are closely related to in subject with subtle differences. Topic 38 focuses on emotion while topic 70 has a higher focus on product specifics such as price, fresh, and lunch. Topic 74 focuses especially on breakfast, sandwiches, and other breakfast products. This shows that based on our model, 5-star reviews in the Yelp dataset focus mainly on emotional ambiance, secondly service, and lastly the product itself. It's also notable that the topic focusing on breakfast almost exclusively mentions the products purchased.

```
#Get the word counts
word_counts = Counter()
for text in predictions.text:
    words = text.split()
    word_counts.update(words)

most_common = word_counts.most_common(10)
print("Most frequent words:", most_common)
```

✓ 0.0s

```
Most frequent words: [('food', 7216), ('place', 6161), ('great', 5386), ('good', 4720),
```

Next, we count the instances or frequencies the words appeared in 5-star reviews.

```
#Top 25 most frequent words across topics
word_list = ['like', 'one', 'back', 'get', 'really', 'go', 'also', 'got'

filtered_counts = Counter({word: count for word, count in word_counts.it

top_n_words = filtered_counts.most_common(25)
print("\nTop 25 Most Frequent Words Across Topics:")
for word, count in top_n_words:
    print(f"- {word}: {count}")
```

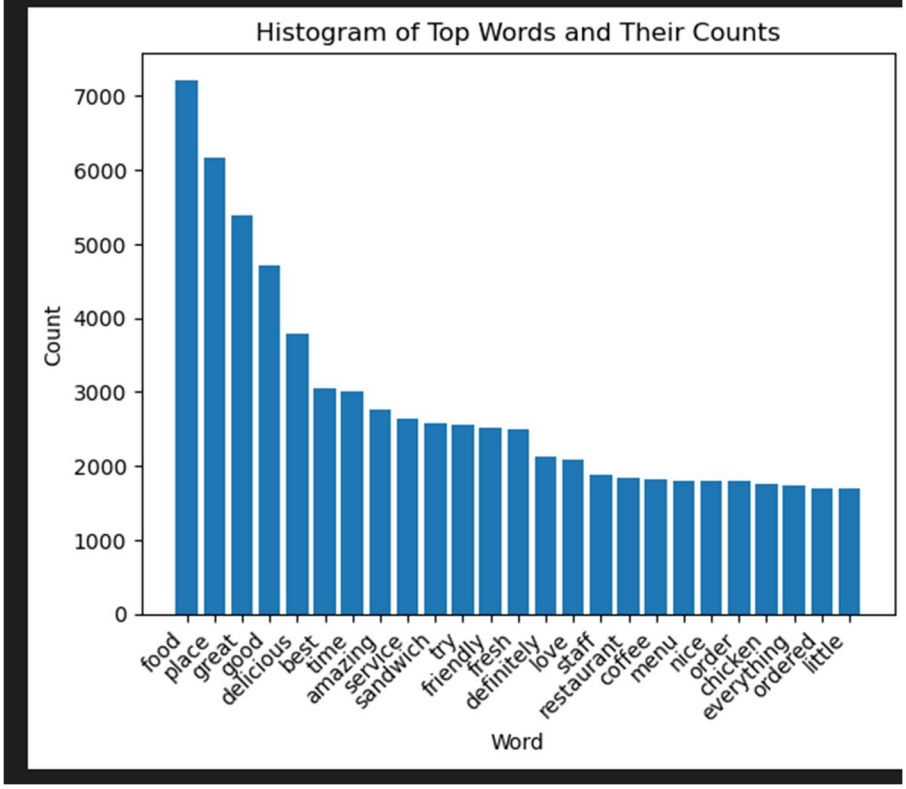
✓ 0.0s

```
op 25 Most Frequent Words Across Topics:
 food: 7216
 place: 6161
 great: 5386
 good: 4720
 delicious: 3787
 best: 3058
 time: 3001
 amazing: 2759
 service: 2636
 sandwich: 2576
 try: 2553
 friendly: 2510
 fresh: 2493
 definitely: 2132
 love: 2093
 staff: 1889
 restaurant: 1848
 coffee: 1821
 menu: 1805
 nice: 1801
 order: 1791
 chicken: 1758
 everything: 1734
 ordered: 1703
 little: 1696
```

Filtering out words that add little meaning such as like, one, back, etc… We find the top 25 most common words among 5-star review raters.

```
# Create the histogram
words, counts = zip(*top_n_words)

plt.bar(words, counts)

plt.xlabel("Word")
plt.ylabel("Count")
plt.title("Histogram of Top Words and Their Counts")

plt.xticks(rotation=45, ha='right')

plt.show()
```

✓ 0.0s



It's not surprising that food is located towards the top when analyzing restaurant data, but the place is mentioned almost as frequently as food, suggesting that the environment is a possible highly ranked reason for 5-star reviews as well. Emotional responses are common across the data, such as good, great, amazing, and time. Notably, the word try is also mentioned suggesting a large portion of 5-star reviews coming from a population who visits the restaurant for the first time.

Justification of Techniques:

TF-IDF vector with logical regression:

Justification: Logistic regression is well-suited for analyzing text data and predicting a binary categorical outcome. TF-IDF helps weight terms based on their informativeness to adjust for more than just word counts and creates weights and combinations of words appearing together.

Advantage:

Classification: Logistic regression excels at classifying data points. In this case, it can categorize reviews as 5-star or not 5-star.

Disadvantage:

Non-linear Relationships: Logistic regression assumes a linear relationship between features and the outcome variable (log odds in this case). This will not capture all the complexities of customer sentiment and include human behaviors in reviews.

Vector Count and LDA with Topic Analysis and Word Count

Justification: LDA is a powerful technique for exploring thematic structures within text data.

Advantages:

Uncovering Hidden Themes: LDA excels at identifying topics that capture recurring themes within the reviews.

Understanding Vocabulary: Word counts provide a basic understanding of the vocabulary used in the reviews.

Disadvantage:

Interpretability: The topics generated by LDA can be challenging to interpret directly. For instance, in the dataset, the word food is mentioned the most. It could be "The food was amazing!" or simply "The food came..."

Part E:

Research Question: Can a Logical Regression model be constructed on the market research dataset?

Findings:

A logical regression model can be constructed due to:

- Statistical tests led to the rejection of the null hypothesis, implying a significant relationship between the independent variables and the dependent variable.
- The model achieved a high accuracy score of 84.5%, indicating its effectiveness in predicting the outcome variable.
- The model maintained reasonable recall rates, suggesting it doesn't miss a significant number of positive cases.
- The confusion matrix analysis further supports the model's validity by demonstrating a balance between false negatives and false positives.
- Word frequencies and word themes of Yelp data suggest previous research on Yelp reviewers holds. Phrases such as try or trying to suggest a large portion of reviews are from first-time visitors and research suggests there is a competitive population of reviewers who actively seek status through reviews (Yu & Margolin, 2021).
- Further research would also be needed to confirm if this segment of the Yelp population is the key driver in emotional lexicon verbiage in reviews. The verbiage found in word

frequencies is very similar to sales strategies, as appealing to emotion is a common sales tactic (Erevelles & Fukawa, 2013).

Implications:

- A logical regression model is a valid model for predicting Yelp star ratings when combined with a TF-IDF.
- Word frequencies are used as the basis for this type of model and show that actionable data can potentially be extracted from the dataset.

Limitations:

- Data: The data is a small dataset of the entire Yelp dataset and may not represent the whole.
- Word Frequencies: Word frequencies reviewed in this manner can only be used in a general sense and do not directly prove anything except that the information was worth mentioning in a review to a customer. For instance, the words amazing, chicken, and service cannot suggest either the service or chicken is amazing because the topic never says any of the words are connected.

Recommendation:

Obtain a larger, much more specialized dataset. For instance, a dataset relatively the same size with pizza locations in one area would improve the accuracy of the model. The model right now analyzes not only diverse customers across The United States but also diverse businesses and products, potentially confusing the model.

Direction For Further Research:

Direction 1: Deep Learning for Aspect-Based Sentiment Analysis (ABSA)

The logistic regression model predicts overall star ratings, customer reviews often contain opinions on specific aspects like food, service, or ambiance. Aspect-based sentiment analysis (ABSA) can identify aspects and analyze the sentiment expressed towards them.

Benefits:

- Provides a focused understanding of customer sentiment towards various aspects of the business.
- Allows for targeted improvements based on aspects with negative sentiment.
- Can be used to build recommendations suggesting items or services based on positive aspects mentioned in reviews or potential opportunities from negative aspects.

Limitations:

- Takes much more processing power and may not be suitable for all businesses because of the computational demand.

Direction 2: Combining N-grams, TF-IDF Vectors, and Logistic Regression for Customer Review Analysis. Logistic regression models with TF-IDF features are valuable for predicting star ratings, they primarily focus on individual words. By combining n-grams into the model it's possible to expand the model's ability to predict while also being able to extract the n-grams for future reference.

Benefits:

- Improved Feature Representation & Deeper Customer Insights: Extracting keywords from n-grams allows you to analyze the importance of specific words across different phrases. This can reveal broader themes or aspects influencing customer sentiment.
- Interpretable Model: Logistic regression remains the core model, allowing you to interpret the coefficients and understand the relationship between features and star ratings.

Limitations:

- Complexity: Engineering steps like keyword extraction and selection can add complexity to the process. Adding N-grams to the process may only add to the complexity of the process.

Summary:

This research demonstrates how a logistic regression model can be successful in predicting Yelp star ratings at an accuracy rating of 84.5% which is above the 70% threshold. It also suggests that by leveraging a proven track record of the logistic regression model, data exploration can be performed on that same data to extract key features. Specifically, our topic model analysis revealed that two topics, which contained similar words consisted of 17.5% of the entirety of the 5-star reviews in the dataset.

References

Erevelles, S., & Fukawa, N. (2013). The Role of Affect in Personal Selling and Sales Management. Journal of Personal Selling & Sales Management, 33(1), 7–24. https://doi.org/10.2753/PSS0885-3134330102

Yu, C., & Margolin, D. (2021). The disproportion of crowd wisdom: The impact of status-seeking on Yelp reviews. PLoS ONE, 16(6), 1–17. https://doi.org/10.1371/journal.pone.0252157