Phase 5 Project: Sentiment Analysis of Yelp Restaurant Reviews.

Submitted by: Rashmi Chauhan (Data Science Live Cohort Oct June-Oct 2024)

Introduction

The primary goal of this sentiment analysis project is to explore and understand the sentiment expressed in Yelp restaurant reviews, focusing on identifying key themes and sentiments that customers associate with their dining experiences. This analysis aims to provide actionable insights for restaurant owners and managers, enabling them to improve customer satisfaction and operational efficiency.

Project Walkthrough

The Analysis consists of 3 parts:

1. Exploratory analysis on the overall review trends across time.
2. Latent Dirichlet Allocation (LDA) model to determine key topics among reviews.
3. Calculate the polarity score and assign sentiment based on Vader NLP technique.

Dataset

The dataset for this project is taken form Yelp's open dataset. It has 6,990,280 reviews about over 150,346 businesses.

Each row in the reviews dataframe consists of reviews data: business_id of the restaurant reviewed, star rating granted, number of [useful, funny, cool] votes, review text, and date.

Each row in the business dataframe consists of business data: business_id, business name, state, overall star rating, number of reviews, categories, and business status (open or closed).

The data is further filtered to show analysis on the restaurant business in Florida state, and the time period is from 2020 to 2024.

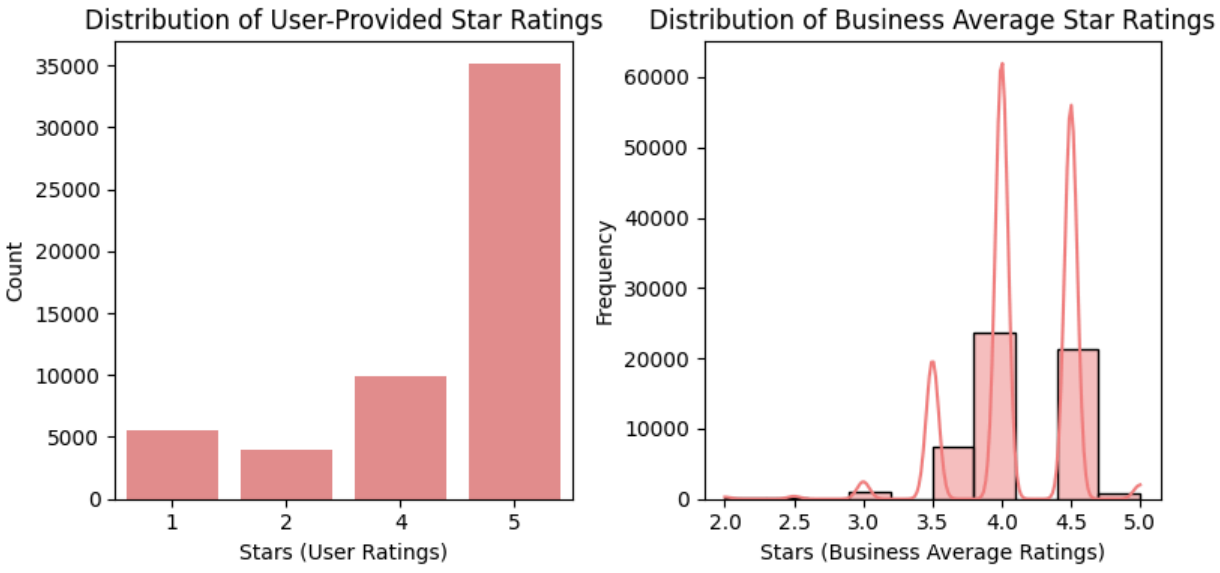
Exploratory Data analysis

In [1]:
```
import pandas as pd
import warnings

# Ignore all warnings
warnings.filterwarnings("ignore")
# Ignore specific warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)
# Replace DeprecationWarning with other warning types as needed

# Load the cleaned dataset
df_filtered = pd.read_csv('filtered_reviews.csv')

# Check for the quick summary
df_filtered.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 54611 entries, 0 to 54610
Data columns (total 16 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   review_id      54611 non-null  object
 1   user_id        54611 non-null  object
 2   business_id    54611 non-null  object
 3   stars_x        54611 non-null  int64
 4   date           54611 non-null  object
 5   name           54611 non-null  object
 6   city           54611 non-null  object
 7   state          54611 non-null  object
 8   postal_code    54611 non-null  int64
 9   latitude       54611 non-null  float64
 10  longitude      54611 non-null  float64
 11  stars_y        54611 non-null  float64
 12  review_count   54611 non-null  int64
 13  categories     54611 non-null  object
 14  cleaned_text   54611 non-null  object
 15  review_length  54611 non-null  int64
dtypes: float64(3), int64(4), object(9)
memory usage: 6.7+ MB
```

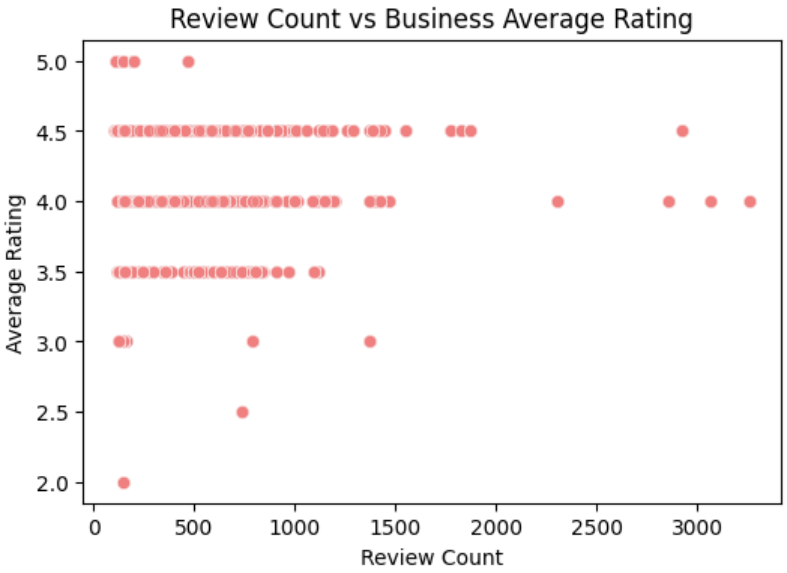
1. Distribution of User-Provided Star Ratings vs Business Average Star Ratings

In [2]:
```python
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(8, 4))
red_color = "#F08080"

# Distribution of user-provided star ratings (stars_x)
plt.subplot(1, 2, 1)
sns.countplot(x='stars_x', data=df_filtered, color=red_color)
plt.title('Distribution of User-Provided Star Ratings')
plt.xlabel('Stars (User Ratings)')
plt.ylabel('Count')

# Distribution of business average star ratings (stars_y)
plt.subplot(1, 2, 2)
sns.histplot(df_filtered['stars_y'], bins=10, kde=True, color=red_color)
plt.title('Distribution of Business Average Star Ratings')
plt.xlabel('Stars (Business Average Ratings)')
plt.ylabel('Frequency')

plt.tight_layout()
plt.show()
```



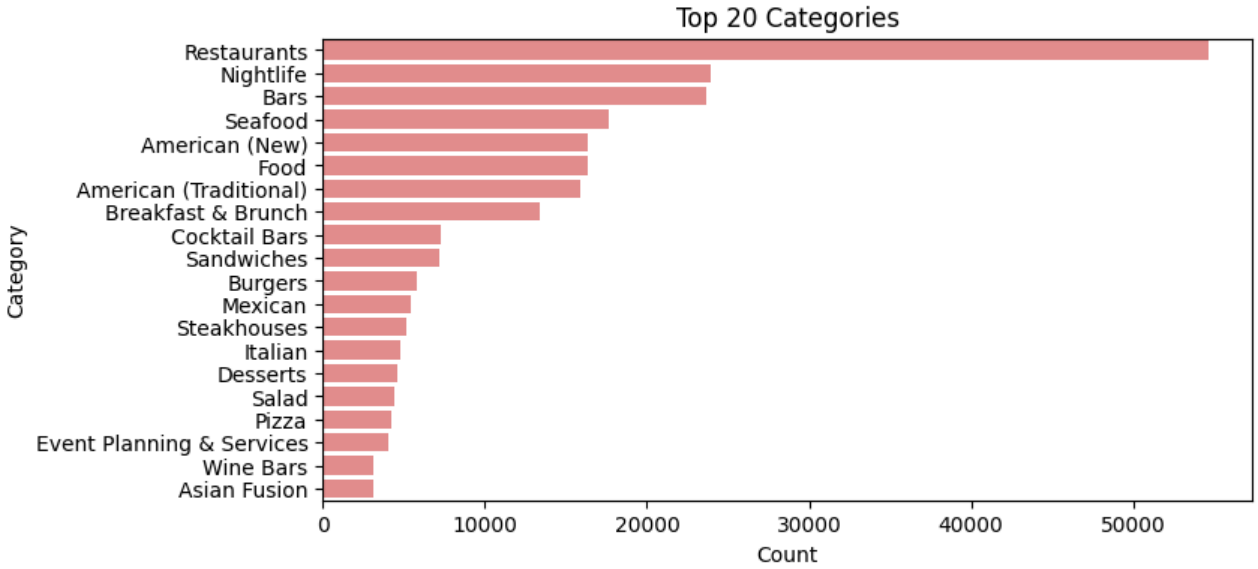
2. Correlation Between Review Count and Business Average Rating

In [3]:
```
plt.figure(figsize=(6, 4))
red_color = "#F08080"
sns.scatterplot(x='review_count', y='stars_y', data=df_filtered, alpha=0.3, color=red_color)
plt.title('Review Count vs Business Average Rating')
plt.xlabel('Review Count')
plt.ylabel('Average Rating')
plt.show()
```



3. Top 20 Categories

In [4]:
```
plt.figure(figsize=(8, 4))
"#F08080"
all_categories = df_filtered['categories'].str.split(',').explode().str.strip()  # Split and explode the categories
top_categories = all_categories.value_counts().head(20)  # Get the top 20 categories
sns.barplot(x=top_categories.values, y=top_categories.index, color=red_color)
plt.title('Top 20 Categories')
plt.xlabel('Count')
plt.ylabel('Category')
plt.show()
```



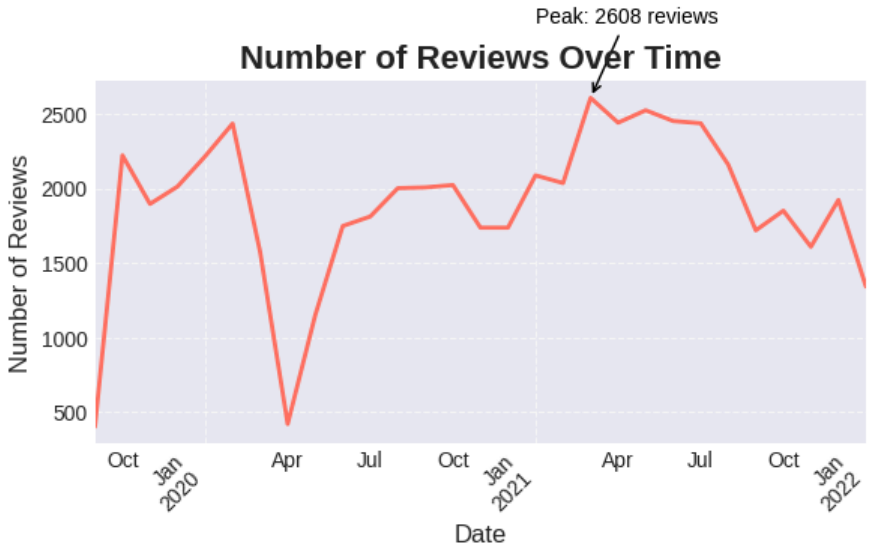
4. Review Length vs User Ratings

In [5]:
```
plt.figure(figsize=(6, 4))
red_color = "#F08080"
```

```
sns.boxplot(x='stars_x', y='review_length', data=df_filtered, color=red_color)
plt.title('Review Length vs User Rating')
plt.xlabel('User Rating (Stars)')
plt.ylabel('Review Length (Characters)')
plt.show()
```

5. Distribution of Reviews Over Time

In [6]:
```
import matplotlib.pyplot as plt

# Convert 'date' column to datetime format
df_filtered['date'] = pd.to_datetime(df_filtered['date'], errors='coerce')

# Set a style for the plot
plt.style.use('seaborn-darkgrid')

# Plot the number of reviews over time by month
plt.figure(figsize=(6, 4))
review_counts_by_month = df_filtered['date'].dt.to_period("M").value_counts().sort_index()

# Plot with enhancements
ax = review_counts_by_month.plot(kind='line', linewidth=2, color='#ff6f61')  # Thicker line and custom color

# Add titles and labels
plt.title('Number of Reviews Over Time', fontsize=16, fontweight='bold')
plt.xlabel('Date', fontsize=12)
plt.ylabel('Number of Reviews', fontsize=12)

# Customize x-ticks
plt.xticks(rotation=45, ha='right')

# Add a grid for better readability
plt.grid(True, linestyle='--', alpha=0.6)

# Highlighting the highest review count
max_reviews = review_counts_by_month.max()
max_date = review_counts_by_month.idxmax()

ax.annotate(f'Peak: {max_reviews} reviews', xy=(max_date, max_reviews),
            xytext=(max_date - 2, max_reviews + 500),
            arrowprops=dict(facecolor='black', arrowstyle='->'),
            fontsize=10, color='black')

plt.tight_layout()
plt.show()
```



Insights:

- The distribution of user-provided star ratings shows that the most common ratings are on the higher end (4 and 5 stars).
- There is no clear linear relationship between the number of reviews and the average business rating.
- The most common categories are American, Bars, Seafood, and Breakfast & Brunch.
- Reviews that are either very positive (5 stars) or very negative (1 star) tend to be longer, indicating that customers may feel more inclined to elaborate when they have strong opinions.
- There is a noticeable increase in the number of reviews over time, especially in the last few years. This could indicate that the platform or business has grown in popularity.
- The noticeable dips in review patterns is during the pandemic period, potentially reflecting shifts in consumer behavior.
- The higher frequency of 4 and 5-star reviews could indicate
  reviewer bias, where people who are more likely to leave reviews tend to have positive experiences.

Inferential Analysis

Compare Average Ratings Across Categories

In [12]:
```
from scipy import stats
import pandas as pd

# Let's group the data by categories and get the average ratings
category_groups = df_filtered.groupby('categories')['stars_y'].mean()

# Perform a one-way ANOVA test to see if there are significant differences in means across categories
anova_result = stats.f_oneway(*(df_filtered[df_filtered['categories'] == category]['stars_y'] for category in df_filt

print(f'ANOVA Result: F-statistic = {anova_result.statistic}, p-value = {anova_result.pvalue}')

# Interpret the results
if anova_result.pvalue < 0.05: # Changed anova_result.p_value to anova_result.pvalue
    print("There is a significant difference in ratings across restaurant categories.")
else:
    print("There is no significant difference in ratings across restaurant categories.")
```

```
ANOVA Result: F-statistic = 9246.245460853697, p-value = 0.0
There is a significant difference in ratings across restaurant categories.
```

One-way ANOVA to test if ratings differ by city.

In [13]:
```
import pandas as pd
import scipy.stats as stats

# Load your dataset
df = pd.read_csv('filtered_reviews.csv')
```

```python
# Group ratings by city
groups = [group['stars_x'].values for name, group in df.groupby('city')]

# Perform one-way ANOVA test
f_stat, p_value = stats.f_oneway(*groups)

# Display the results
print(f"F-statistic: {f_stat}")
print(f"P-value: {p_value}")

if p_value < 0.05:
    print("There is a significant difference in ratings across cities.")
else:
    print("There is no significant difference in ratings across cities.")
```

```
F-statistic: 19.599299061108265
P-value: 2.8444121798675206e-124
There is a significant difference in ratings across cities.
```

LDA and VADER Sentiment Analysis on the Restaurant reviews.

Topic Modeling

Topic Modeling is an unsupervised learning approach to clustering documents, to discover topics based on their contents. LDA, or Latent Derelicht Analysis is a probabilistic model, and to obtain cluster assignments, it uses two probability values: P(word | topics) and P(topics | documents). These values are calculated based on an initial random assignment, after which they are repeated for each word in each document, to decide their topic assignment. In an iterative procedure, these probabilities are calculated multiple times, until the convergence of the algorithm.

For each review, lets extract the dominant topic using our LDA model followed by VADER sentiment analysis to get the sentiment score (positive, neutral, or negative). After assigning sentiment scores to each review, calculate the sentiment distribution for each topic.

Sentiment Analysis

Sentiment Analysis also known as opinion mining, is a method for identifying the positive, negative, or neutral tone of a piece of text,it analyzes words and phrases to determine if a review is positive, negative, or neutral.

After we identified positive and negative sentiments, we would like to further explore what specific messages they conveyed. For example, among all negative reviews, what aspects of restaurants do they talk about the most? Do they complain about food, services or environments?

To solve this problem, we used topic modeling to identify key messages or topics customers are talking about. We determine number of topics based on coherence score, which measures score a single topic by measuring the degree of semantic similarity between high scoring words in the topic. Then we identified the topic based on the word distribution of the topic.

Using VADER (Rule-Based Method)

VADER (Valence Aware Dictionary and Sentiment Reasoner) is a lexicon-based sentiment analysis tool, which provides a sentiment score based on the polarity of the text. It's suitable for classifying sentiment into categories like positive, neutral, or negative.

Here's how we can classify review sentiment using VADER: • Compound Score: The VADER analyzer gives a compound score that ranges from -1 (most negative) to 1 (most positive). Using simple thresholds, you can classify the review as positive, negative, or neutral.

- compound_score >= 0.05: Positive
- compound_score <= -0.05: Negative
- otherwise: Neutral

Data Preprocessing for Text Data

- Let's preprocessing text data to make it ready for topic modeling and sentiment analysis:

1. Stop word removal: Removing common words (like "the", "is", "and") from the reviews that don't add meaningful information.
2. Tokenization: The text in each review is split into individual words (tokens).
3. Creating Bigrams and Trigrams: allows frequently co-occurring words to be treated as single entities.
4. Lemmatization: Reducing each word to its base form
5. Also Part-of-Speech Filtering for keping nouns, adjectives, verbs, and adverbs.

- Converting text to bag of words:

1. Prior to topic modelling, the tokenized and lemmatized text is

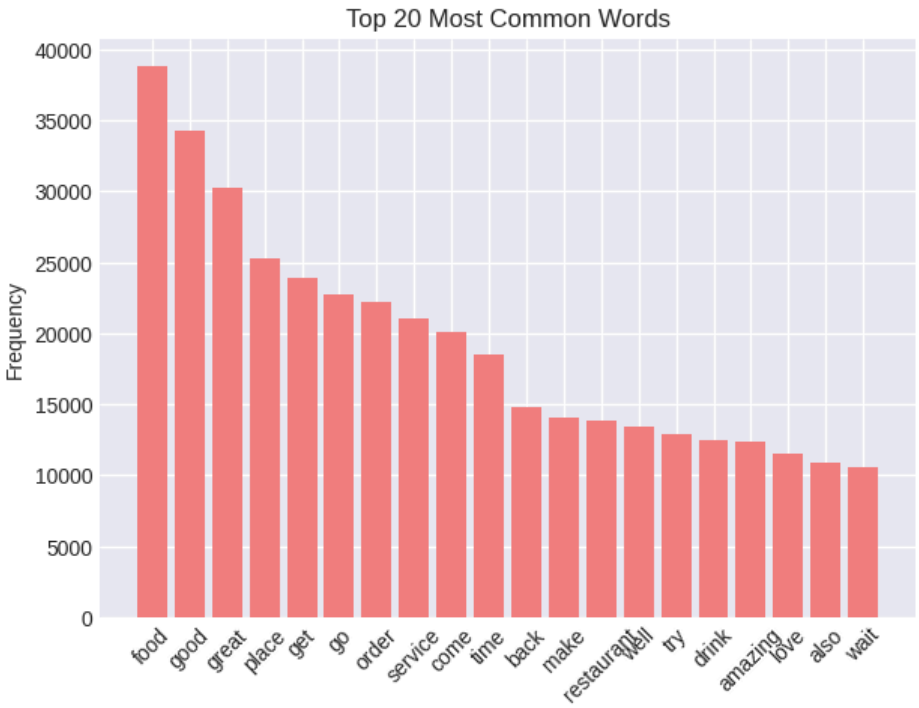
converetd to a bag of words — which is a kind of dictionary where the key is the word and value is the number of times that word occurs in the entire corpus.

In [14]:
```
# Pipeline for data preprocessing on text data

import nltk
from gensim.utils import simple_preprocess
from gensim.models.phrases import Phrases, Phraser
from nltk.corpus import stopwords
import spacy

# Load necessary resources
nltk.download('stopwords')
stop_words = stopwords.words('english')
nlp = spacy.load('en_core_web_sm', disable=['parser', 'ner'])

# Tokenization and stopword removal
texts = [simple_preprocess(doc) for doc in df_filtered['cleaned_text']]
texts = [[word for word in doc if word not in stop_words] for doc in texts]

# Creating bigrams and trigrams
bigram = Phrases(texts, min_count=5, threshold=100)
trigram = Phrases(bigram[texts], threshold=100)
bigram_mod = Phraser(bigram)
trigram_mod = Phraser(trigram)

texts_bigrams = [bigram_mod[doc] for doc in texts]

# Lemmatization
def lemmatization(texts, allowed_postags=['NOUN', 'ADJ', 'VERB', 'ADV']):
    texts_out = []
    for sent in texts:
        doc = nlp(" ".join(sent))
        texts_out.append([token.lemma_ for token in doc if token.pos_ in allowed_postags])
    return texts_out

texts_lemmatized = lemmatization(texts_bigrams)
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

In [15]:
```
from collections import Counter
import matplotlib.pyplot as plt

# Step 1: Flatten the list of tokens
all_words = [word for sublist in texts_lemmatized for word in sublist]

# Step 2: Count the frequency of each word
word_freq = Counter(all_words)

# Step 3: Display the top 20 most common words
most_common_words = word_freq.most_common(20)
print("Most Common Words:", most_common_words)

# Step 4: Visualize the word frequencies with a bar plot
words, counts = zip(*most_common_words)
plt.figure(figsize=(7, 5))
plt.bar(words, counts, color='#F08080')
plt.title('Top 20 Most Common Words')
plt.ylabel('Frequency')
plt.xticks(rotation=45)
plt.show()
```

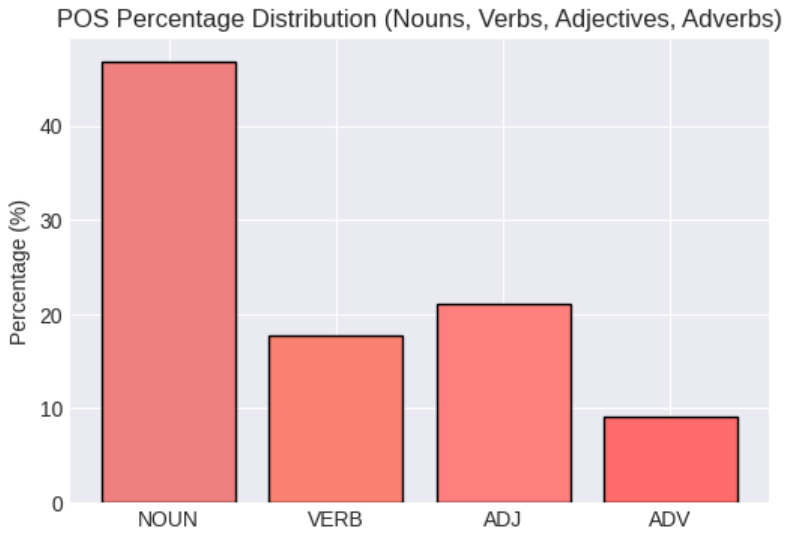
```
Most Common Words: [('food', 38803), ('good', 34220), ('great', 30216), ('place', 25313), ('get', 23848), ('go', 2273
4), ('order', 22228), ('service', 21001), ('come', 20103), ('time', 18511), ('back', 14763), ('make', 14054), ('resta
urant', 13803), ('well', 13378), ('try', 12877), ('drink', 12509), ('amazing', 12410), ('love', 11531), ('also', 1082
4), ('wait', 10540)]
```



In [16]:

```python
import spacy
import matplotlib.pyplot as plt

# Load the SpaCy model
nlp = spacy.load('en_core_web_sm', disable=['parser', 'ner'])

# Function to count POS categories
def count_pos_percentages(texts):
    total_words = 0
    pos_counts = {'NOUN': 0, 'VERB': 0, 'ADJ': 0, 'ADV': 0}

    for doc in texts:
        spacy_doc = nlp(" ".join(doc))  # Join tokens and process with SpaCy
        for token in spacy_doc:
            if token.pos_ in pos_counts:  # Only count NOUN, VERB, ADJ, ADV
                pos_counts[token.pos_] += 1
            total_words += 1  # Keep track of total words

    # Calculate percentages for each POS
    pos_percentages = {pos: (count / total_words) * 100 for pos, count in pos_counts.items()}

    return pos_percentages

# Call the function on the lemmatized texts
pos_percentages = count_pos_percentages(texts_lemmatized)

# Display the percentages
print("POS Percentages:", pos_percentages)
# Define the colors
colors = ['#F08080', '#FA8072', '#FF7F7F', '#FF6A6A']

# Visualize the POS percentages
plt.figure(figsize=(6, 4))
plt.bar(pos_percentages.keys(), pos_percentages.values(), color = colors, edgecolor ='black')
plt.title('POS Percentage Distribution (Nouns, Verbs, Adjectives, Adverbs)')
plt.ylabel('Percentage (%)')
plt.show()
```

```
POS Percentages: {'NOUN': 46.82913993382868, 'VERB': 17.77705215620296, 'ADJ': 21.17316237733753, 'ADV': 9.1759376282
97497}
```



In [17]:
```
from gensim.corpora import Dictionary
from gensim.models import LdaModel
from gensim.models.coherencemodel import CoherenceModel

# Create a dictionary and corpus for LDA
id2word = Dictionary(texts_lemmatized)
corpus = [id2word.doc2bow(text) for text in texts_lemmatized]

# Build LDA model and evaluate coherence
lda_model = LdaModel(corpus=corpus, id2word=id2word, num_topics=4, random_state=100, update_every=1, passes=10, alpha

# Coherence model
coherence_model = CoherenceModel(model=lda_model, texts=texts_lemmatized, dictionary=id2word, coherence='c_v')
coherence_score = coherence_model.get_coherence()
print(f'Coherence Score: {coherence_score}')
```

```
Coherence Score: 0.5015917155704872
```

Lets check the the topic-word distributions and the topic probabilities.

In [18]:
```
# Get the top words for each topic
topics = lda_model.print_topics(num_words=10)  # num_words specifies how many words to display per topic
for topic in topics:
    print(topic)
```

```
(0, '0.019*"get" + 0.018*"order" + 0.016*"go" + 0.015*"come" + 0.015*"table" + 0.013*"wait" + 0.013*"time" + 0.013*"t
ake" + 0.012*"say" + 0.012*"food"')
(1, '0.041*"food" + 0.040*"great" + 0.035*"good" + 0.030*"place" + 0.024*"service" + 0.017*"amazing" + 0.015*"go" +
0.015*"love" + 0.013*"back" + 0.013*"definitely"')
(2, '0.033*"parking" + 0.022*"view" + 0.018*"bar" + 0.013*"room" + 0.012*"beach" + 0.012*"hotel" + 0.011*"lot" + 0.01
1*"sit" + 0.010*"area" + 0.010*"water"')
(3, '0.020*"order" + 0.015*"good" + 0.015*"get" + 0.013*"fry" + 0.011*"flavor" + 0.010*"also" + 0.010*"sauce" + 0.010
*"taste" + 0.009*"side" + 0.009*"chicken"')
```

The LDA topics that are generated show the most frequent and relevant words for each topic, along with their weights, which represent the probability of those words being part of that specific topic.

Lets label these Topics based on frequency of words and weights assigned to them:

- 1- Order & Dining Experience: Customer experience with ordering, waiting times, and interactions around the table.
- 2- Overall Experience: Positive reviews related to the quality of food and service, with a high likelihood of customers recommending the restaurant.
- 3- Ambiance & Location: Comments about the restaurant's ambiance, view, and parking or proximity to scenic spots like the beach.
- 4- Food Quality & Taste: Focus on the taste and quality of food, with emphasis on specific dishes and flavors.

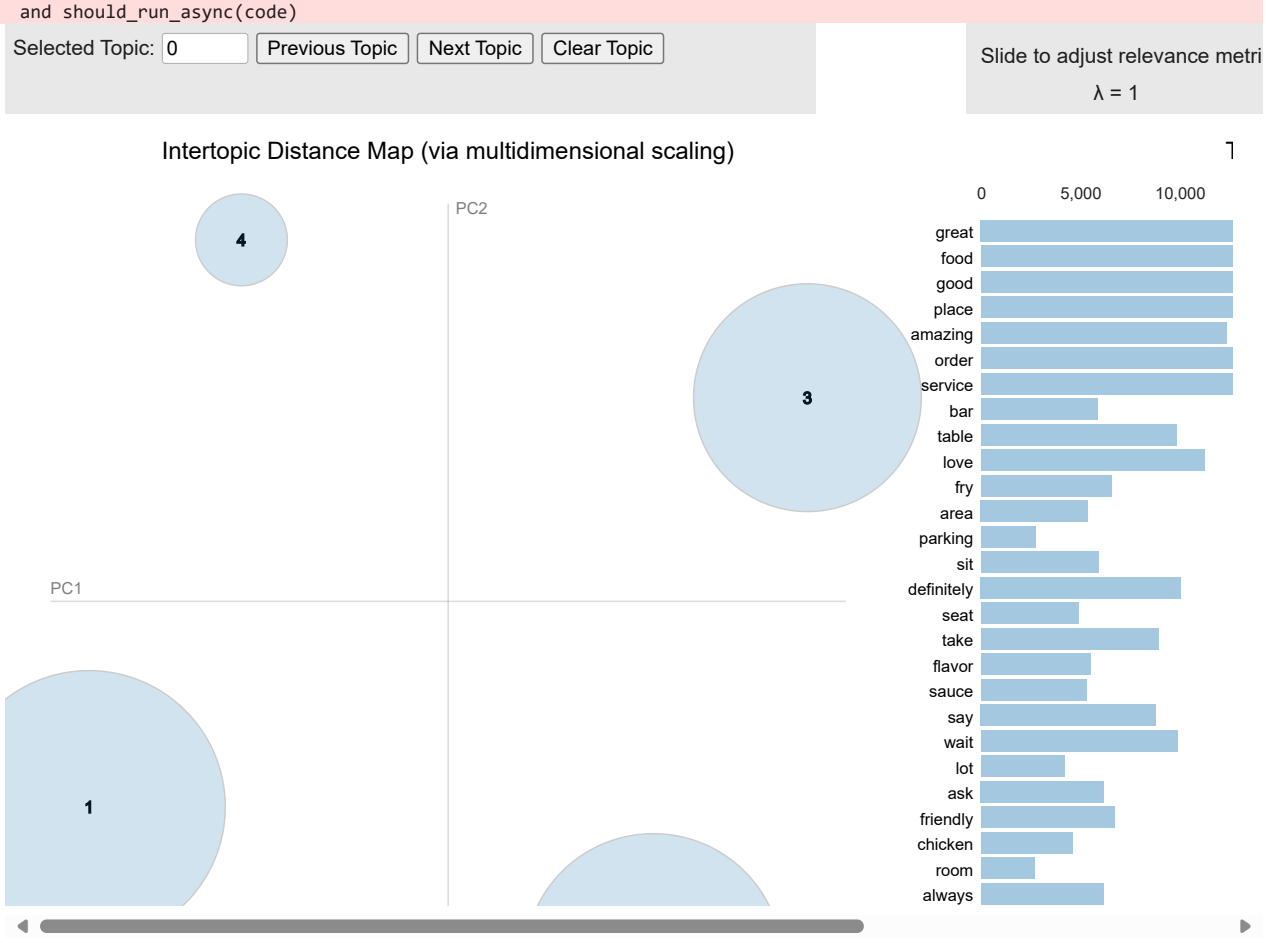
Now lets analyze the sentiment score associated with each topic to understand if customers are satisfied with specific aspects like food, service, or the ambiance.

Visualizing these topics with a tool like pyLDAvis helps to see how the topics are distributed across reviews and their importance.

In [ ]:
```
import pyLDAvis.gensim_models as gensimvis
import pyLDAvis

# Prepare the data for visualization
lda_vis_data = gensimvis.prepare(lda_model, corpus, id2word, mds='tsne')

# Enable pyLDAvis to display in the notebook
pyLDAvis.enable_notebook()

# Display the visualization
pyLDAvis.display(lda_vis_data)
```

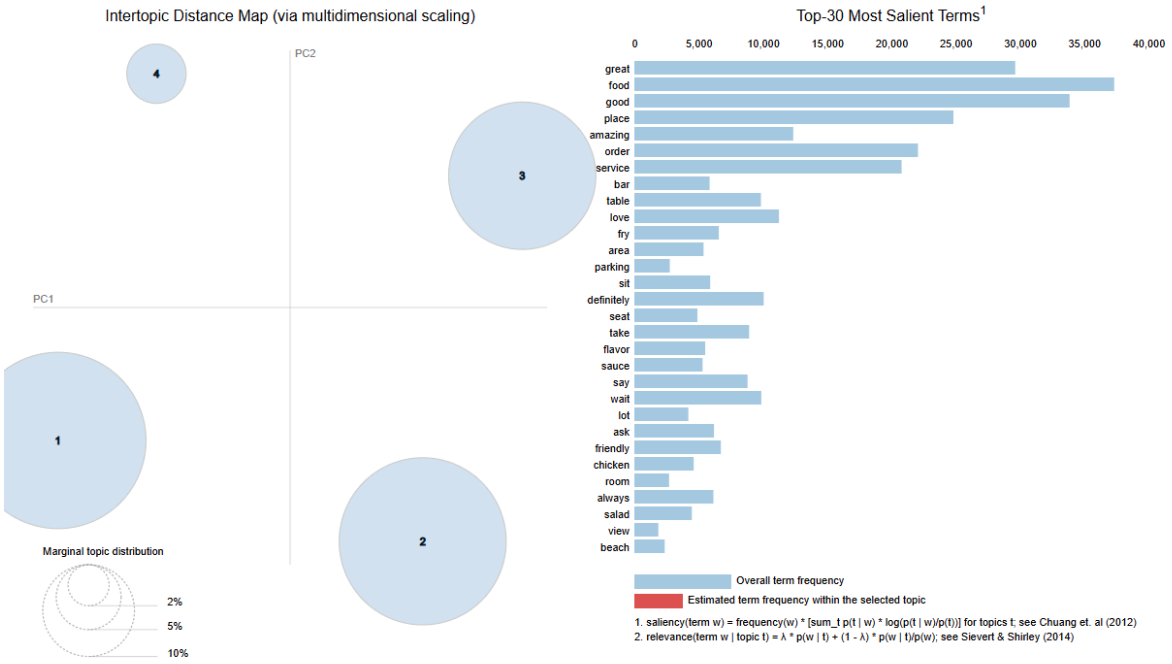
```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not ca
ll `transform_cell` automatically in the future. Please pass the result to `transformed_cell` argument and any except
ion that happen during thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

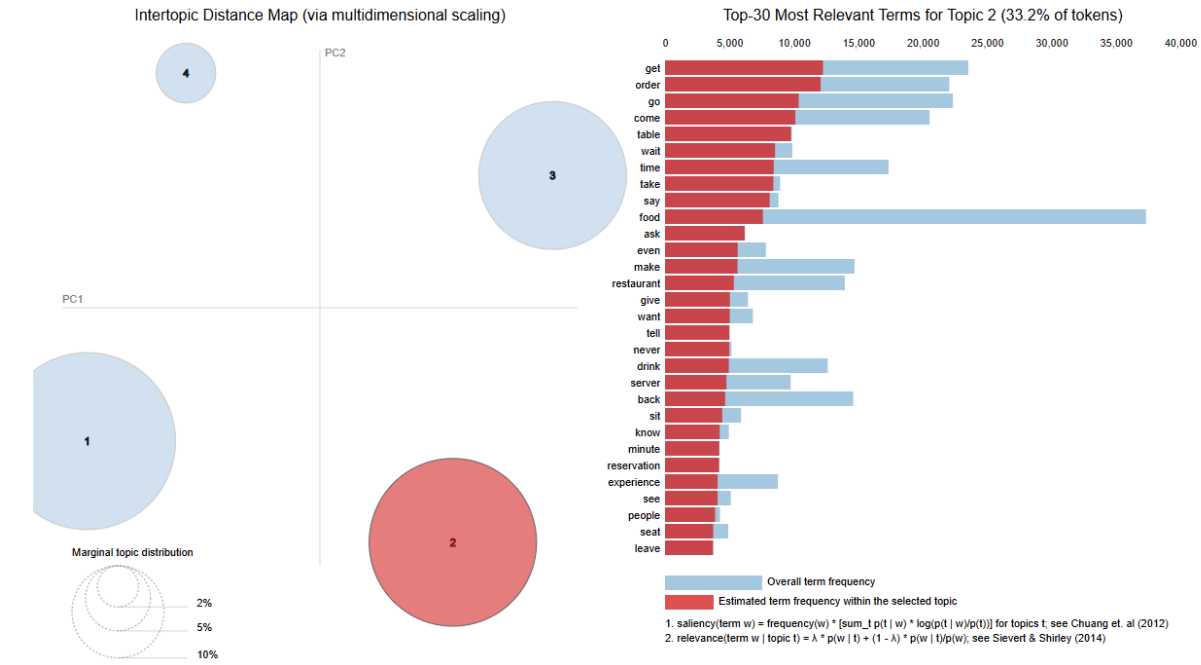
Out[ ]:



In [ ]:
```
# Save the visualization to an HTML file
pyLDAvis.save_html(lda_vis_data, 'lda_visualization.html') # Save before download

# Download the HTML file using google.colab.files
from google.colab import files
files.download('lda_visualization.html')
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not ca
ll `transform_cell` automatically in the future. Please pass the result to `transformed_cell` argument and any except
ion that happen during thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```





Let's analyze the distribution of sentiments (positive, negative, neutral) for each topic extracted from the LDA model, and understand how different topics are perceived by customers.

In [40]:
```
from collections import defaultdict
import pandas as pd

# Create a dictionary to store topic sentiment counts
topic_sentiment = defaultdict(lambda: {'Positive': 0, 'Negative': 0, 'Neutral': 0})

# Function to assign sentiment to a topic
def assign_sentiment_to_topic(ldamodel, corpus, texts, sentiment_scores):
    for i, row in enumerate(ldamodel[corpus]):
        # Get dominant topic for the document
        dominant_topic = sorted(row[0], key=lambda x: x[1], reverse=True)[0][0]
        # Get sentiment of the corresponding review
        sentiment = sentiment_scores[i][1]  # Make sure this accesses the sentiment score
        # Update the topic_sentiment dictionary with the review's sentiment
```

```
        if sentiment > 0.05:
            topic_sentiment[dominant_topic]['Positive'] += 1
        elif sentiment < -0.05:
            topic_sentiment[dominant_topic]['Negative'] += 1
        else:
            topic_sentiment[dominant_topic]['Neutral'] += 1

# Use the display function from pandas to show the dataframe
display(topic_sentiment_df)
```

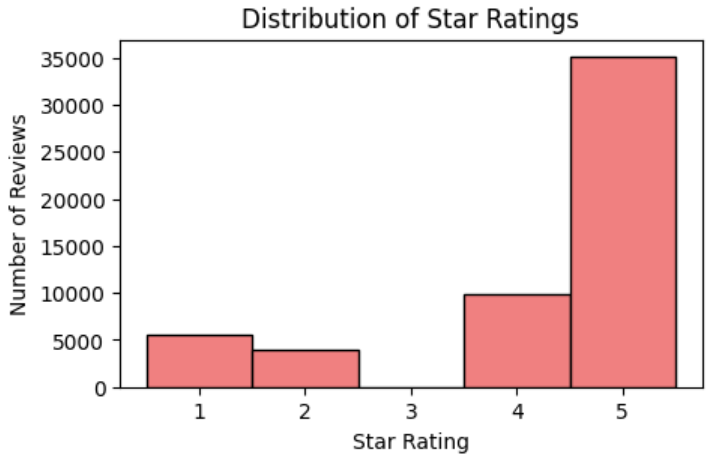
	Positive	Negative	Neutral
1	32893	338	167
0	8601	2832	287
3	8816	429	87
2	122	12	27

The table is showing the distribution of sentiment (Positive, Negative, Neutral) for each topic. While many customers are happy with wait times and efficiency, there's a significant amount of negative feedback, indicating this might be an area for improvement.

Let's analyze the distribution of review ratings to understand the overall pattern.

In [ ]:
```
import pandas as pd
import matplotlib.pyplot as plt

# Load the dataframe
df_filtered = pd.read_csv('filtered_reviews.csv')

# Check distribution of star ratings
plt.figure(figsize=(5, 3))
plt.hist(df_filtered['stars_x'], bins=[0.5, 1.5, 2.5, 3.5, 4.5, 5.5], edgecolor='black',color='#F08080')
plt.xlabel('Star Rating')
plt.ylabel('Number of Reviews')
plt.title('Distribution of Star Ratings')
plt.xticks([1, 2, 3, 4, 5])
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not ca
ll `transform_cell` automatically in the future. Please pass the result to `transformed_cell` argument and any except
ion that happen during thetransform in `preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```



The ratings of each review also follow the same trend, with a significant skew towards 5-star ratings, followed by 4-star rating

Sentiment Scores using Vader

For each document in the corpus, the LDA model generates a list of topics and their associated probabilities. The sorted function is used to extract the dominant topic (the one with the highest probability) for each document.

Sentiment Analysis: For each document, the VADER sentiment analyzer computes the compound sentiment score of the associated review text.

- If the score is close to +1, the sentiment is positive.
- If the score is close to -1, the sentiment is negative.

In [42]:
```
!pip install vaderSentiment
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer

# Function to extract topics and perform sentiment analysis
def get_sentiment_and_topics(ldamodel, corpus, texts):
    analyzer = SentimentIntensityAnalyzer()
    topic_sentiments = []

    for i, row in enumerate(ldamodel[corpus]):
        # Extract the dominant topic for each review
        dominant_topic = sorted(row[0], key=lambda x: x[1], reverse=True)[0][0]
        # Calculate sentiment score for the review text using VADER
        sentiment = analyzer.polarity_scores(texts[i])['compound']
        # Append a tuple of dominant topic and sentiment scor
        topic_sentiments.append((dominant_topic, sentiment))

    return topic_sentiments

# Get the sentiment scores and dominant topics for each review
df_filtered['Topic_Sentiment'] = get_sentiment_and_topics(lda_model, corpus, df_filtered['cleaned_text'])
```

```
Requirement already satisfied: vaderSentiment in /usr/local/lib/python3.10/dist-packages (3.3.2)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from vaderSentiment) (2.32.3)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->va
derSentiment) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->vaderSentimen
t) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->vaderSen
timent) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->vaderSen
timent) (2024.8.30)
```

In [26]:
```
# Separate the topics and sentiment into different columns
df_filtered[['Dominant_Topic', 'Sentiment_Score']] = pd.DataFrame(df_filtered['Topic_Sentiment'].tolist(), index=df_f

# Calculate Positive and Negative Sentiment Scores
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
analyzer = SentimentIntensityAnalyzer()

def get_sentiment_scores(text):
  scores = analyzer.polarity_scores(text)
  return scores['pos'], scores['neg']

df_filtered[['Positive_Sentiment', 'Negative_Sentiment']] = df_filtered['cleaned_text'].apply(get_sentiment_scores).a

# Save the DataFrame for Tableau import
df_filtered.to_csv('topics_and_sentiments_for_tableau.csv', index=False)
```

Apply the function and create the new column

In [43]:
```
# Separate the topics and sentiment into different columns
df_filtered[['Dominant_Topic', 'Sentiment_Score']] = pd.DataFrame(df_filtered['Topic_Sentiment'].tolist(), index=df_f

# Save the DataFrame for Tableau import
df_filtered.to_csv('topics_and_sentiments_for_tableau.csv', index=False)
```

Summary

This code performs topic modeling and sentiment analysis on the reviews text data.

- It identifies the dominant topic of each review (e.g., food,

serive, ambience).

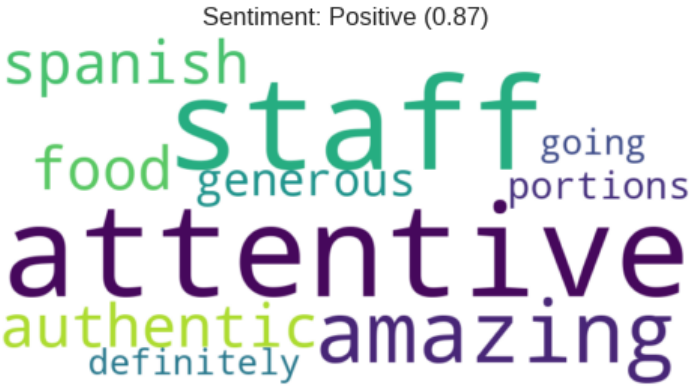
- It computes the sentiment score for each review using VADER (positive, negative, or neutral).
- The results are stored in a new DataFrame column (Topic_Sentiment) for further analysis, such as visualizing how sentiment varies by topic.

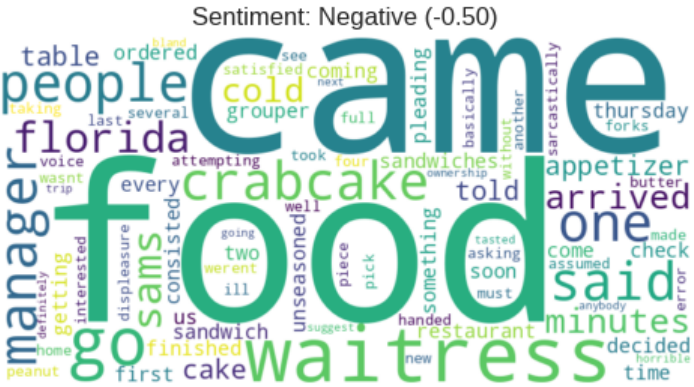
Visualization: Word Cloud

Let's generate a word cloud for each review to analyze the sentiment.

In [48]:
```
!pip install wordcloud
from wordcloud import WordCloud
import matplotlib.pyplot as plt

def generate_wordcloud(text, sentiment):
    """Generates and displays a word cloud from the given text,
    including the sentiment score in the title.

    Args:
        text (str): The text to generate the word cloud from.
        sentiment (float): The sentiment score of the text.
    """
    # Create a WordCloud object
    wordcloud = WordCloud(width=800, height=400, background_color='white').generate(text)

    # Display the generated image with sentiment score in the title:
    plt.figure(figsize=(6, 3))
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.axis("off")
    plt.title(f"Sentiment: {'Positive' if sentiment >= 0 else 'Negative'} ({sentiment:.2f})")  # Display sentiment sc
    plt.show()

# Initialize counters for positive and negative word clouds
positive_count = 0
negative_count = 0

# Iterate through the rows in the dataframe
for index, row in df_filtered.iterrows():
    text = row['cleaned_text']  # Use the cleaned review text
    sentiment = row['Sentiment_Score']  # Get the sentiment score

# Generate word cloud based on the sentiment score
    if sentiment >= 0 and positive_count < 2:  # Positive sentiment
        generate_wordcloud(text, sentiment)
        positive_count += 1
    elif sentiment < 0 and negative_count < 2:  # Negative sentiment
        generate_wordcloud(text, sentiment)
        negative_count += 1
# Stop once 2 positive and 2 negative word clouds have been generated
    if positive_count >= 2 and negative_count >= 1:
        break  # Exit the loop
```

```
Requirement already satisfied: wordcloud in /usr/local/lib/python3.10/dist-packages (1.9.3)
Requirement already satisfied: numpy>=1.6.1 in /usr/local/lib/python3.10/dist-packages (from wordcloud) (1.26.4)
Requirement already satisfied: pillow in /usr/local/lib/python3.10/dist-packages (from wordcloud) (10.4.0)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (from wordcloud) (3.7.1)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->wordclou
d) (1.3.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib->wordcloud)
(0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->wordclo
ud) (4.54.1)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->wordclo
ud) (1.4.7)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->wordclou
d) (24.1)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->wordclou
d) (3.1.4)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib->word
cloud) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplo
tlib->wordcloud) (1.16.0)
```





Inference from Text Analytics

1. The word "food" is the most frequent, indicating that the primary reason people are reviewing the restaurant revolves around the quality and experience of the food itself. This suggests that food quality is the dominant factor that influences customer reviews.

2. Words like "good," "great," "amazing," and "love" indicate that a significant portion of the reviews contains positive sentiment.

3. The word "wait" also appears in the top 20, indicating that wait times are a frequent topic of discussion. It's an operational area that need attention.

4. The presence of the word "back" suggests that customers often mention their desire to return or share their previous dining experiences. This indicates a level of customer loyalty, where diners are willing to come back if they had a positive experience.

5. The words "service" and "place" appear frequently, this suggests that customers are not only focused on the food but also place importance on how they are treated by staff and the ambiance of the location.

6. Neutral sentiment may be a signal of early-stage dissatisfaction, giving the restaurant an opportunity to address concerns before they lead to negative reviews.

Conclusion

From the sentiment analysis, it is evident that restaurants should prioritize delivering exceptional food while minimizing negative customer service experiences. Interestingly, being located in a popular area doesn't necessarily guarantee higher ratings. For aspiring restaurant owners, it is crucial to carefully evaluate the category they choose to enter, as this has a significant impact on the ratings they are likely to receive.

By focusing on delivering great food, improving service, and creating a balanced overall experience, restaurants can enhance customer satisfaction and achieve better ratings.

References

- Hutto, C. & Gilbert, E. (2014). VADER: A Parsimonious Rule-Based Model for Sentiment Analysis of Social Media Text.
- Model for Sentiment Analysis of Social Media Text. In Proceedings of the International AAAI Conference on Web and Social Media.
- Gensim Documentation. https://radimrehurek.com/gensim/ https://cs229.stanford.edu/proj2016/report/
- Nichols-LearningFromYelp-report.
- Blei, D., Ng, A., & Jordan, M. (2003). Latent Dirichlet Allocation. Journal of Machine Learning Research.
- About the Scoring — VaderSentiment 3.3.1 documentation

Future Work

In continuation of this project, the use of more deep learning NLP methods, like LSTM and BERT on the reviews would be helpful in extracting features from the language data.

Geospatial Analysis can be done to review trends and sentiments based on geographic locations to identify patterns.