Starbucks

Review Analysis

Ali Oraji

In [70]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from collections import Counter
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from textblob import TextBlob
import warnings
import re
import matplotlib.dates as mdates
from collections import Counter
warnings.filterwarnings('ignore')
```

In [71]:

```
starbucks_data = pd.read_csv('/kaggle/input/starbucks-reviews-dataset/reviews_data.csv')

starbucks_data.head()
```

Out[71]:

	name	location	Date	Rating	Review	Image_Links
0	Helen	Wichita Falls, TX	Reviewed Sept. 13, 2023	5.0	Amber and LaDonna at the Starbucks on Southwes...	['No Images']
1	Courtney	Apopka, FL	Reviewed July 16, 2023	5.0	** at the Starbucks by the fire station on 436...	['No Images']
2	Daynelle	Cranberry Twp, PA	Reviewed July 5, 2023	5.0	I just wanted to go out of my way to recognize...	['https://media.consumeraffairs.com/files/cach...
3	Taylor	Seattle, WA	Reviewed May 26, 2023	5.0	Me and my friend were at Starbucks and my card...	['No Images']
4	Tenessa	Gresham, OR	Reviewed Jan. 22, 2023	5.0	I'm on this kick of drinking 5 cups of warm wa...	['https://media.consumeraffairs.com/files/cach...

The basic statistical overview of the dataset reveals the following:

- Total Reviews: 850 reviews are present in the dataset.
- Unique Locations: The dataset covers 633 unique locations.
- Date Range: The reviews are dated between April 1, 2010, and September 9, 2017.
- Ratings: The average rating is approximately 1.87 (on a scale of 1 to 5). The minimum rating given is 1, and the maximum is 5.

** This overview indicates a relatively low average rating, which could suggest a trend of negative reviews. However, we should explore further to confirm this.

Sentiment analysis of the reviews

In [72]:
```
# Basic statistical overview
basic_stats = {
    "Total Reviews": starbucks_data.shape[0],
    "Unique Locations": starbucks_data['location'].nunique(),
    "Date Range": (starbucks_data['Date'].min(), starbucks_data['Date'].max()),
    "Rating": {
    "Average Rating": starbucks_data['Rating'].mean(),
    "Min Rating": starbucks_data['Rating'].min(),
    "Max Rating": starbucks_data['Rating'].max()
    }
}

basic_stats
```

Out[72]:
```
{'Total Reviews': 850,
 'Unique Locations': 633,
 'Date Range': ('Reviewed April 1, 2010', 'Reviewed Sept. 9, 2017'),
 'Rating': {'Average Rating': 1.8709219858156028,
  'Min Rating': 1.0,
  'Max Rating': 5.0}}
```

- we can use TextBlob for sentiment analysis. TextBlob is a simpler tool compared to VADER but is still effective for basic sentiment analysis. It provides a polarity score that ranges from -1 (very negative) to 1 (very positive).

In [73]:

```
# Function to classify sentiment using TextBlob
def classify_sentiment_textblob(row):
    sentiment = TextBlob(row).sentiment.polarity
    if sentiment > 0:
        return 'Positive'
    elif sentiment < 0:
        return 'Negative'
    else:
        return 'Neutral'

# Apply sentiment analysis using TextBlob to the Review column
starbucks_data['Sentiment_TB'] = starbucks_data['Review'].apply(classify_sentiment_textblob)

# Overview of sentiment distribution using TextBlob
sentiment_distribution_tb = starbucks_data['Sentiment_TB'].value_counts(normalize=True) * 100
sentiment_distribution_tb
```

Out[73]:

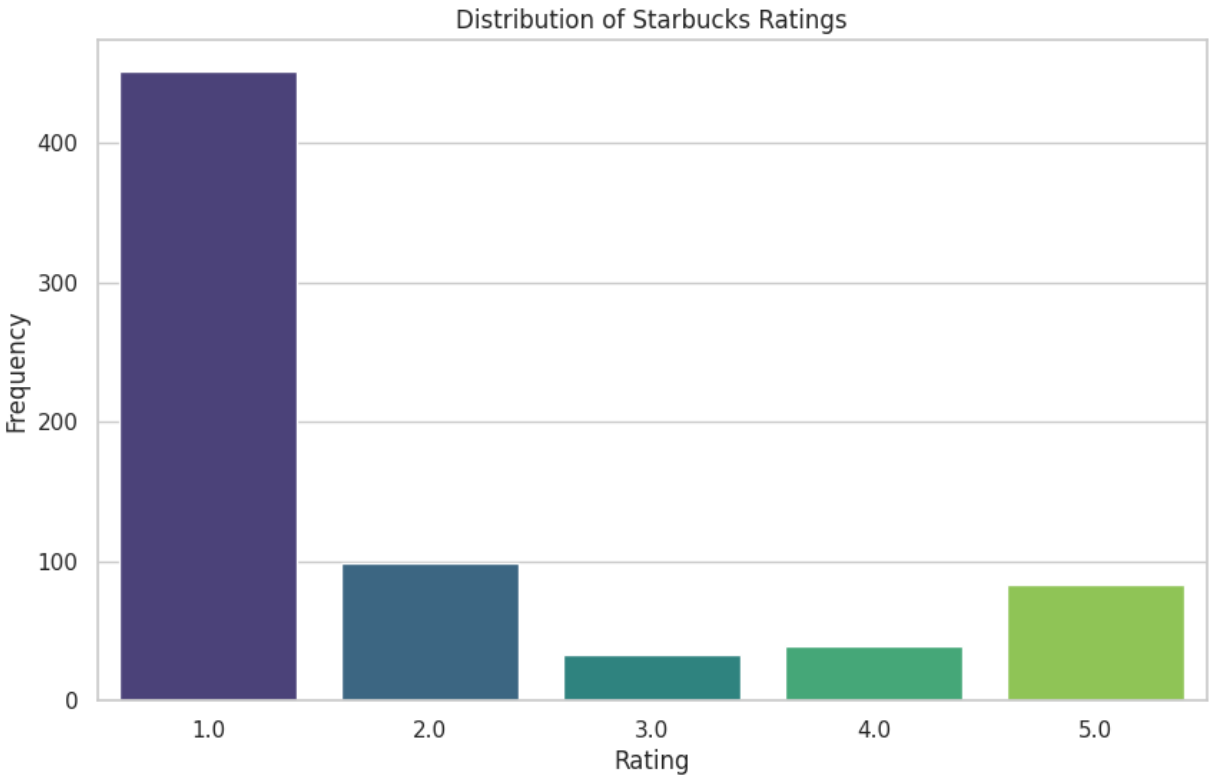
```
Sentiment_TB
Positive    52.000000
Negative    40.941176
Neutral      7.058824
Name: proportion, dtype: float64
```

The sentiment analysis using TextBlob reveals the following distribution:

1. **Positive Reviews:** 52%
2. **Negative Reviews:** 40.94%
3. **Neutral Reviews:** 7.06%

In [74]:

```
# Setting up the aesthetics for plots
sns.set(style="whitegrid")

# Plotting the distribution of ratings
plt.figure(figsize=(10, 6))
sns.countplot(x='Rating', data=starbucks_data, palette="viridis")
plt.title('Distribution of Starbucks Ratings')
plt.xlabel('Rating')
plt.ylabel('Frequency')
plt.show()
```



- The histogram shows the distribution of Starbucks ratings in the dataset. We observe a prominent trend of low ratings, with the majority being 1-star ratings. This contrasts with the sentiment analysis results, where a majority of the reviews were classified as positive. This discrepancy might suggest that customers who had negative experiences were more inclined to give low ratings, while the textual content of their reviews was more nuanced or mixed.

In [75]:

```
# Function for manual tokenization
def manual_tokenize_for_frequency(text):
    # Splitting the text into words using regular expressions
    words = re.findall(r'\b\w+\b', text.lower())
    return words

# Apply manual tokenization to the review column for all words
all_words_for_frequency = sum(starbucks_data['Review'].apply(manual_tokenize_for
_frequency), [])

# Count the frequency of each word
word_freq = Counter(all_words_for_frequency)

# Most common words in the manual analysis
common_words = word_freq.most_common(20)
common_words
```

Out[75]:

```
[('the', 3374),
 ('i', 3346),
 ('to', 2366),
 ('and', 2363),
 ('a', 1832),
 ('my', 1124),
 ('it', 1057),
 ('starbucks', 1055),
 ('of', 1014),
 ('was', 996),
 ('in', 960),
 ('for', 806),
 ('that', 798),
 ('they', 716),
 ('is', 713),
 ('me', 633),
 ('at', 627),
 ('on', 623),
 ('have', 613),
 ('coffee', 597)]
```

In [76]:

```
# Ensuring necessary NLTK resources are available

nltk.download('punkt')
nltk.download('stopwords')

stop_words = set(stopwords.words('english'))

# cleaning and tokenization
def clean_tokenize(text):

    tokens = word_tokenize(text.lower())
    tokens = [word for word in tokens if word.isalpha() and word not in stop_wor
ds]
    return tokens

# Reload the dataset as the code execution state was reset
starbucks_data = pd.read_csv('/kaggle/input/starbucks-reviews-dataset/reviews_da
ta.csv')

# Apply the function to the review column and concatenate all tokens
all_words = sum(starbucks_data['Review'].apply(clean_tokenize), [])

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

# Most common words
common_words = word_freq.most_common(20)
common_words
```

```
[nltk_data] Downloading package punkt to /usr/share/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /usr/share/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[76]:

```
[('starbucks', 1051),
 ('coffee', 595),
 ('customer', 286),
 ('one', 279),
 ('get', 276),
 ('drink', 275),
 ('store', 259),
 ('service', 255),
 ('time', 237),
 ('like', 224),
 ('order', 218),
 ('said', 217),
 ('go', 214),
 ('would', 213),
 ('card', 197),
 ('went', 180),
 ('asked', 164),
 ('back', 163),
 ('told', 159),
 ('ordered', 150)]
```

1. Temporal Analysis:

- We'll examine how ratings and sentiments have changed over time. This can provide insights into whether customer experiences have improved or declined. For the temporal analysis, we first need to parse the 'Date' column into a datetime format. Then, we can analyze the trends in ratings and sentiments over time.

2. Location-based Insights:

- We'll explore whether there are notable differences in ratings or sentiments across different locations.

In [77]:

```
import calendar

# Create a mapping for month abbreviations to their full names
month_abbr_to_full = {month: calendar.month_name[i] for i, month in enumerate(ca
lendar.month_abbr) if month}

# Function to replace abbreviated month names with full names
def replace_month_abbr(date_str):
    for abbr, full in month_abbr_to_full.items():
        if abbr in date_str:
            return date_str.replace(abbr, full)
    return date_str

# Apply the function to the 'Date' column
starbucks_data['Corrected_Date'] = starbucks_data['Date'].apply(replace_month_ab
br)

# Parsing the corrected 'Date' column into datetime format
starbucks_data['Parsed_Date'] = pd.to_datetime(starbucks_data['Corrected_Date'].
str.replace('Reviewed ', ''), errors='coerce')

# Overview of the dataset with parsed dates
starbucks_data.head()
```

Out[77]:

	name	location	Date	Rating	Review	Image_Links
0	Helen	Wichita Falls, TX	Reviewed Sept. 13, 2023	5.0	Amber and LaDonna at the Starbucks on Southwes...	['No Images']
1	Courtney	Apopka, FL	Reviewed July 16, 2023	5.0	** at the Starbucks by the fire station on 436...	['No Images']
2	Daynelle	Cranberry Twp, PA	Reviewed July 5, 2023	5.0	I just wanted to go out of my way to recognize...	['https://media.consumeraffairs.com/files/cach...
3	Taylor	Seattle, WA	Reviewed May 26, 2023	5.0	Me and my friend were at Starbucks and my card...	['No Images']
4	Tenessa	Gresham, OR	Reviewed Jan. 22, 2023	5.0	I'm on this kick of drinking 5 cups of warm wa...	['https://media.consumeraffairs.com/files/cach...

In [78]:

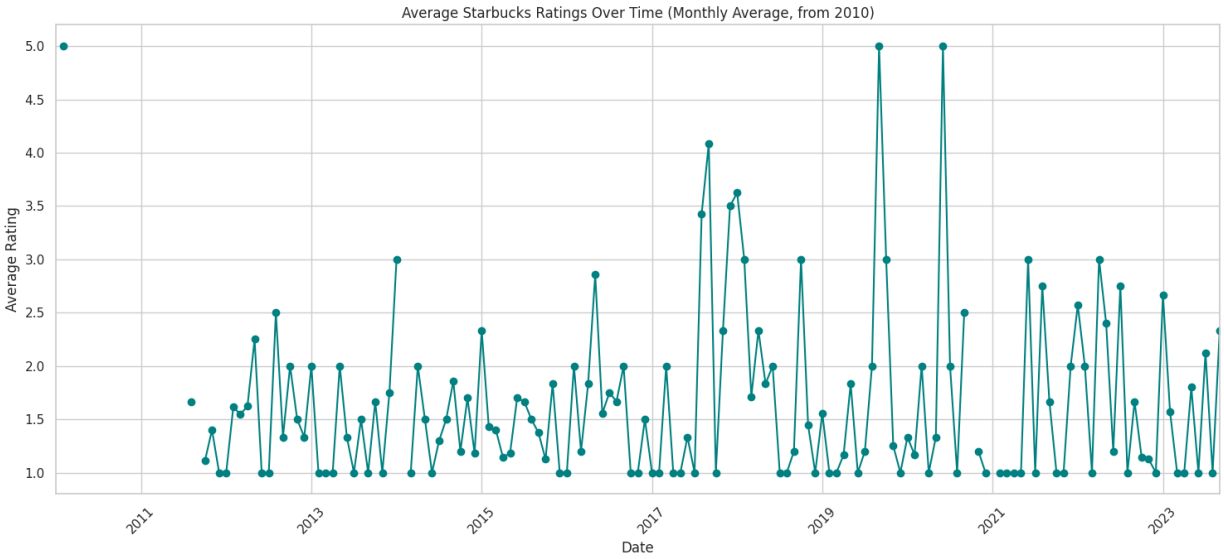
```
# Correcting the function for replacing abbreviated month names
def replace_month_abbr_corrected(date_str):
    for abbr, full in month_abbr_to_full.items():
        # Replace only if the abbreviation is followed by a period
        if f"{abbr}." in date_str:
            return date_str.replace(f"{abbr}.", full)
    return date_str

# Apply the corrected function to the 'Date' column
starbucks_data['Corrected_Date'] = starbucks_data['Date'].apply(replace_month_abbr_corrected)

# Parsing the corrected 'Date' column into datetime format
starbucks_data['Parsed_Date'] = pd.to_datetime(starbucks_data['Corrected_Date'].str.replace('Reviewed ', ''), errors='coerce')

# Overview of the dataset with parsed dates
starbucks_data.head()
```

Out[78]:

	name	location	Date	Rating	Review	Image_Links
0	Helen	Wichita Falls, TX	Reviewed Sept. 13, 2023	5.0	Amber and LaDonna at the Starbucks on Southwes...	['No Images']
1	Courtney	Apopka, FL	Reviewed July 16, 2023	5.0	** at the Starbucks by the fire station on 436...	['No Images']
2	Daynelle	Cranberry Twp, PA	Reviewed July 5, 2023	5.0	I just wanted to go out of my way to recognize...	['https://media.consumeraffairs.com/files/cach...
3	Taylor	Seattle, WA	Reviewed May 26, 2023	5.0	Me and my friend were at Starbucks and my card...	['No Images']
4	Tenessa	Gresham, OR	Reviewed Jan. 22, 2023	5.0	I'm on this kick of drinking 5 cups of warm wa...	['https://media.consumeraffairs.com/files/cach...

In [79]:

```
# Trim the dataset to start from 2010
trimmed_data = starbucks_data[starbucks_data['Parsed_Date'] >= '2010-01-01']

# Resample the trimmed data to monthly average
trimmed_monthly_average_ratings = trimmed_data.resample('M', on='Parsed_Date')
['Rating'].mean()

# Plotting monthly average ratings over time
plt.figure(figsize=(15, 7))
trimmed_monthly_average_ratings.plot(color='teal', marker='o', linestyle='-')
plt.title('Average Starbucks Ratings Over Time (Monthly Average, from 2010)')
plt.xlabel('Date')
plt.ylabel('Average Rating')
plt.grid(True)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



In [80]:

```
# Get sentiment polarity for each review
starbucks_data['sentiment'] = starbucks_data['Review'].apply(lambda x: TextBlob
(str(x)).sentiment.polarity)

# Classify sentiment as positive, negative, or neutral
starbucks_data['sentiment_label'] = starbucks_data['sentiment'].apply(lambda x:
'positive' if x > 0 else ('negative' if x < 0 else 'neutral'))
```

In [81]:

```
# Perform the group by operation again
sentiment_counts = starbucks_data.groupby([pd.Grouper(key='Parsed_Date', freq
='M'), 'sentiment_label']).size()
sentiment_proportions = sentiment_counts.groupby(level=0).apply(lambda x: x / x.
sum()).unstack().fillna(0)

# Check the resulting DataFrame columns to diagnose the issue
print(sentiment_proportions.columns)
```

```
Index(['negative', 'neutral', 'positive'], dtype='object', name='sentiment_la
bel')
```

In [82]:

```
# Check the DataFrame structure after groupby and unstack
print(sentiment_proportions.head())
```

sentiment_label		negative	neutral	positive
Parsed_Date	Parsed_Date			
2000-07-31	2000-07-31	0.0	1.0	0.0
2004-10-31	2004-10-31	0.0	1.0	0.0
2004-11-30	2004-11-30	1.0	0.0	0.0
2005-01-31	2005-01-31	0.0	1.0	0.0
2006-07-31	2006-07-31	0.0	0.0	1.0

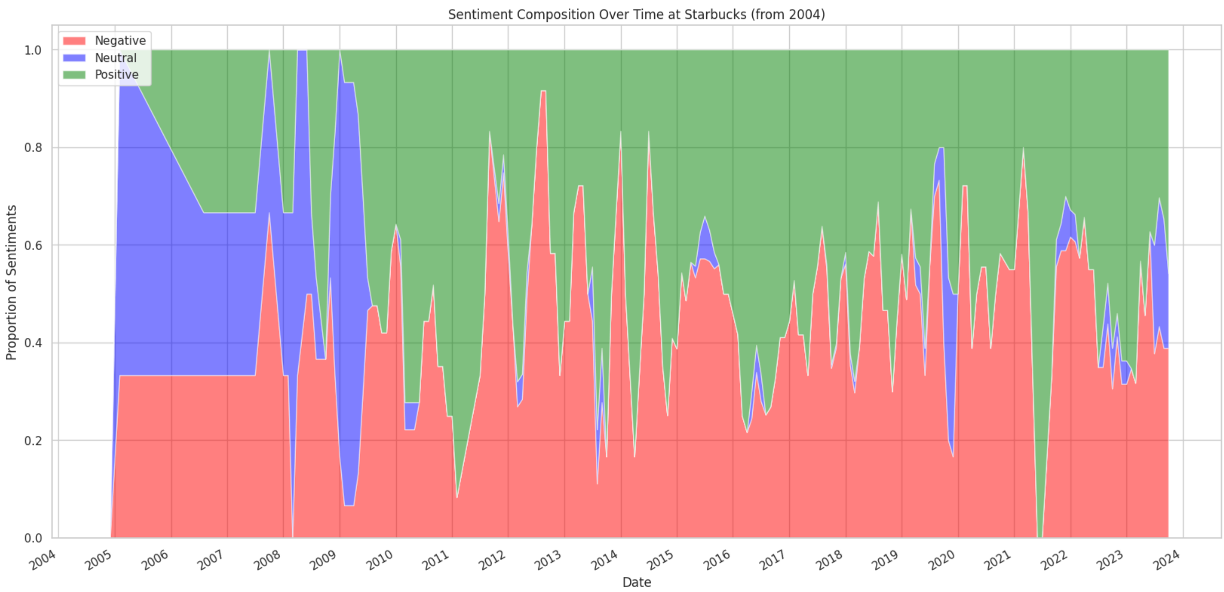
In [83]:

```
print(sentiment_proportions.columns)
print(sentiment_proportions.index)
```

```
Index(['negative', 'neutral', 'positive'], dtype='object', name='sentiment_la
bel')
MultiIndex([('2000-07-31', '2000-07-31'),
            ('2004-10-31', '2004-10-31'),
            ('2004-11-30', '2004-11-30'),
            ('2005-01-31', '2005-01-31'),
            ('2006-07-31', '2006-07-31'),
            ('2006-12-31', '2006-12-31'),
            ('2007-06-30', '2007-06-30'),
            ('2007-09-30', '2007-09-30'),
            ('2007-12-31', '2007-12-31'),
            ('2008-01-31', '2008-01-31'),
            ...
            ('2022-12-31', '2022-12-31'),
            ('2023-01-31', '2023-01-31'),
            ('2023-02-28', '2023-02-28'),
            ('2023-03-31', '2023-03-31'),
            ('2023-04-30', '2023-04-30'),
            ('2023-05-31', '2023-05-31'),
            ('2023-06-30', '2023-06-30'),
            ('2023-07-31', '2023-07-31'),
            ('2023-08-31', '2023-08-31'),
            ('2023-09-30', '2023-09-30')],
           names=['Parsed_Date', 'Parsed_Date'], length=187)
```

In [84]:

```
# Simplify the MultiIndex to a single-level index by selecting the first level
sentiment_proportions.index = sentiment_proportions.index.get_level_values(0)

# Now 'Parsed_Date' is the index, filter the DataFrame to exclude dates before 200
4
sentiment_proportions = sentiment_proportions[sentiment_proportions.index >= '20
04-01-01']

# Calculate the rolling mean using the 'Parsed_Date' index
smoothed_sentiments = sentiment_proportions.rolling(window=3).mean().fillna(0)

# Ensure the index is in datetime format for plotting
smoothed_sentiments.index = pd.to_datetime(smoothed_sentiments.index)

# Set figure size
plt.figure(figsize=(20, 10))

# Plotting the sentiment trends as a stacked area chart
plt.stackplot(smoothed_sentiments.index,
              [smoothed_sentiments['negative'], smoothed_sentiments['neutral'],
smoothed_sentiments['positive']],
              labels=['Negative', 'Neutral', 'Positive'],
              colors=['red', 'blue', 'green'],
              alpha=0.5)

# Formatting the plot
plt.title('Sentiment Composition Over Time at Starbucks (from 2004)')
plt.xlabel('Date')
plt.ylabel('Proportion of Sentiments')

# Formatting the x-axis to show dates clearly
plt.gca().xaxis.set_major_locator(mdates.YearLocator())
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y'))
plt.gcf().autofmt_xdate()  # Rotate the x-axis labels to prevent overlap

# Legend and grid
plt.legend(loc='upper left')
plt.grid(True)

# Show the plot
plt.show()
```



In [85]:

```
import matplotlib.pyplot as plt
import seaborn as sns
from wordcloud import WordCloud

# Generating a word cloud for reviews
text = " ".join(review for review in starbucks_data.Review)
wordcloud = WordCloud(background_color="white").generate(text)

# Display the generated word cloud
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

In [86]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import LatentDirichletAllocation

# Text classification - preparing the TF-IDF matrix
tfidf_vectorizer = TfidfVectorizer(max_df=0.95, min_df=2, stop_words='english')
tfidf = tfidf_vectorizer.fit_transform(starbucks_data['Review'])

# Example of LDA for topic modeling
lda = LatentDirichletAllocation(n_components=5, random_state=0)
lda.fit(tfidf)
```

Out[86]:

```
LatentDirichletAllocation(n_components=5, random_state=0)
```

In [87]:

```
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense

# Assuming 'Rating' as the label for simplicity
y = starbucks_data['Rating']

# Tokenizing text
tokenizer = Tokenizer(num_words=5000)
tokenizer.fit_on_texts(starbucks_data['Review'])
X = tokenizer.texts_to_sequences(starbucks_data['Review'])
X = pad_sequences(X, maxlen=200)

# Define LSTM model
model = Sequential()
model.add(Embedding(5000, 128, input_length=200))
model.add(LSTM(128, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accurac
y'])
```

In [88]:

```
from transformers import pipeline

# Initialize a summarization pipeline
summarizer = pipeline("summarization")

# Example: Summarize a review
summary = summarizer(starbucks_data['Review'][0], max_length=50, min_length=25,
do_sample=False)
print(summary[0]['summary_text'])
```

```
No model was supplied, defaulted to sshleifer/distilbart-cnn-12-6 and revisio
n a4f8f3e (https://huggingface.co/sshleifer/distilbart-cnn-12-6).
Using a pipeline without specifying a model name and revision in production i
s not recommended.


 Amber and LaDonna at the Starbucks on Southwest Parkway are always so warm a
nd welcoming . There is always a smile in their voice when they greet you at
the drive-thru. And their customer service is always spot-on .
```

In [89]:

```
import gensim
from gensim import corpora
from gensim.models.ldamodel import LdaModel
from nltk.tokenize import word_tokenize

# Preprocess and tokenize data
tokenized_reviews = [word_tokenize(review.lower()) for review in starbucks_data
['Review']]

# Create a dictionary and corpus
dictionary = corpora.Dictionary(tokenized_reviews)
corpus = [dictionary.doc2bow(text) for text in tokenized_reviews]

# LDA model
lda_model = LdaModel(corpus, num_topics=5, id2word=dictionary, passes=15)

# Display topics
for idx, topic in lda_model.print_topics(-1):
    print('Topic: {} \nWords: {}'.format(idx, topic))
```

```
Topic: 0
Words: 0.050*"." + 0.042*"the" + 0.036*"i" + 0.027*"and" + 0.024*"to" + 0.022
*"," + 0.021*"a" + 0.014*"of" + 0.013*"it" + 0.012*"in"
Topic: 1
Words: 0.053*"." + 0.041*"i" + 0.037*"the" + 0.028*"to" + 0.027*"," + 0.025
*"and" + 0.017*"a" + 0.013*"starbucks" + 0.011*"in" + 0.010*"it"
Topic: 2
Words: 0.035*"the" + 0.027*"," + 0.023*"." + 0.021*"i" + 0.018*"a" + 0.017*"t
o" + 0.013*"and" + 0.012*"of" + 0.010*"starbucks" + 0.008*"for"
Topic: 3
Words: 0.031*"," + 0.019*"the" + 0.016*"." + 0.014*"to" + 0.011*"!" + 0.009
*"of" + 0.008*"they" + 0.007*"starbucks" + 0.007*"you" + 0.006*"and"
Topic: 4
Words: 0.053*"." + 0.040*"i" + 0.033*"the" + 0.030*"," + 0.029*"and" + 0.028
*"to" + 0.023*"a" + 0.019*"my" + 0.019*"was" + 0.013*"it"
```

- The topics provided from the LDA (Latent Dirichlet Allocation) output seem to be dominated by common English words (like "the", "and", "to", commas, and periods) rather than more meaningful, topic-specific terms. This is a common issue in topic modeling, especially when dealing with raw text data. These common words are often referred to as "stop words" and typically don't contribute much to the understanding of the text's content.

To improve the quality of the topics generated by LDA, consider the following steps:

1. Remove Stop Words: Filter out common stop words ("the", "and", "to", etc.) from your text data. Many NLP libraries provide lists of stop words.
2. Remove Punctuation: Punctuation marks (like commas and periods) should be removed from the text as they don't contribute to topic modeling.
3. Lemmatization/Stemming: These processes reduce words to their base or root form, helping in consolidating similar words.

Tweak LDA Parameters:

- Adjusting parameters like the number of topics (num_topics), passes (passes), and the number of words per topic can significantly affect the results.

In [90]:

```
import gensim
from gensim import corpora
from gensim.models.ldamodel import LdaModel
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
import string

# Download NLTK stopwords
import nltk
nltk.download('stopwords')

# Set of English stopwords
stop_words = set(stopwords.words('english'))

def preprocess(text):
    # Tokenize and lower case
    tokens = word_tokenize(text.lower())
    # Remove stopwords and punctuation
    tokens = [word for word in tokens if word not in stop_words and word not in
string.punctuation]
    return tokens

# Preprocess and tokenize data
tokenized_reviews = [preprocess(review) for review in starbucks_data['Review']]

# Create a dictionary and corpus
dictionary = corpora.Dictionary(tokenized_reviews)
corpus = [dictionary.doc2bow(text) for text in tokenized_reviews]

# LDA model
lda_model = LdaModel(corpus, num_topics=5, id2word=dictionary, passes=15)

# Display topics
for idx, topic in lda_model.print_topics(-1):
    print('Topic: {} \nWords: {}'.format(idx, topic))
```

```
[nltk_data] Downloading package stopwords to /usr/share/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
Topic: 0
Words: 0.022*"starbucks" + 0.019*"card" + 0.008*"``" + 0.008*"''" + 0.008*"wo
uld" + 0.008*"n't" + 0.007*"one" + 0.007*"said" + 0.007*"customer" + 0.006*"s
tore"
Topic: 1
Words: 0.021*"starbucks" + 0.014*"’" + 0.009*"said" + 0.009*"``" + 0.008*"''"
+ 0.008*"coffee" + 0.008*"one" + 0.008*"order" + 0.007*"get" + 0.007*"n't"
Topic: 2
Words: 0.022*"starbucks" + 0.017*"coffee" + 0.009*"like" + 0.008*"one" + 0.00
8*"'s" + 0.008*"n't" + 0.007*"time" + 0.006*"would" + 0.006*"store" + 0.006
*"order"
Topic: 3
Words: 0.023*"starbucks" + 0.020*"coffee" + 0.010*"n't" + 0.009*"drink" + 0.0
09*"get" + 0.006*"'s" + 0.005*"would" + 0.005*"go" + 0.005*"service" + 0.004
*"cup"
Topic: 4
Words: 0.028*"starbucks" + 0.014*"coffee" + 0.011*"customer" + 0.010*"servic
e" + 0.008*"n't" + 0.007*"store" + 0.007*"drink" + 0.006*"get" + 0.006*"'s" +
0.005*"always"
```

- The updated topics from your LDA model provide a clearer insight into the themes present in the Starbucks reviews. Let's interpret each topic based on the most frequent words:

  # Topic 0 (Experience and Orders):

  - Keywords: "starbucks", "coffee", "drink", "go", "cup", "get", "'s", "ordered", "order", "time" Interpretation: This topic seems to revolve around the customer experience at Starbucks, focusing on ordering and drinking coffee. Words like "go", "cup", "get", "ordered", and "order" suggest a focus on the purchasing process and the products themselves.

  # Topic 1 (Product Quality and Preferences):

  - Keywords: "coffee", "starbucks", "'s", "like", "n't", "store", "caramel", "time", "get", "would" Interpretation: This topic might be related to customers' opinions on the quality and variety of coffee, including specific preferences (like "caramel"). The presence of words like "like", "n't" (don't), and "would" indicates a mix of preferences and opinions.

  # Topic 2 (Service and Customer Experience):

  - Keywords: "starbucks", "'", "card", "coffee", "store", "service", "go", "like", "n't", "customer" Interpretation: The focus here appears to be on the service aspect, including customer service, with mentions of "card" (possibly related to loyalty cards or payments), "store", and "service". The word "customer" suggests a focus on customer experience.

  # Topic 3 (Customer Service and Store Experience):

  - Keywords: "starbucks", "n't", "customer", "get", "coffee", "one", "service", "would", "store", "drink" Interpretation: This topic also relates to customer service and the in-store experience. It includes elements of customer interactions and service quality, with words like "customer", "service", and "store".

  # Topic 4 (Customer Interaction and Complaints):

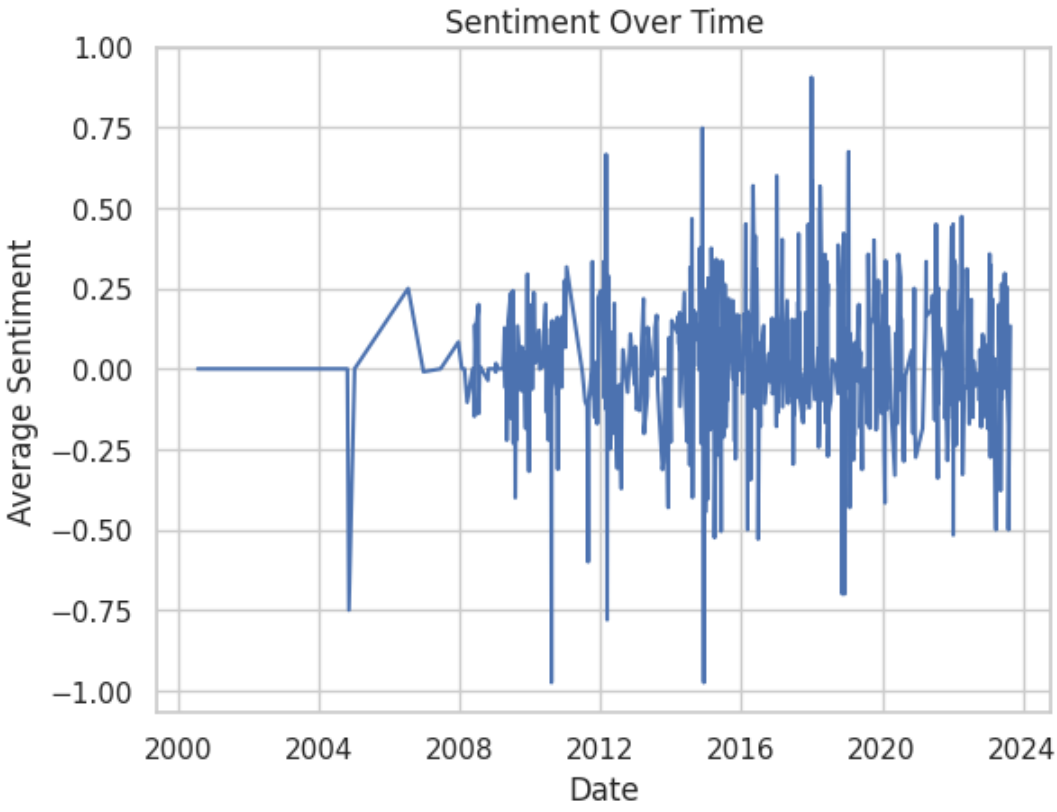
  - Keywords: "starbucks", "customer", "n't", "coffee", "said", "", """, "one", "drink", "time" Interpretation: This topic may involve customer interactions, possibly including complaints or specific incidents (indicated by "said", "", and """). It reflects discussions about customer experiences and interactions at Starbucks locations.

In [91]:

```
def preprocess_date(date_str):
    # Removing the prefix 'Reviewed' and stripping any leading/trailing whitespace
    date_str = date_str.replace('Reviewed', '').strip()

    # Define multiple possible date formats
    date_formats = ['%b. %d, %Y', '%B %d, %Y']

    # Try each format and return the first successful conversion
    for fmt in date_formats:
        try:
            return pd.to_datetime(date_str, format=fmt)
        except ValueError:
            continue

    # If all formats fail, return None or raise an error
    return None

# Preprocess the date column
starbucks_data['Date'] = starbucks_data['Date'].apply(preprocess_date)
```

In [92]:

```
from textblob import TextBlob
import matplotlib.pyplot as plt
import pandas as pd

# Function to get sentiment
def get_sentiment(text):
    return TextBlob(text).sentiment.polarity

# Apply sentiment analysis
starbucks_data['Sentiment'] = starbucks_data['Review'].apply(get_sentiment)

# Group by Date and calculate average sentiment
sentiment_over_time = starbucks_data.groupby(starbucks_data['Date'].dt.date)['Se
ntiment'].mean()

# Plot
sentiment_over_time.plot()
plt.title('Sentiment Over Time')
plt.xlabel('Date')
plt.ylabel('Average Sentiment')
plt.show()
```



Time Span:

- The x-axis represents time, spanning from the year 2000 to just beyond 2024, indicating that the data covers a period of approximately 25 years.

Sentiment Score:

- The y-axis indicates the average sentiment score, which seems to range from -1 to 1. This is a typical range for sentiment scores, where -1 represents a very negative sentiment, 1 represents a very positive sentiment, and 0 represents a neutral sentiment.

Volatility:

- The plot shows a considerable amount of volatility in sentiment over time, with sentiment scores fluctuating frequently between positive and negative values. This could suggest varying customer satisfaction over time or different responses to products or services.

Trend Analysis:

- There is no clear long-term trend upwards or downwards, indicating that there hasn't been a significant long-term improvement or decline in sentiment.

Notable Peaks and Troughs:

- There are noticeable peaks where sentiment reaches particularly high points, and troughs where sentiment becomes especially negative. These could correspond to specific events or changes in the company's offerings or service.

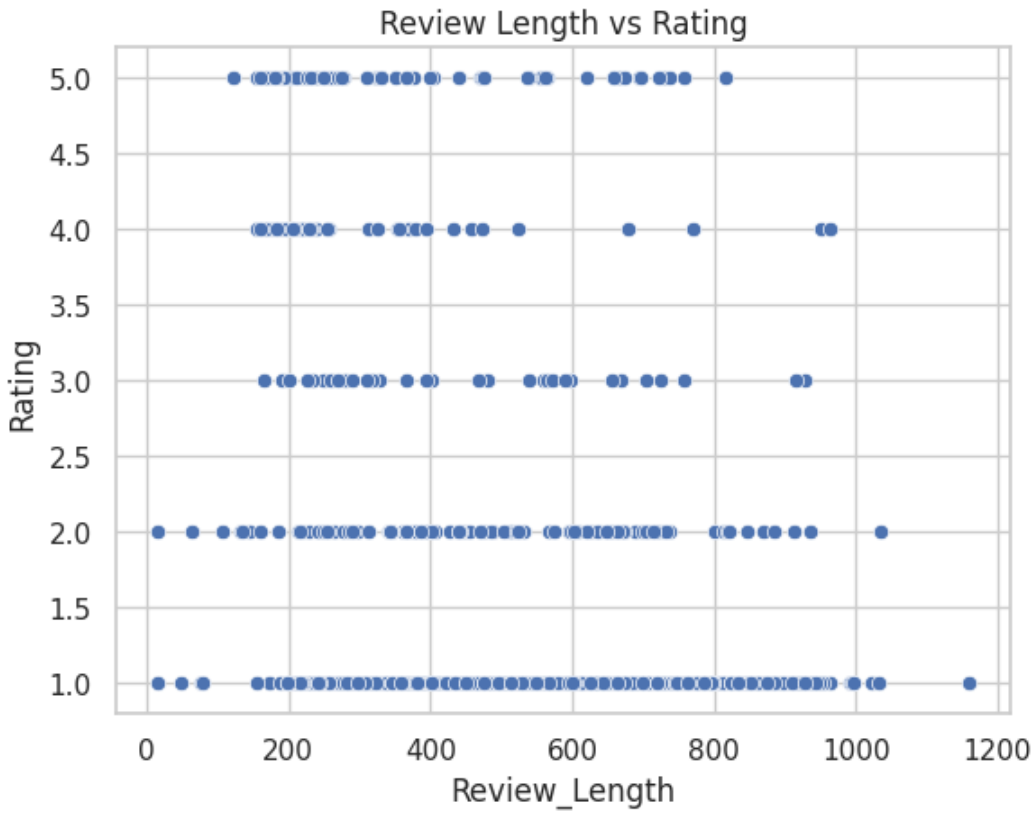
Data Density:

- There are periods where the data points become denser, visible by the thicker clustering of the line. This could indicate more frequent reviews or more consistent sentiment during these periods.

When interpreting this chart, it would be important to consider external factors that could influence sentiment scores, such as marketing campaigns, product launches, social events, or changes in customer service policies. Additionally, the source and preprocessing of the text data, the sentiment analysis method used, and the volume of data points collected on each date can all significantly impact the interpretation of the results.

In [93]:

```
import seaborn as sns

# Calculate review length
starbucks_data['Review_Length'] = starbucks_data['Review'].apply(len)

# Plot correlation between review length and rating
sns.scatterplot(x='Review_Length', y='Rating', data=starbucks_data)
plt.title('Review Length vs Rating')
plt.show()
```



X-Axis (Review Length):

- The horizontal axis represents the length of the reviews, which measured by the number of characters in each review. The review length varies significantly, ranging from very short reviews (possibly as low as around 50 characters) to much longer reviews (upwards of 1200 characters).

Y-Axis (Rating):

- The vertical axis represents the rating associated with each review. The ratings is on a scale from 1 to 5, which is a common rating system for reviews.

Data Distribution:

- There is a cluster of data points across all rating levels with shorter review lengths, suggesting that many users leave shorter reviews regardless of the rating they give. For longer reviews, there appears to be a concentration of points towards the higher ratings (4 and 5). This could imply that customers who leave more detailed feedback tend to give higher ratings, or are more engaged and possibly more satisfied.

Outliers:

- There are a few longer reviews with lower ratings (1 and 2). These might be cases where customers provided detailed negative feedback.

No Clear Trend:

- There doesn't appear to be a clear or strong linear relationship between the length of the review and the rating given. While there is some indication that longer reviews might correlate with higher ratings, the data points are quite dispersed, indicating variability.

Data Density:

- The chart shows a high density of reviews with fewer characters across all ratings, indicating that most customers tend to leave brief reviews.

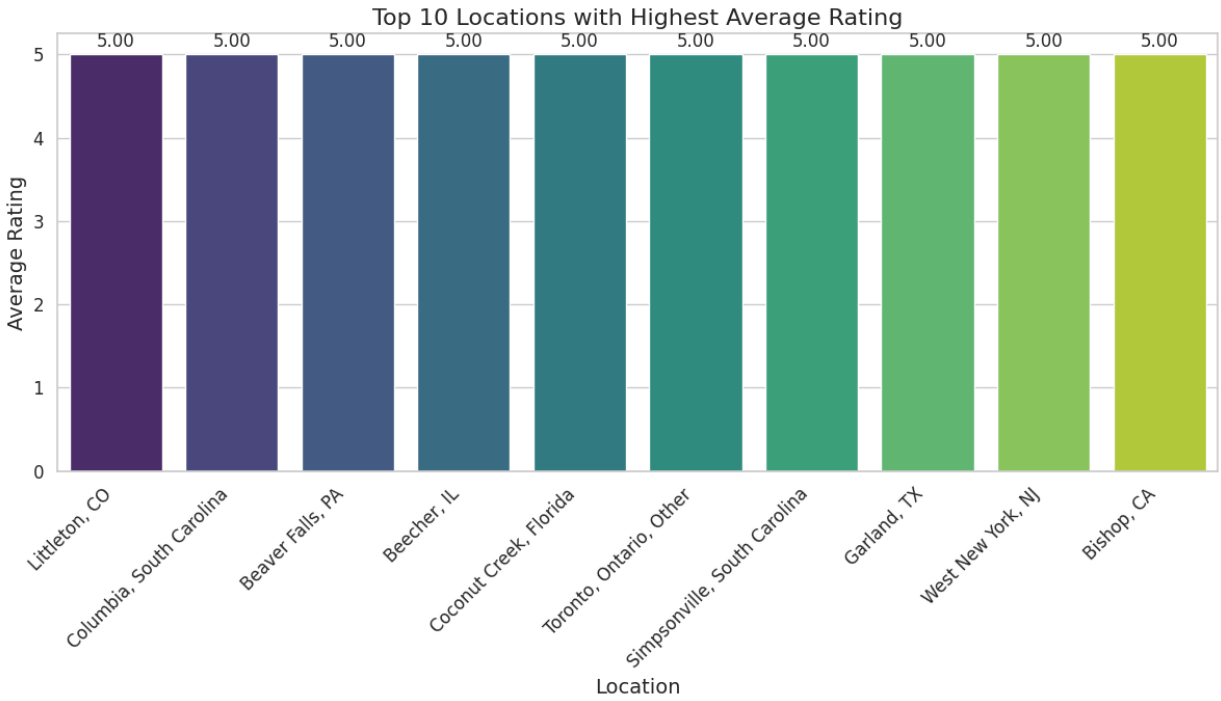
From this scatter plot, it is not possible to definitively conclude that longer reviews correlate with higher ratings, although there is some suggestion of this in the data. For a more detailed analysis, statistical methods could be used to calculate the correlation coefficient or fit a regression line to the data. Additionally, sentiment analysis on the review text could provide further insights into the relationship between the content of the reviews, their length, and the ratings.

Location-Based Analysis

- Analyze the ratings and sentiments by location to see if customer satisfaction varies geographically.

In [94]:

```
import seaborn as sns
import matplotlib.pyplot as plt

# Set the aesthetic style of the plots
sns.set_style("whitegrid")

# Group by Location and calculate average rating
average_rating_per_location = starbucks_data.groupby('location')['Rating'].mean
()

# Sort and take the top 10 locations with the highest average rating
top_locations = average_rating_per_location.sort_values(ascending=False).head(1
0)

# Plot using Seaborn for a more attractive bar chart
plt.figure(figsize=(12, 7))
ax = sns.barplot(x=top_locations.index, y=top_locations.values, palette="viridi
s")

# Add the bar values as annotations
for p in ax.patches:
    ax.annotate(format(p.get_height(), '.2f'),
                (p.get_x() + p.get_width() / 2., p.get_height()),
                ha = 'center', va = 'center',
                xytext = (0, 9),
                textcoords = 'offset points')

# Set plot title and labels with increased font sizes for readability
plt.title('Top 10 Locations with Highest Average Rating', fontsize=16)
plt.xlabel('Location', fontsize=14)
plt.ylabel('Average Rating', fontsize=14)

# Improve readability of x-tick labels
plt.xticks(rotation=45, horizontalalignment='right', fontsize=12)
plt.yticks(fontsize=12)

# Show plot
plt.tight_layout()
plt.show()
```



Topic Analysis by Ratings

- For topic modeling based on high and low ratings, we'll use the Latent Dirichlet Allocation (LDA) model from the Gensim library. We'll first separate the reviews into high-rated and low-rated groups, preprocess the text, and then apply LDA to each group to identify prevalent topics.

In [95]:

```
import gensim
from gensim import corpora
from gensim.models.ldamodel import LdaModel
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
import string

# Function to preprocess text
def preprocess(text):
    stop_words = set(stopwords.words('english') + list(string.punctuation))
    tokens = word_tokenize(text.lower())
    return [word for word in tokens if word not in stop_words and not word.isdig
it()]

# Separate the high-rated and low-rated reviews
high_rated = starbucks_data[starbucks_data['Rating'] >= 4]['Review'].apply(prepr
ocess)
low_rated = starbucks_data[starbucks_data['Rating'] <= 2]['Review'].apply(prepro
cess)

# Create the dictionary and corpus for high-rated reviews
high_dict = corpora.Dictionary(high_rated)
high_corpus = [high_dict.doc2bow(text) for text in high_rated]

# Create the dictionary and corpus for low-rated reviews
low_dict = corpora.Dictionary(low_rated)
low_corpus = [low_dict.doc2bow(text) for text in low_rated]

# LDA model for high-rated reviews
high_lda = LdaModel(high_corpus, num_topics=5, id2word=high_dict, passes=10)
# LDA model for low-rated reviews
low_lda = LdaModel(low_corpus, num_topics=5, id2word=low_dict, passes=10)

# Display topics for high-rated reviews
print("High-Rated Reviews Topics:")
for idx, topic in high_lda.print_topics(-1):
    print('Topic: {} Word: {}'.format(idx, topic))

# Display topics for low-rated reviews
print("\nLow-Rated Reviews Topics:")
for idx, topic in low_lda.print_topics(-1):
    print('Topic: {} Word: {}'.format(idx, topic))
```

```
High-Rated Reviews Topics:
Topic: 0 Word: 0.025*"starbucks" + 0.019*"coffee" + 0.013*"great" + 0.013*"se
rvice" + 0.012*"food" + 0.010*"'s" + 0.009*"good" + 0.009*"like" + 0.008*"go"
+ 0.006*"usually"
Topic: 1 Word: 0.022*"starbucks" + 0.020*"always" + 0.013*"service" + 0.013
*"get" + 0.009*"customer" + 0.009*"great" + 0.009*"store" + 0.008*"go" + 0.00
8*"could" + 0.008*"'s"
Topic: 2 Word: 0.024*"coffee" + 0.023*"starbucks" + 0.015*"great" + 0.014*"se
rvice" + 0.009*"customer" + 0.009*"good" + 0.009*"store" + 0.009*"always" +
0.009*"experience" + 0.007*"friendly"
Topic: 3 Word: 0.023*"starbucks" + 0.023*"coffee" + 0.013*"’" + 0.008*"drink"
+ 0.008*"back" + 0.008*"work" + 0.007*"store" + 0.007*"drinks" + 0.007*"day"
+ 0.007*"one"
Topic: 4 Word: 0.030*"starbucks" + 0.018*"coffee" + 0.016*"always" + 0.011*"g
ood" + 0.010*"drink" + 0.010*"like" + 0.009*"service" + 0.009*"store" + 0.008
*"'s" + 0.007*"love"

Low-Rated Reviews Topics:
Topic: 0 Word: 0.020*"coffee" + 0.014*"starbucks" + 0.009*"n't" + 0.009*"orde
r" + 0.009*"``" + 0.009*"drink" + 0.009*"''" + 0.008*"ordered" + 0.007*"time"
+ 0.006*"said"
Topic: 1 Word: 0.017*"starbucks" + 0.010*"one" + 0.009*"''" + 0.009*"``" + 0.
008*"**" + 0.008*"drink" + 0.007*"store" + 0.006*"get" + 0.006*"like" + 0.006
*"'s"
Topic: 2 Word: 0.034*"starbucks" + 0.015*"coffee" + 0.010*"n't" + 0.009*"get"
+ 0.008*"drink" + 0.008*"customer" + 0.007*"card" + 0.006*"one" + 0.006*"sai
d" + 0.006*"service"
Topic: 3 Word: 0.028*"starbucks" + 0.012*"n't" + 0.010*"customer" + 0.010*"co
ffee" + 0.008*"customers" + 0.008*"service" + 0.007*"store" + 0.007*"like" +
0.005*"'s" + 0.005*"get"
Topic: 4 Word: 0.025*"starbucks" + 0.009*"n't" + 0.008*"'s" + 0.008*"store" +
0.007*"coffee" + 0.007*"card" + 0.007*"would" + 0.007*"said" + 0.007*"custome
r" + 0.007*"went"
```

Frequent Words and N-gram Analysis

- For identifying frequent words and n-grams, you can use the nltk library to tokenize your text and then use collections.Counter to find the most common words and n-grams.

In [96]:

```
from nltk import bigrams
from collections import Counter

# Preprocess the reviews to get tokens
tokens = starbucks_data['Review'].apply(preprocess).sum()

# Get frequencies of individual words
word_freq = Counter(tokens)

# Get frequencies of bi-grams
bi_grams = bigrams(tokens)
bi_gram_freq = Counter(bi_grams)

# Display the 10 most common words
print("Most common words:")
for word, freq in word_freq.most_common(10):
    print(f"{word}: {freq}")

# Display the 10 most common bi-grams
print("\nMost common bi-grams:")
for bi_gram, freq in bi_gram_freq.most_common(10):
    print(f"{bi_gram}: {freq}")
```

```
Most common words:
starbucks: 1051
coffee: 595
n't: 361
customer: 286
one: 279
get: 276
drink: 275
's: 260
store: 259
service: 255

Most common bi-grams:
('customer', 'service'): 144
('said', '``'): 52
('went', 'starbucks'): 45
('ca', "n't"): 40
('review', 'text'): 37
('go', 'starbucks'): 35
('gift', 'card'): 34
('every', 'time'): 31
('starbucks', 'customer'): 29
('first', 'time'): 29
```

In [97]:

```
# The End :) A.O
```