DATA MINING MGT7216

Assignment 2

Word Count: 2200

Muhammad Muneeb Ullah Ansari - 40426685

Contents

Introduction

Text analytics, leveraging statistical natural language processing, machine learning, and natural language processing (NLP), plays a critical role in deriving valuable insights from unstructured text. Manning and Schütze (1999) discuss foundational NLP techniques that analyze text data for various applications, including sentiment analysis as detailed by Pang and Lee (2008), and topic modeling using models like LDA introduced by Blei et al. (2003).

Applications extend across multiple domains. For instance, Hearst (1999) discusses text classification for detecting anomalies, while Fan and Bifet (2013) focus on adapting text mining techniques for large datasets. Nakov et al. (2013) explore the challenges of multilingual text analytics, and Kouloumpis et al. (2011) illustrate real-time sentiment analysis using social media data.

Text analytics also impacts the health sector, where Savova et al. (2010) use NLP tools to analyze medical records. Mihalcea and Tarau (2004) provide methods for academic literature analysis, helping identify significant trends. Ethical considerations in the use of text analytics tools are discussed by Boyd and Crawford (2012), highlighting the importance of regulatory guidelines.

Support Vector Machines (SVMs) are highly effective in semi-supervised learning, particularly due to their ability to maximize margins, a principle highlighted by Vapnik (1998). Joachims (1999) developed Transductive SVMs, which significantly improve performance in emotion prediction from mostly unlabeled textual data. Bennett and Demiriz (1999) extended this by adapting SVMs to scenarios with minimal labeled data but extensive unlabeled data. The kernel trick, essential for processing textual data features, is explored by Schölkopf et al. (1999).

Zhu (2005) noted that SVMs help in capturing the underlying structure of emotional data more effectively. Belkin et al. (2006) demonstrated how SVMs use unlabeled data to adjust the decision boundary based on data distribution, enhancing learning accuracy. Goldberg and Zhu (2006) combined graph-based methods with SVMs to further benefit emotion prediction. Weston et al. (2008) showed that SVMs could adapt to various data modalities, expanding their application range.

Recent developments integrate SVMs with deep neural networks to leverage both representational learning and classification capabilities, as discussed by Deng et al. (2012). Additionally, Wang and Manning (2012) confirmed that SVM-based models excel over other classifiers in semi-supervised settings.

Methodology

There were many comments with letters not belonging in the English dictionary (like ÃÂ£ÃÆÃÂ£Ã¯Â¾Ã). This could mean that the data of the text reviews was corrupted. So the data was cleaned to only keep observations with text reviews containing English alphabets, numbers, and punctuations. Then further processing resulted in removal of numbers and punctuations from the text reviews of observations. Following were also some of the preprocessing done:

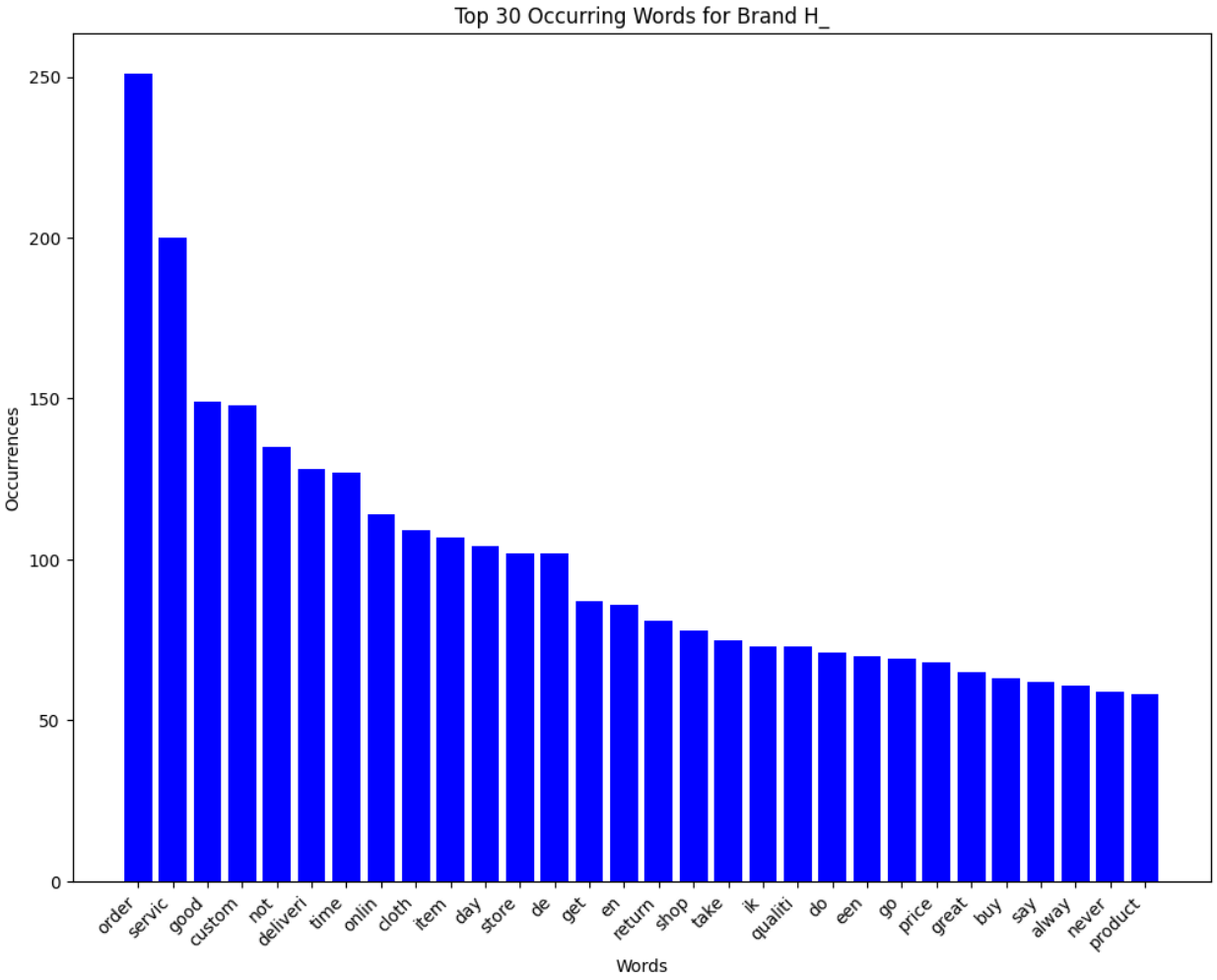
- Removing HTML tags, punctuation, and special characters
- Converting text to lowercase
- Tokenization (Done on a word-level so formation of Bag of Words would be easier)
- Stop word removal
- Stemming
- Lemmatization

Bag of word formulation also displayed words that had English alphabets but the words were not English (Like zufriedenheit, zufriedenkann, zutiefst etc.) and words that started with a number (like 11^{th}, 12^{th} etc.). Words present in a foreign language could have given us a deeper understanding if the analysis did not focus on English language only. Hence these words were entirely removed. Note that only the words were removed and not the entire observation or text review that these words belonged to so some context was saved.

Out of all the observations when corrupted comments were cleaned, we were only left with 1443 rows out of 5722, hence data quality in this assignment was a significant issue. After bag of words were created, 4899 columns represented that many unique words. However when words present only in the English language were kept and excluding those starting with a number, we were left with only 1512 unique words. In those words present in the English dictionary, there were still words that had no meaning and were not classified as stop words (like en, de, do, het etc.). Only a total of 627 rows out of 5722 had emotions not missing. This could have been the reason why semi-supervised learning for predicting missing emotions had a low accuracy.

Results

The following are plots that represents word counts in both respective brands:



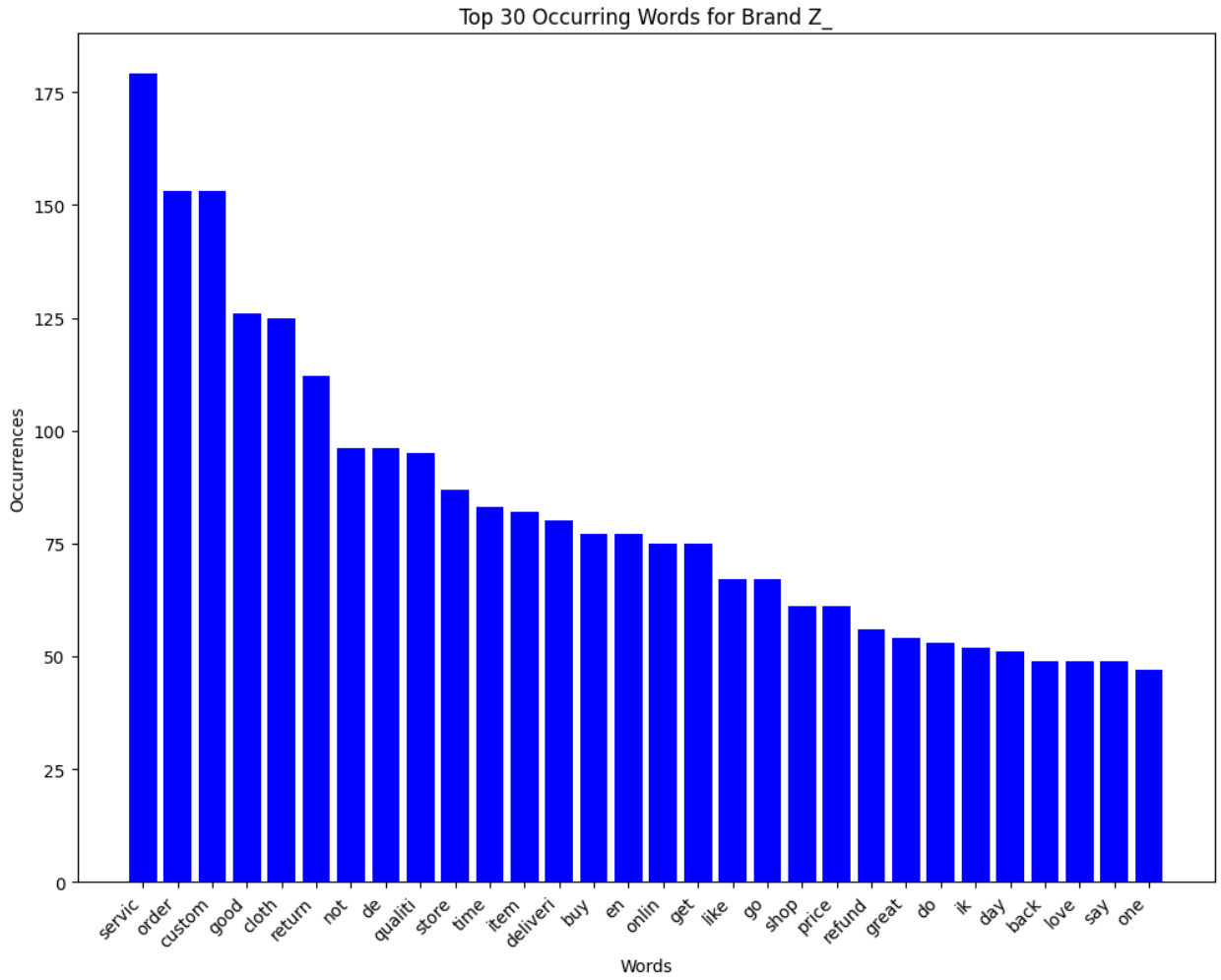
The word "order" appears most frequently, with a count nearing 250. This suggests that much of the discussion or content related to Brand H_ revolves around the ordering process. This could imply that customers frequently talk about placing orders, which might be a central aspect of their interaction with the brand.

Words like "service," "delivery," and "return" are also very prominent. This indicates that customer service aspects such as the quality of service, delivery times, and return policies are significant topics. These elements are crucial for customer satisfaction and often discussed in reviews.

Descriptive words like "quality" and "great" suggest that reviews or discussions often focus on evaluating the quality of the products or services. The frequent use of "great" points to generally positive sentiment.

The presence of words like "buy" and "price" highlights discussions centered around purchasing decisions and considerations of cost. This could reflect the importance of pricing in customer discussions or queries about value for money.

The word "never" appears towards the end of the list, which might indicate some negative experiences or emphatic statements regarding expectations not being met ("never buy," "never delivered on time").



For Brand Z_, there seems to be a higher frequency of words like "not", "like", and "bad" compared to Brand H_, which might suggest more mixed or negative sentiments in the customer feedback or more comparisons and expectations expressed.

Brand Z_ includes words such as "phone", "shop", and "online", which could indicate a focus on both online and physical retail aspects, or issues related to these areas.

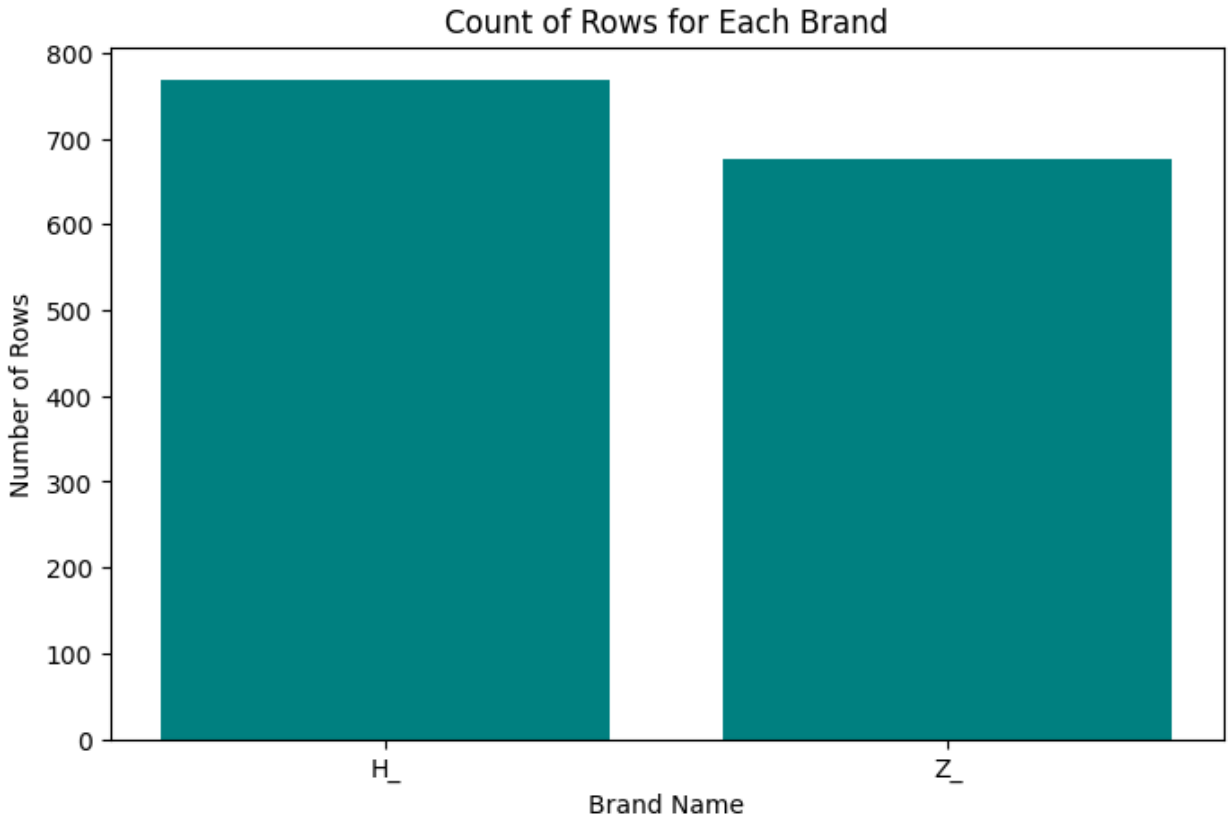
The decline in word frequency from the most common word to the 30th is quite steep for Brand Z_, suggesting a few topics dominate the discussion more distinctly than for Brand H_, where the distribution was slightly more even.

Brand Z_ includes interaction-focused words like "get", "buy", "back", and "say" which might indicate more active discussions around purchasing decisions and customer service interactions. Creating topic models could have given us more insights into the context of each word occurring.

Focusing on sentiments, the following were the polarities:

- Brand Z: 0.174
- Brand H: 0.156
- Overall: 0.164

The following is the count of observations for each brand:



Despite Brand Z having more negative words, it shows a slightly more positive sentiment. Brand Z, with 675 mentions, has fewer mentions than Brand H, suggesting possibly a smaller customer base or less marketing activity, though the difference is minor, showing both brands have significant engagement. Brand H, with 768 observations, likely has higher sales or more active marketing due to its higher customer interaction. Brand Z's modestly positive polarity score indicates generally favorable but not exceptionally positive customer experiences. Brand H shows similar, yet slightly fewer positive sentiments. For emotion prediction in semi-supervised learning, we used the label spreading algorithm, and the model predictions were later added to the original dataset for more analysis.

The following is the evaluation of the semi-supervised model to predict missing emotions in the dataset:

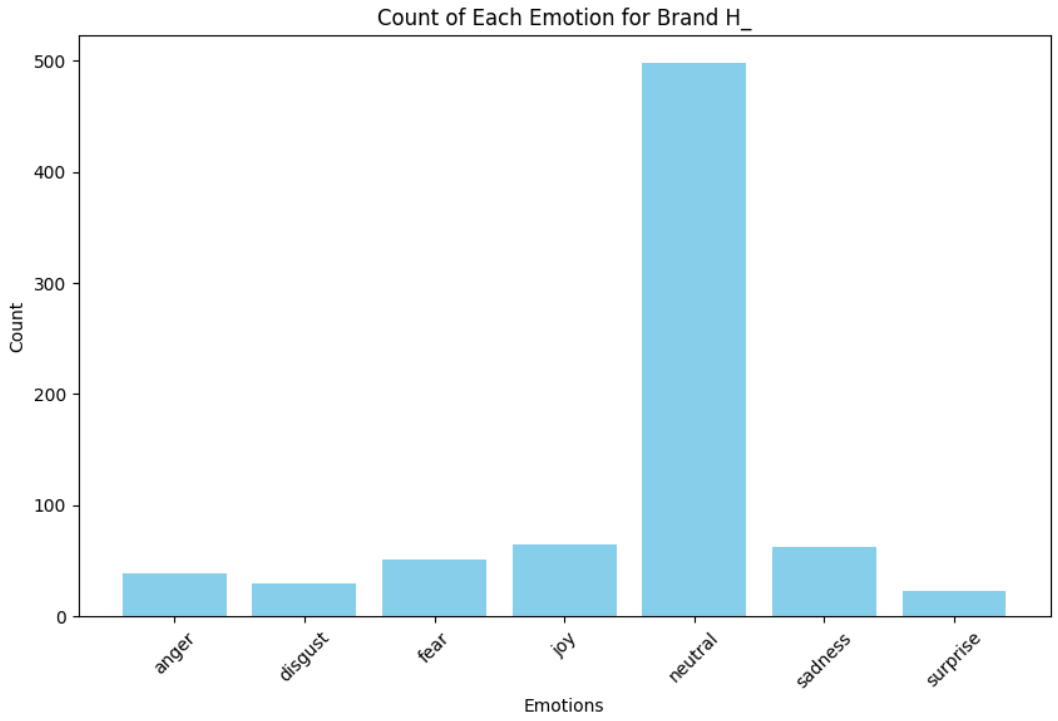
```
Label  Classifier on the labeled and unlabeled data:
Number of training samples: 1387
Unlabeled samples in training set: 1165
Micro-averaged F1 score on test set: 0.429

Confusion Matrix:
 [[2 0 1 0 2 0 0]
 [0 0 0 1 2 2 0]
 [0 0 4 0 3 0 0]
 [0 0 0 8 2 2 0]
 [0 1 1 2 7 1 0]
 [1 0 2 0 3 3 0]
 [0 0 0 1 5 0 0]]

Classification Report:
              precision    recall  f1-score   support

           1       0.67      0.40      0.50         5
           2       0.00      0.00      0.00         5
           3       0.50      0.57      0.53         7
           4       0.67      0.67      0.67        12
           5       0.29      0.58      0.39        12
           6       0.38      0.33      0.35         9
           7       1.00      0.00      0.00         6

    accuracy                           0.43        56
   macro avg       0.50      0.36      0.35        56
weighted avg       0.49      0.43      0.39        56
```

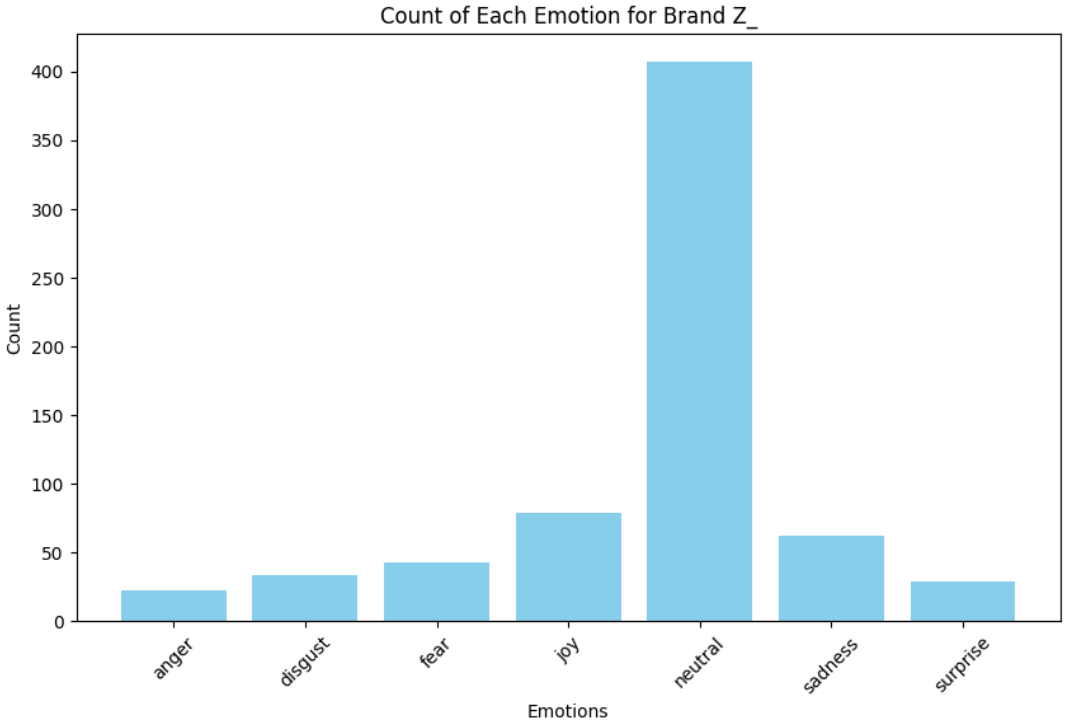
The model is working on 1,387 observations which includes both labeled and unlabeled samples. This is a relatively good sample size for training machine learning models, particularly for semi-supervised learning. Unlabeled samples in training set is 1,165 indicating a significant portion of the data did not initially have labels.

An F1 score of around 0.43 suggests moderate performance, indicating there's room for improvement, particularly in making the model more sensitive and precise. The confusion matrix shows several off-diagonal elements, indicating several misclassifications. For example, emotion 1 is sometimes confused with emotions 3 and 4, among others. Some emotions like 3 and 4 seem to be predicted with more frequency (higher false positives and true positives), indicating potential biases or overfitting to these labels in the dataset.

Emotion 1 has a precision of 0.67 but a recall of 0.40, suggesting while the predictions that were made as emotion 1 are fairly accurate, the model is missing a lot of actual cases of emotion 1. Emotion 7 has a precision of 1.00 but a recall of 0.00, indicating that while the predictions are very precise, the model fails to identify any true cases of emotion 7 (likely no predictions were made for this class).

The following are the counts of total emotions, emotions for brand H, emotions for brand z respectively:





In all three plots, the emotion "neutral" has the highest count, indicating that the majority of the sentiments expressed are neither extremely positive nor negative. This could suggest that most customer interactions or reviews tend to be factual or tempered in emotion, possibly focusing on informational content rather than emotional expression.

In brand h, "Joy" and "sadness" follow, but with much lower frequencies. This could indicate that while there are positive and negative experiences associated with Brand H_, they are not as pronounced. Notably, "joy" appears to be less common in Brand Z_ compared to Brand H_, suggesting that positive experiences or sentiments might be less frequent with Brand Z_. "Sadness" and "anger" are somewhat more pronounced in Brand Z_ than in Brand H_, which could be a point of concern, indicating potential areas where customer expectations may not be fully met.

Both brands show a similar range of emotions but with different proportions. Brand H_ seems to have a slightly better emotional profile with higher "joy" and lower "sadness" and "anger" compared to Brand Z_. Still from a sentiment polarity perspective, Brand Z_ scores higher than Brand H_.

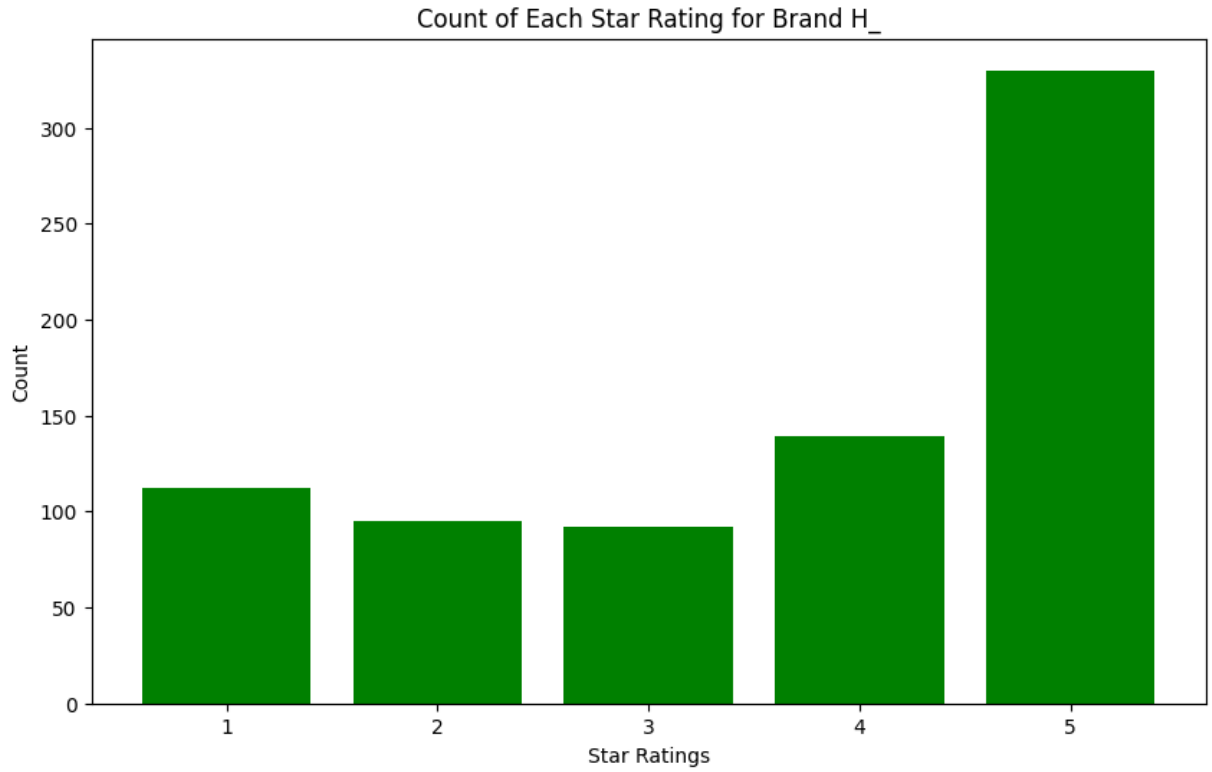
For creating a model that predicts the sentiment or polarity, we opted for the support vector classifier as our supervised technique.

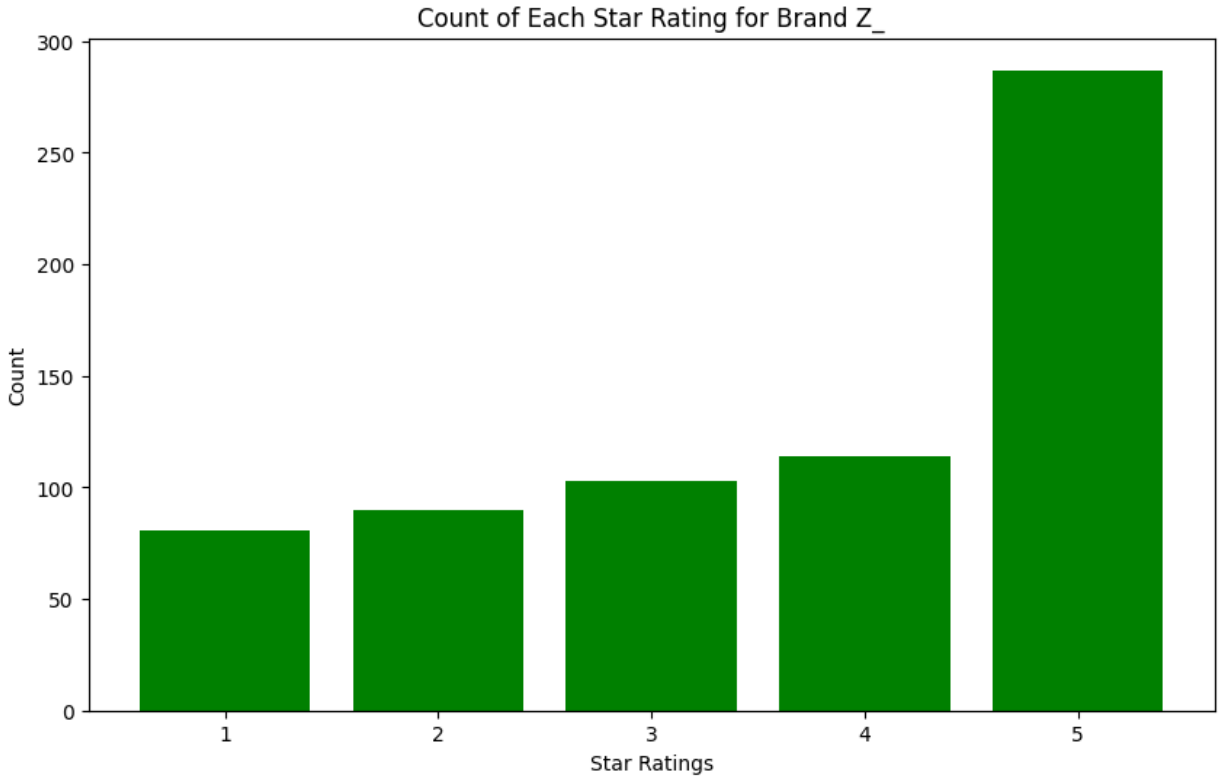
The following is the accuracy evaluation:

```
Confusion Matrix:
 [[ 34   8  19]
 [  1  82   8]
 [  7   8 122]]

Accuracy: 0.8235294117647058

Classification Report:
              precision    recall  f1-score   support

    Negative       0.81      0.56      0.66        61
     Neutral       0.84      0.90      0.87        91
    Positive       0.82      0.89      0.85       137

    accuracy                           0.82       289
   macro avg       0.82      0.78      0.79       289
weighted avg       0.82      0.82      0.82       289
```

The model shows strong overall accuracy of 82.4% and excellent performance in identifying 'Neutral' with a precision of 0.84 and 'Positive' sentiments with a precision of 0.82. This suggests a low rate of false positives. Lower recall of 0.56 for 'Negative' of sentiment indicates missed negative instances. The relatively lower F1-score of 0.66 for 'Negative' sentiment might suggest a need to better differentiate negative sentiment cues or adjust class balance handling.

The following is count of star ratings for Brand H and for Brand Z respectively:





Both brands have a strong showing at the 5-star level, indicating effective customer satisfaction management. However, Brand H_ has a slightly better distribution towards the higher end, which may suggest superior product or service quality, or more effective customer service practices. The presence of 1-star and 2-star ratings, while lower, still exists and warrants attention for both brands. These ratings can be insightful for identifying areas needing improvement such as product issues, customer service, or other operational aspects. 3-star ratings, representing neutral experiences, are moderate for both brands. These ratings are crucial as they can represent a tipping point where improvements could convert neutral customers into satisfied ones.

Limitations

A significant number of comments in the dataset were corrupted, impacting the integrity and usability of the data. Poor data quality can lead to inaccurate model training, skewed analysis, and unreliable outcomes, which in turn affect the model's ability to make accurate predictions. The data at collection could have been reviewed to ensure that the data quality issues are addressed at the source. This could involve setting stricter validation rules on data entry points or revising how data is gathered and stored.

Only 627 out of 5722 rows had non-missing emotion labels, which restricts the training capacity of supervised learning components in semi-supervised models. The lack of sufficient labeled data can lead to lower predictive accuracy and hinder the model's ability to learn complex patterns in the data. The volume of labeled data could have been increased either through manual labeling efforts or by employing crowdsourcing techniques. This approach would provide more examples for the model to learn from, thereby potentially increasing its accuracy

Topic models could have provided deeper insights into the context and thematic structure of the text data, which could enhance the interpretability and granularity of sentiment analysis. Latent Dirichlet Allocation (LDA) or Non-negative Matrix Factorization (NMF) could have been used to identify underlying topics within the text data. This can help in categorizing feedback into distinct themes for more targeted analysis and improved understanding of customer sentiments.

Conclusion

Understanding which words occur most frequently in discussions about the brand can help identify what matters most to customers, whether it is the service, the quality of the product, or the efficiency of the delivery. Identifying frequent mentions of terms related to service and processes (like "return" and "delivery") might pinpoint areas needing improvement. Knowing that words like "quality" and "great" are common can guide the marketing strategy to highlight these aspects more prominently. The presence of words with potentially negative connotations ("not", "bad") for Brand Z_ could be crucial for sentiment analysis, indicating areas that may require attention or improvement. The frequent mention of "return" for both brands points to returns being a significant aspect of customer experience, possibly indicating dissatisfaction or an area where expectations are not met consistently. The usage of more direct-action words in Brand Z_ suggests possibly higher engagement in discussions around service interactions, which could be seen in both positive and negative lights.

References

Bennett, K.P. and Demiriz, A., 1999. Semi-supervised support vector machines. Neural Information Processing Systems.

Belkin, M., Niyogi, P., and Sindhwani, V., 2006. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. Journal of Machine Learning Research, 7(Nov), pp.2399-2434.

Deng, L., Li, J., Huang, J.-T., Yao, K., Yu, D., Seide, F., Seltzer, M., Zweig, G., He, X., Williams, J., Gong, Y. and Acero, A., 2012. Recent advances in deep learning for speech research at Microsoft. International Conference on Acoustics, Speech, and Signal Processing.

Goldberg, A.B. and Zhu, X., 2006. Seeing stars when there aren't many stars: Graph-based semi-supervised learning for sentiment categorization. Proceedings of the First Workshop on Graph Based Methods for Natural Language Processing.

Joachims, T., 1999. Transductive inference for text classification using support vector machines. International Conference on Machine Learning.

Schölkopf, B., Burges, C.J.C., and Smola, A.J., 1999. Advances in kernel methods: Support vector learning. MIT Press.

Vapnik, V., 1998. Statistical Learning Theory. Wiley.

Wang, S. and Manning, C.D., 2012. Baselines and bigrams: Simple, good sentiment and topic classification. Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics.

Weston, J., Ratle, F., Mobahi, H., and Collobert, R., 2008. Deep learning via semi-supervised embedding. Neural Networks: Tricks of the Trade, Springer.

Zhu, X., 2005. Semi-supervised learning literature survey. Computer Sciences, University of Wisconsin-Madison.

Blei, D.M., Ng, A.Y. and Jordan, M.I., 2003. Latent dirichlet allocation. Journal of Machine Learning Research, 3(Jan), pp.993-1022.

Boyd, D. and Crawford, K., 2012. Critical questions for big data: Provocations for a cultural, technological, and scholarly phenomenon. Information, Communication & Society, 15(5), pp.662-679.

Fan, W. and Bifet, A., 2013. Mining big data: current status, and forecast to the future. ACM sIGKDD Explorations Newsletter, 14(2), pp.1-5.

Hearst, M.A., 1999. Untangling text data mining. Proceedings of the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistics.

Kouloumpis, E., Wilson, T. and Moore, J., 2011. Twitter sentiment analysis: The good the bad and the OMG! Fifth International AAAI Conference on Weblogs and Social Media.

Manning, C.D. and Schütze, H., 1999. Foundations of statistical natural language processing. MIT press.

Mihalcea, R. and Tarau, P., 2004. TextRank: Bringing order into text. Proceedings of the 2004 conference on empirical methods in natural language processing.

Nakov, P., Ritter, A., Rosenthal, S., Sebastiani, F. and Stoyanov, V., 2013. SemEval-2013 task 4: Free paraphrase identification. Second Joint Conference on Lexical and Computational Semantics.

Pang, B. and Lee, L., 2008. Opinion mining and sentiment analysis. Foundations and Trends in Information Retrieval, 2(1-2), pp.1-135.

Savova, G.K., Masanz, J.J., Ogren, P.V., Zheng, J., Sohn, S., Kipper-Schuler, K.C. and Chute, C.G., 2010. Mayo clinical Text Analysis and Knowledge Extraction System (cTAKES): architecture, component evaluation and applications. Journal of the American Medical Informatics Association, 17(5), pp.507-513.

Appendix

```
#Load file as a pandas dataframe


from google.colab import files


uploaded = files.upload()


import pandas as pd

df = pd.read_excel('A_II_Emotion_Data_Student_Copy_Final.xlsx')

print(df.head())


#Cleaning corrupted comments


# Use a regular expression to match rows that contain only English letters, digits, spaces, and selected punctuation

mask = df['text_reviews_'].str.match(r'^[A-Za-z0-9\s.,?!\'"-]*$')


# Apply the mask to filter the DataFrame

df = df[mask]


# Display the filtered DataFrame to verify that rows with non-English characters have been removed while keeping numerics

print(df)


#Removing HTML tags, punctuation, and special characters.

#Converting text to lowercase.


import re
```

```
import string
import pandas as pd

def clean_text(text):
    # Remove HTML tags
    text = re.sub(r'<.*?>', '', text)
    # Remove punctuation and special characters
    text = ''.join([char for char in text if char not in string.punctuation])
    # Convert text to lowercase
    text = text.lower()
    return text

# Apply the cleaning function to each row
df['cleaned_reviews'] = df['text_reviews_'].apply(clean_text)

# Display the DataFrame to verify changes
print(df[['text_reviews_', 'cleaned_reviews']].head())



#Word Level Tokenization

import nltk
nltk.download('punkt')
from nltk.tokenize import word_tokenize

def tokenize_text(text):
    tokens = word_tokenize(text)
    return tokens
df['tokenized_reviews'] = df['cleaned_reviews'].apply(tokenize_text)
```

```
# Display the DataFrame to verify changes
print(df[['cleaned_reviews', 'tokenized_reviews']].head())



#Stop Word Removal

from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

import nltk
from nltk.corpus import stopwords

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

def remove_stopwords(tokens):
    stop_words = set(stopwords.words('english'))
    filtered_tokens = [word for word in tokens if word.lower() not in stop_words]
    filtered_text = ' '.join(filtered_tokens)
    return filtered_text

df['no_stopwords'] = df['tokenized_reviews'].apply(remove_stopwords)

print(df[['tokenized_reviews', 'no_stopwords']].head())



#Stemming Words
```

```
from nltk.stem import PorterStemmer

def stem_text(text):
    stemmer = PorterStemmer()
    words = word_tokenize(text)

    stemmed_words = [stemmer.stem(word) for word in words]

    stemmed_text = ' '.join(stemmed_words)
    return stemmed_text

df['stemmed_text'] = df['no_stopwords'].apply(stem_text)

# Display the DataFrame to verify changes
print(df[['no_stopwords', 'stemmed_text']].head())


#Lemitization

import spacy

# Load the spaCy model
nlp = spacy.load("en_core_web_sm")
```

```
def lemmatize_text(text):
    # Create a spaCy document object
    doc = nlp(text)
    # Extract the lemmatized form of each word
    lemmatized_text = ' '.join([token.lemma_ for token in doc])
    return lemmatized_text

# Apply the lemmatize_text function to each row in the 'stemmed_text' column
df['lemmatized_text'] = df['stemmed_text'].apply(lemmatize_text)

# Display the DataFrame to verify changes
print(df[['stemmed_text', 'lemmatized_text']].head())

print(df)

#Bag of Words

from sklearn.feature_extraction.text import CountVectorizer

print(df['lemmatized_text'])



# Initialize the CountVectorizer
vectorizer = CountVectorizer()

# Fit and transform the 'lemmatized_text' data to a bag of words representation
X = vectorizer.fit_transform(df['lemmatized_text'])

# Get the feature names, which corresponds to the vocabulary
```

```
feature_names = vectorizer.get_feature_names_out()
print(feature_names)

# Convert the sparse matrix to a dense matrix
X_dense = X.toarray()
print(X_dense)



# Create a new DataFrame with the BoW features
bow_df = pd.DataFrame(X_dense, columns=feature_names)

# Display the new DataFrame with the bag of words representation
print(bow_df)



#Removing columns that have numeric names

# Filter out columns where the column name represents a numeric value
bow_df = bow_df.loc[:, ~bow_df.columns.str.isnumeric()]

bow_df = bow_df.loc[:, ~bow_df.columns.to_series().apply(lambda x: x.replace('.', '', 1).isdigit())]

# Display the DataFrame to verify that numeric columns have been removed
print(bow_df)



#Removing columns that have names starting with numeric

bow_df = bow_df.loc[:, ~bow_df.columns.str.match(r'^\d+.*')]
```

```
print(bow_df)


#Keeping only columns with column names present in the english dictionary

nltk.download('words')

# Load the set of English words from NLTK
from nltk.corpus import words
english_words = set(words.words())

# Filter columns based on whether their names are in the English words set
bow_df = bow_df.loc[:, bow_df.columns.isin(english_words)]

# Display the DataFrame to verify that non-English word columns have been removed
print(bow_df)


#Word Cloud

# Calculate the sum of each column to get the count of occurrences
word_counts = bow_df.sum()

# Create a DataFrame with words and their occurrences
word_occurrences = pd.DataFrame({
    'Word': word_counts.index,
    'Occurrences': word_counts.values
})
```

```
# Sort the DataFrame by occurrences in descending order
word_occurrences_sorted = word_occurrences.sort_values(by='Occurrences', ascending=False)

# Reset the index for a clean table
word_occurrences_sorted.reset_index(drop=True, inplace=True)

# Display the DataFrame
print(word_occurrences_sorted)



#Top occuring words

import matplotlib.pyplot as plt

# Select the top 30 words
top_words = word_occurrences_sorted.head(30)

# Create a bar plot
plt.figure(figsize=(10, 8))  # Set the figure size
plt.bar(top_words['Word'], top_words['Occurrences'], color='blue')  # Create a bar plot

# Add title and labels
plt.title('Top 30 Occurring Words')
plt.xlabel('Words')
plt.ylabel('Occurrences')

# Rotate the x-axis labels for better readability
plt.xticks(rotation=45, ha="right")
```

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



#Sentiment Analysis

from textblob import TextBlob

def analyze_sentiment(text):
    blob = TextBlob(text)  # Create a TextBlob object
    polarity = blob.sentiment.polarity  # Get the polarity score
    if polarity > 0:
        sentiment = "Positive"
    elif polarity < 0:
        sentiment = "Negative"
    else:
        sentiment = "Neutral"
    return sentiment, polarity

# Apply the analyze_sentiment function to each review in the DataFrame
df['Sentiment'], df['Polarity'] = zip(*df['text_reviews_'].apply(analyze_sentiment))

# Display the DataFrame to verify the sentiment and polarity columns
print(df)

# Calculate the average of the 'Polarity' column

print("Average Polarity:", df['Polarity'].mean())
```

```
#Semi-supervised learning for emotion prediction

import pandas as pd
import numpy as np
import re

from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import f1_score
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import FunctionTransformer
from sklearn.semi_supervised import LabelSpreading
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report

unlabeled_data = df[df['emotions_'].isna()][['lemmatized_text']]
unlabeled_data['emotions_'] = -1
print(unlabeled_data)

# Define labeled data as data where "emotions_" is not missing
labeled_data = df[df['emotions_'].notna() & (df['emotions_'] != 'NaN')]
# Extract labels from labeled_data
y_labeled = labeled_data['emotions_']
y_unlabeled = unlabeled_data['emotions_']
X_labeled = labeled_data['lemmatized_text']
X_unlabeled = unlabeled_data['lemmatized_text']
```

```
print("y_labeled ",y_labeled )
print("y_unlabeled",y_unlabeled)
print("X_unlabeled",X_unlabeled)
print("X_labeled",X_labeled)
print(df)

# Parameters
sdg_params = dict(alpha=1e-5, penalty="l2", loss="log_loss")
vectorizer_params = dict(ngram_range=(1, 2), min_df=1, max_df=0.8)

# Supervised Pipeline
pipeline = Pipeline(
    [
        ("vect", CountVectorizer(**vectorizer_params)),
        ("tfidf", TfidfTransformer()),
        ("clf", SGDClassifier(**sdg_params)),
    ]
)

# LabelSpreading Pipeline
ls_pipeline = Pipeline(
    [
        ("vect", CountVectorizer(**vectorizer_params)),
        ("tfidf", TfidfTransformer()),
        ("toarray", FunctionTransformer(lambda x: x.toarray())),
        ("clf", LabelSpreading(kernel='rbf', gamma=20, n_neighbors=7, alpha=0.2, max_iter=30, tol=0.001,
n_jobs=None)),
    ]
)
```

```
def eval_and_print_metrics(clf, X_train, y_train, X_test, y_test):
    print("Number of training samples:", len(X_train))
    print("Unlabeled samples in training set:", sum(1 for x in y_train if x == -1)) #if x == 'NaN'
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)

    print(
        "Micro-averaged F1 score on test set: %0.3f"
        % f1_score(y_test, y_pred, average="micro")
    )
    print("\nConfusion Matrix:\n", confusion_matrix(y_test,y_pred))
    print("\nClassification Report:\n", classification_report(y_test, y_pred,zero_division=1))
    print("\n\n")

X_train, X_test, y_train, y_test = train_test_split(X_labeled, y_labeled, test_size=0.2, stratify=y_labeled,
random_state=42)

print("Supervised SGDClassifier on the labeled data:")
eval_and_print_metrics(pipeline, X_train, y_train, X_test, y_test)

print("Label Spreading on the labeled data:")
eval_and_print_metrics(ls_pipeline, X_train, y_train, X_test, y_test)

test_indices = X_test.index

# Exclude test data from X_labeled and y_labeled based on the identified indices
X_labeled_filtered = X_labeled.drop(index=test_indices, errors='ignore')
y_labeled_filtered = y_labeled.drop(index=test_indices, errors='ignore')
```

```
# Concatenate the filtered labeled data with the unlabeled data
X=X_combined = pd.concat([X_labeled_filtered, X_unlabeled])
y=y_combined = pd.concat([y_labeled_filtered, y_unlabeled])

# Define the mapping for labels
label_mapping = {'anger': 1, 'disgust': 2, 'fear': 3, 'joy': 4, 'neutral': 5, 'sadness': 6, "surprise": 7,-1:-1 }

# Apply the mapping to labels
y  = [label_mapping[label] for label in y]
y_test  = [label_mapping[label] for label in y_test]


print("Label  Classifier on the labeled and unlabeled data:")
eval_and_print_metrics(ls_pipeline, X, y, X_test, y_test)

print(df)

#Imputing predicted emotions
predicted_emotions = ls_pipeline.predict(X_unlabeled)
print(predicted_emotions)

# Add predicted labels to the unlabeled data DataFrame
unlabeled_data['emotions_'] = predicted_emotions

combined_data = pd.concat([labeled_data, unlabeled_data])
print(combined_data)

# Replace the 'emotions_' column in the original DataFrame
```

```
df['emotions_'] = combined_data['emotions_']

# Check for any remaining NaNs in 'emotions_'
print(df['emotions_'].isna().sum())

print(df.head())



#Changing the original labels in emotions_ to numeric mapping

emotion_mapping = {
    'anger': 1,
    'disgust': 2,
    'fear': 3,
    'joy': 4,
    'neutral': 5,
    'sadness': 6,
    'surprise': 7
}
df['emotions_'] = df['emotions_'].replace(emotion_mapping)

print(df)

#Count of Emotions

# Count the occurrences of each emotion and sort them
emotion_counts = df['emotions_'].value_counts().sort_index()

emotion_labels = {
```

```
    1: 'anger',
    2: 'disgust',
    3: 'fear',
    4: 'joy',
    5: 'neutral',
    6: 'sadness',
    7: 'surprise'
}

# Set the figure size
plt.figure(figsize=(10, 6))

# Create a bar plot
plt.bar(emotion_counts.index.map(emotion_labels), emotion_counts.values, color='skyblue')

# Add labels and title
plt.xlabel('Emotions')
plt.ylabel('Count')
plt.title('Count of Each Emotion')

# Rotate the x-axis labels for better readability
plt.xticks(rotation=45)

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

```
#Supervised Machine Learning to predict sentiment

X_train2, X_test2, y_train2, y_test2 = train_test_split(df['lemmatized_text'], df['Sentiment'],
test_size=0.2, random_state=42)

from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_vectorizer = TfidfVectorizer()

X_train_tfidf = tfidf_vectorizer.fit_transform(X_train2)

X_test_tfidf = tfidf_vectorizer.transform(X_test2)

#SUPERVISED LEARNING, Support Vector Classifier

from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.model_selection import train_test_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.naive_bayes import MultinomialNB

from sklearn.svm import SVC

from sklearn.linear_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import confusion_matrix, accuracy_score, classification_report

clf_SVC = SVC(kernel='linear')

clf_SVC.fit(X_train_tfidf, y_train2)

y_pred2 = clf_SVC.predict(X_test_tfidf) # Predict on the test set

print("\nConfusion Matrix:\n", confusion_matrix(y_test2, y_pred2))

print("\nAccuracy:", accuracy_score(y_test2, y_pred2))

print("\nClassification Report:\n", classification_report(y_test2, y_pred2))

#Visualizations for Brand H
```

```
# Filter the DataFrame for brand H_
df_h = df[df['brand_name_'] == 'H_']

# Count the occurrences of each emotion for brand H_
emotion_counts_h = df_h['emotions_'].value_counts().sort_index()

# Create a bar plot for emotions
plt.figure(figsize=(10, 6))
plt.bar(emotion_counts_h.index.map(emotion_labels), emotion_counts_h.values, color='skyblue')  # Assuming emotion_labels mapping exists
plt.xlabel('Emotions')
plt.ylabel('Count')
plt.title('Count of Each Emotion for Brand H_')
plt.xticks(rotation=45)
plt.show()

# Count the occurrences of each star rating for brand H_
star_counts_h = df_h['star_rating_'].value_counts().sort_index()

# Create a bar plot for star ratings
plt.figure(figsize=(10, 6))
plt.bar(star_counts_h.index, star_counts_h.values, color='green')
plt.xlabel('Star Ratings')
plt.ylabel('Count')
plt.title('Count of Each Star Rating for Brand H_')
plt.xticks(rotation=0)  # Star ratings are usually better viewed without rotation
plt.show()
```

```
# Calculate the average polarity for brand H_
average_polarity_h = df_h['Polarity'].mean()

# Print the average polarity
print("Average Polarity for Brand H_:", average_polarity_h)



# Initialize the vectorizer
vectorizer = CountVectorizer()

# Fit and transform the text data
X = vectorizer.fit_transform(df_h['lemmatized_text'])

# Convert to a DataFrame
word_counts_df = pd.DataFrame(X.toarray(), columns=vectorizer.get_feature_names_out())

# Sum occurrences across all documents to get total counts per word
word_counts = word_counts_df.sum().sort_values(ascending=False)

# Create a DataFrame with words and their occurrences
word_occurrences_sorted = pd.DataFrame({
    'Word': word_counts.index,
    'Occurrences': word_counts.values
}).sort_values(by='Occurrences', ascending=False)

# Select the top 30 words for brand H_
top_words_h = word_occurrences_sorted.head(30)
```

```
# Create a bar plot
plt.figure(figsize=(12, 9))  # Adjust the figure size as needed
plt.bar(top_words_h['Word'], top_words_h['Occurrences'], color='blue')  # Create a bar plot

# Add title and labels
plt.title('Top 30 Occurring Words for Brand H_')
plt.xlabel('Words')
plt.ylabel('Occurrences')

# Rotate the x-axis labels for better readability
plt.xticks(rotation=45, ha="right")

# Show the plot
plt.show()

#Visualizations for Brand Z_

# Filter the DataFrame for brand Z_
df_z = df[df['brand_name_'] == 'Z_']

# Count the occurrences of each emotion for brand Z_
emotion_counts_z = df_z['emotions_'].value_counts().sort_index()

# Create a bar plot for emotions
plt.figure(figsize=(10, 6))
plt.bar(emotion_counts_z.index.map(emotion_labels), emotion_counts_z.values, color='skyblue')  # Assuming emotion_labels mapping exists
plt.xlabel('Emotions')
plt.ylabel('Count')
```

```
plt.title('Count of Each Emotion for Brand Z_')
plt.xticks(rotation=45)
plt.show()

# Count the occurrences of each star rating for brand Z_
star_counts_z = df_z['star_rating_'].value_counts().sort_index()

# Create a bar plot for star ratings
plt.figure(figsize=(10, 6))
plt.bar(star_counts_z.index, star_counts_z.values, color='green')
plt.xlabel('Star Ratings')
plt.ylabel('Count')
plt.title('Count of Each Star Rating for Brand Z_')
plt.xticks(rotation=0)  # Star ratings are numeric and typically don't need rotation
plt.show()



# Calculate the average polarity for brand Z_
average_polarity_z = df_z['Polarity'].mean()

# Print the average polarity
print("Average Polarity for Brand Z_:", average_polarity_z)



# Initialize the vectorizer
vectorizer = CountVectorizer()

# Fit and transform the text data
X = vectorizer.fit_transform(df_z['lemmatized_text'])
```

```
# Convert to a DataFrame
word_counts_df = pd.DataFrame(X.toarray(), columns=vectorizer.get_feature_names_out())

# Sum occurrences across all documents to get total counts per word
word_counts = word_counts_df.sum().sort_values(ascending=False)

# Create a DataFrame with words and their occurrences
word_occurrences_sorted = pd.DataFrame({
    'Word': word_counts.index,
    'Occurrences': word_counts.values
}).sort_values(by='Occurrences', ascending=False)

# Select the top 30 words for brand Z_
top_words_z = word_occurrences_sorted.head(30)

# Create a bar plot
plt.figure(figsize=(12, 9))  # Adjust the figure size as needed
plt.bar(top_words_z['Word'], top_words_z['Occurrences'], color='blue')  # Create a bar plot

# Add title and labels
plt.title('Top 30 Occurring Words for Brand Z_')
plt.xlabel('Words')
plt.ylabel('Occurrences')

# Rotate the x-axis labels for better readability
plt.xticks(rotation=45, ha="right")

# Show the plot
```

```
plt.show()

# Count the occurrences of each brand
brand_counts = df['brand_name_'].value_counts()
print(brand_counts)

# Set up the matplotlib figure
plt.figure(figsize=(8, 5))

# Create a bar plot
plt.bar(brand_counts.index, brand_counts.values, color='teal')

# Add labels and title
plt.xlabel('Brand Name')
plt.ylabel('Number of Rows')
plt.title('Count of Rows for Each Brand')

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