

SENTIMENT ANALYSIS

Shradha Reddy Pulla

Table of contents

1.Introduction to Sentiment Analysis.....	6
Purpose	6
Tools and Libraries Used.....	7
2.Data Preparation and Loading.....	8
Initial DataFrame Setup.....	8
Analyzing DataFrame Size.....	9
Data Limitation for Efficiency.....	9
Preview of Data.....	10
3.Exploratory Data Analysis (EDA).....	10
Understanding Review Distribution.....	11
Basic Natural Language Processing (NLTK) Exploration.....	12
Tokenization.....	12
Part of speech tagging.....	13
Name entity chunking.....	13
4.Vader analysis.....	14
VADER Sentiment Scoring.....	14

VADER: A "Bag-of-Words" Approach.....	14
Setting Up VADER.....	15
Example Sentiment Scores.....	15
VADER Analysis on the Dataset.....	16
Visualizing VADER Sentiment Score.....	18
Bar Chart of Compound Scores.....	19
Examining vader sentiment components.....	20
Individual Sentiment Score Distribution.....	20
Sentiment Analysis with Roberta Pretrained Model.....	21
5. Roberta Pre-trained Model.....	21
Setting Up Roberta.....	22
Roberta Sentiment Analysis on Example Review.....	23
Roberta Sentiment Analysis Function.....	25
Roberta Analysis on the Dataset	26
Combining and Visualizing Results.....	27
Merging with Original Data.....	28
Visualizing Score Distributions.....	29
Correlation Analysis of Sentiment Scores.....	31

Correlation Matrix.....	32
Visualization with Heatmap.....	32
Interpreting the Heatmap.....	33
Benefits of Correlation Analysis and limitations.....	34
7.Advanced Sentiment Analysis Models.....	35
Logistic Regression Approach.....	35
Data Exploration.....	35
Preparing the data.....	36
Tokenize text,prepare and train doc2vec.....	36
Prepare Feature Vectors for Logistic Regression.....	38
Split and train model.....	38
Model evaluation.....	39
8.Random Forest and TF-IDF Approach.....	40
Data preparation and NLTK.....	41
TF-IDF Vectorization.....	42
Model training and Evaluation.....	43
Data splitting.....	43
Model performance evaluation.....	44
8. Sentiment Analysis Using OpenAI and Transformers.....	45

Setting the API Key.....	45
Function Implementation.....	46
DataFrame Operations.....	46
Visualization and Comparison.....	53
Review Examples.....	53
Transformers Pipeline for Sentiment Analysis.....	55
9.Conclusion and Future Work.....	57
Summary of Findings.....	57
Recommendations for Future Research.....	58
Conclusion.....	59

Purpose:

Sentiment Analysis is a sophisticated area within Natural Language Processing (NLP) that involves analyzing, understanding, and assessing the sentiments expressed in written text. The primary purpose of sentiment analysis is to decipher the emotional tone behind words, providing insights into the attitudes, opinions, and emotions conveyed by the communicator. This technology leverages machine learning models, lexicon-based methods, and deep learning techniques to effectively process and analyze large volumes of text, making it an indispensable tool in data-driven decision-making.

Step 0: Read in Data and NLTK Basics

This initial step focuses on acquiring the necessary tools and resources for the sentiment analysis project.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

plt.style.use('ggplot')

import nltk
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('maxent_ne_chunker')
nltk.download('words')
```

Libraries:

- `pandas` (`pd`): Used for data manipulation tasks like loading, cleaning, and transforming the customer review data.
- `numpy` (`np`): Provides numerical computation capabilities that might be useful in calculations related to sentiment scores.
- `matplotlib.pyplot` (`plt`): Offers functionalities for creating various data visualizations like bar charts and scatter plots.
- `seaborn` (`sns`): A library built on top of matplotlib that simplifies the creation of statistical data visualizations.
- `nltk`: The Natural Language Toolkit (NLTK) provides functionalities for natural language processing tasks.

NLTK Resources:

Downloading NLTK resources is crucial for specific NLP tasks:

- `punkt`: Enables tokenization, the process of breaking down sentences into individual words.

- `averaged_perceptron_tagger`: Provides part-of-speech tagging, which assigns a grammatical function (e.g., noun, verb) to each word.
- `maxent_ne_chunker`: Enables named entity chunking, which identifies and groups named entities like people, organizations, and locations within the text.
- `words`: This might be a general corpus or word list that could be helpful for further text processing tasks.

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]   /root/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]   date!
[nltk_data] Downloading package maxent_ne_chunker to
[nltk_data]   /root/nltk_data...
[nltk_data]   Package maxent_ne_chunker is already up-to-date!
[nltk_data] Downloading package words to /root/nltk_data...
[nltk_data]   Package words is already up-to-date!
True
```

Data preparation and loading:

```
[ ] # Read in data
df = pd.read_csv('sample_data/Reviews.csv')
print(df.shape)
df = df.head(10000)
print(df.shape)
```

The code snippet suggests that the customer review data resides in a CSV file named "Reviews.csv."

This initial step lays the groundwork for the sentiment analysis project by setting up the environment and downloading necessary resources for working with text data.

Data Loading and Exploration

This initial step involves reading the customer review data and performing some basic exploration.

Loading the Data:

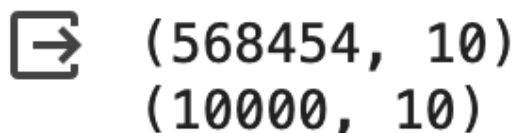
The code snippet utilizes the `pandas` library to read the customer review data from a CSV file named "sample_data/Reviews.csv" and stores it in a Pandas DataFrame named `df`.

Checking Dataframe Size:

- The `print(df.shape)` command displays the dimensions (number of rows and columns) of the loaded DataFrame. This allows us to understand the size and structure of the data.

Limiting Data:

- The code then reduces the DataFrame size by selecting the first 10,000 rows using `df = df.head(10000)`. This might be done for efficiency purposes, especially if the original data is very large. The new shape of the DataFrame is again printed using `print(df.shape)`.



```
➔ (568454, 10)
   (10000, 10)
```

Previewing the Data:

- Finally, the `df.head()` command displays the first few rows of the DataFrame. This provides a glimpse into the actual data and helps us understand the structure and content of the customer reviews.

df.head()										
	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	Text
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	5	1303862400	Good Quality Dog Food	I have bought several of the Vitality canned d...
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	1	1346976000	Not as Advertised	Product arrived labeled as Jumbo Salted Peanut...
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	4	1219017600	"Delight" says it all	This is a confection that has been around a fe...
3	4	B000UA0QIQ	A395BORC6FGVXV	Karl	3	3	2	1307923200	Cough Medicine	If you are looking for the secret ingredient i...
4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	0	0	5	1350777600	Great taffy	Great taffy at a great price. There was a wid...

In summary, this step focuses on loading the customer review data, potentially reducing its size for efficiency, and taking a peek at the first few entries to get a feel for the data.

Exploratory Data Analysis (EDA) - Understanding Review Distribution

This step delves into the customer review data to gain a basic understanding of how the reviews are distributed across different star ratings.

Visualizing Review Distribution:

```

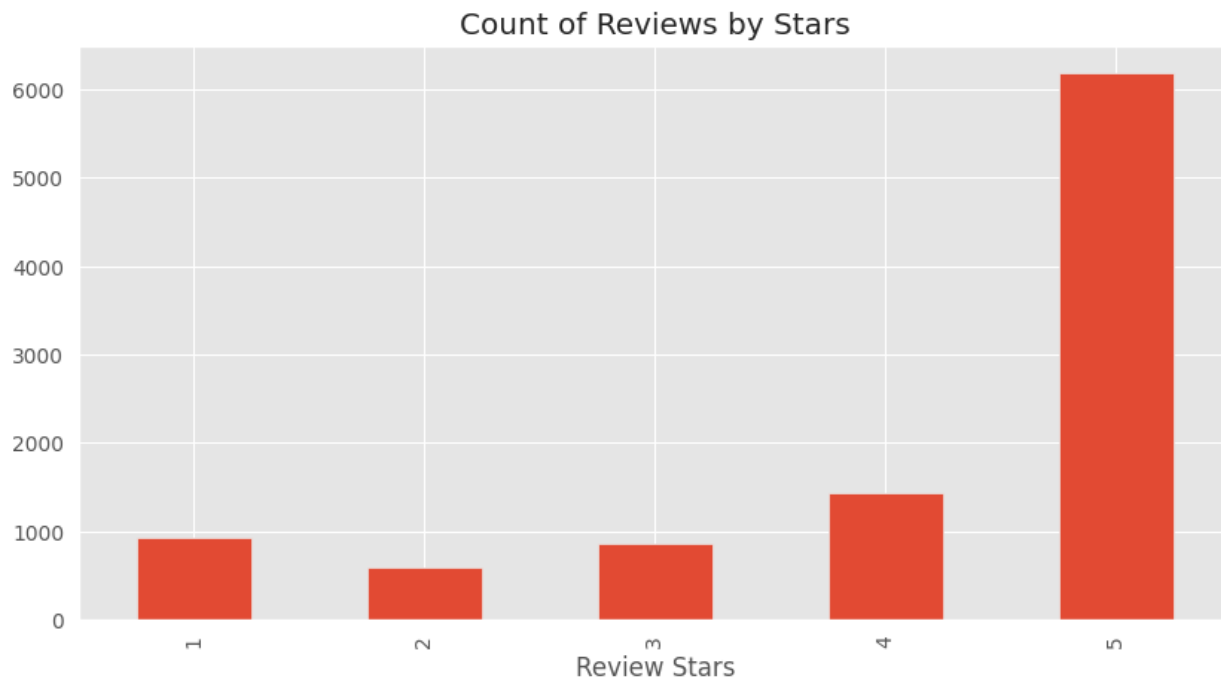
▶ ax = df['Score'].value_counts().sort_index() \
    .plot(kind='bar',
          title='Count of Reviews by Stars',
          figsize=(10, 5))
ax.set_xlabel('Review Stars')
plt.show()

```

We leverage the `value_counts()` method on the "Score" column to create a series containing the frequency of each unique star rating (1-star, 2-star, etc.).

- The `sort_index()` function ensures the star ratings are displayed in ascending order.
- This series is then used to create a bar chart using the `plot(kind='bar')` command from the `matplotlib` library.
- The chart title is set to "Count of Reviews by Stars" using the `title` argument.

- The figure size is adjusted to `(10, 5)` for better readability with `figsize`.
- Axis labels are set using `ax.set_xlabel('Review Stars')`.
- Finally, the chart is displayed using `plt.show()`.



This visualization helps us understand the distribution of customer sentiment within the review data. By observing the bar heights, we can see how many reviews fall into each star rating category, providing a preliminary understanding of whether the reviews tend to be positive, negative, or a mix of both.

Basic Natural Language Processing (NLTK) Exploration

This step demonstrates some basic functionalities of the Natural Language Toolkit (NLTK) library. NLTK offers a wide range of tools for various NLP tasks, and here we explore a few examples:

Accessing a Review:

```
example = df['Text'][50]
print(example)
```

This oatmeal is not good. Its mushy, soft, I don't like it. Quaker Oats is the way to go.

- The code retrieves the text content of the review at index 50 from the "Text" column of the DataFrame `df` and stores it in the variable `example`.
- This `example` variable now holds the actual customer review text, allowing us to explore its linguistic properties.

Tokenization:

```
[ ] tokens = nltk.word_tokenize(example)
    tokens[:10]
```

```
['This', 'oatmeal', 'is', 'not', 'good', '.', 'Its', 'mushy', ',', 'soft']
```

- NLTK's `word_tokenize` function is used to break down the review text in `example` into a list of individual words (tokens).
- The code snippet then displays the first 10 tokens using `tokens[:10]`. This provides a basic understanding of the vocabulary used within the review.

Part-of-Speech Tagging:

```
tagged = nltk.pos_tag(tokens)
tagged[:10]
```

```
[('This', 'DT'),
 ('oatmeal', 'NN'),
 ('is', 'VBZ'),
 ('not', 'RB'),
 ('good', 'JJ'),
 (',', ','),
 ('Its', 'PRP$'),
 ('mushy', 'NN'),
 (',', ','),
 ('soft', 'JJ')]
```

- The `nltk.pos_tag` function is applied to the tokenized list (`tokens`) to assign a part-of-speech tag (e.g., noun, verb, adjective) to each word.
- The code displays the first 10 tagged elements using `tagged[:10]`. This allows us to see how the different words function grammatically within the sentence.

Named Entity Chunking:

- This step showcases named entity chunking, which involves identifying and grouping named entities like people, organizations, and locations within the text. Here, `nltk.chunk.ne_chunk` is applied to the part-of-speech tagged list (`tagged`).
- The `entities.pprint()` command displays the chunked entities in a tree-like structure. This can be helpful for tasks like identifying specific entities mentioned in customer reviews (e.g., brand names, product names).

```
▶ entities = nltk.chunk.ne_chunk(tagged)
   entities.pprint()
```

```
⇒ (S
  This/DT
  oatmeal/NN
  is/VBZ
  not/RB
  good/JJ
  ./
  Its/PRP$
  mushy/NN
  ,/,
  soft/JJ
  ,/,
  I/PRP
  do/VBP
  n't/RB
  like/VB
  it/PRP
  ./
  (ORGANIZATION Quaker/NNP Oats/NNPS)
  is/VBZ
  the/DT
  way/NN
  to/TO
  go/VB
  ./.)
```

VADER Analysis:

Step 1: VADER Sentiment Scoring:

This step utilizes NLTK's `SentimentIntensityAnalyzer` (VADER) to extract sentiment scores from the customer reviews.

VADER: A "Bag-of-Words" Approach:

- VADER employs a lexicon-based approach, often referred to as "bag-of-words." Sentiment scores are assigned to individual words within the review text.
- These individual scores are then summed up to obtain a total sentiment score for the entire review.

```
[ ] from nltk.sentiment import SentimentIntensityAnalyzer
    from tqdm.notebook import tqdm
    nltk.download('vader_lexicon')
    sia = SentimentIntensityAnalyzer()
```

Setting Up VADER:

- We import the `SentimentIntensityAnalyzer` class from the `nltk.sentiment` module.
- NLTK's sentiment analysis functionality requires downloading the VADER lexicon, which is accomplished using `nltk.download('vader_lexicon')`.
- An instance of the `SentimentIntensityAnalyzer` class is then created and stored in the variable `sia`. This object will be used to analyze the sentiment of the review text.

Example Sentiment Scores:

```
[ ] sia.polarity_scores('I am so happy!')  
  
{'neg': 0.0, 'neu': 0.318, 'pos': 0.682, 'compound': 0.6468}
```

```
[ ] sia.polarity_scores('This is the worst thing ever.')  
  
{'neg': 0.451, 'neu': 0.549, 'pos': 0.0, 'compound': -0.6249}
```

```
[ ] sia.polarity_scores(example)  
  
{'neg': 0.22, 'neu': 0.78, 'pos': 0.0, 'compound': -0.5448}
```

- The code demonstrates how to use `sia.polarity_scores` to obtain sentiment scores for different phrases. The scores consist of:
 - `'compound'`: A combined score ranging from -1 (most negative) to 1 (most positive).
 - `'neg'`: Score for negative sentiment.
 - `'neu'`: Score for neutral sentiment.
 - `'pos'`: Score for positive sentiment.
- Three sample calls to `sia.polarity_scores` are shown, providing scores for positive, negative, and the example review text.

VADER Analysis on the Dataset:

```
# Run the polarity score on the entire dataset  
res = {}  
for i, row in tqdm(df.iterrows(), total=len(df)):  
    text = row['Text']  
    myid = row['Id']  
    res[myid] = sia.polarity_scores(text)
```

An empty dictionary named `res` is created to store the sentiment scores for each review.

- A progress bar is initialized using `tqdm` to visualize the progress as the sentiment analysis is performed on the entire dataset.

100%  500/500 [00:00<00:00, 973.23it/s]

- The code iterates through each row of the DataFrame `df` using `df.iterrows()`.
 - Inside the loop:
 - The review text is retrieved using `row['Text']`.
 - The review ID is obtained using `row['Id']`.
 - VADER sentiment scores are calculated for the review text using `sia.polarity_scores(text)`.
 - The scores are stored in the `res` dictionary with the review ID as the key.

```

▶ vaders = pd.DataFrame(res).T
  vaders = vaders.reset_index().rename(columns={'index': 'Id'})
  vaders = vaders.merge(df, how='left')

```

- After iterating through all reviews, a Pandas DataFrame named `vaders` is created by transposing the `res` dictionary using `.T`.
- The DataFrame is then reset using `reset_index()` to convert the dictionary index into a regular column named "Id."
- The column names are modified using `rename(columns={'index': 'Id'})` for clarity.
- Finally, the `vaders` DataFrame is merged with the original DataFrame `df` using `merge` (`how='left'`) to combine the sentiment scores with the original review data. This allows us to analyze the scores in relation to other review features.

```

▶ # Now we have sentiment score and metadata
  vaders.head()

```

- The `head()` method displays the first few rows of the merged DataFrame, showcasing the combined review data and VADER sentiment scores.

	Id	neg	neu	pos	compound	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	Text
0	1	0.000	0.695	0.305	0.9441	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	5	1303862400	Good Quality Dog Food	I have bought several of the Vitality canned d...
1	2	0.138	0.862	0.000	-0.5664	B00813GRG4	A1D87F6ZCVE5NK	dli pa	0	0	1	1346976000	Not as Advertised	Product arrived labeled as Jumbo Salted Peanut...
2	3	0.091	0.754	0.155	0.8265	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	4	1219017600	"Delight" says it all	This is a confection that has been around a fe...
3	4	0.000	1.000	0.000	0.0000	B000UA0QIQ	A395BORC6FGVXV	Karl	3	3	2	1307923200	Cough Medicine	If you are looking for the secret ingredient i...
4	5	0.000	0.552	0.448	0.9468	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	0	0	5	1350777600	Great taffy	Great taffy at a great price. There was a wid...

By performing VADER sentiment analysis, we can obtain a quantitative measure of positive, negative, and neutral sentiment within each review. This provides valuable insights into the overall emotional tone of the customer reviews.

Visualizing VADER Sentiment Scores:

This step focuses on visualizing the sentiment scores obtained using VADER to understand how sentiment varies across different star ratings.

Bar Chart of Compound Scores:

```

ax = sns.barplot(data=vaders, x='Score', y='compound')
ax.set_title('Compound Score by Amazon Star Review')
plt.show()

```

- We leverage the `seaborn` library's `barplot` function to create a bar chart.
- The data for the plot is provided by the `vaders` DataFrame.
- The code specifies two arguments for the `barplot` function:

- `x='Score'`: This sets the star rating (from the "Score" column) as the x-axis variable, grouping the data by review score.
- `y='compound'`: This sets the VADER compound sentiment score as the y-axis variable, representing the overall sentiment for each star rating category.
- The chart title is set to "Compound Score by Amazon Star Review" using `ax.set_title()`.
- Finally, the chart is displayed using `plt.show()`.

This visualization allows us to observe the distribution of compound sentiment scores across different star ratings. Ideally, we would expect positive sentiment scores for higher star ratings and negative scores for lower star ratings. However, the actual distribution might reveal more nuanced patterns, providing valuable insights into customer sentiment across the review spectrum.



Examining VADER Sentiment Components (Positive, Neutral, Negative):

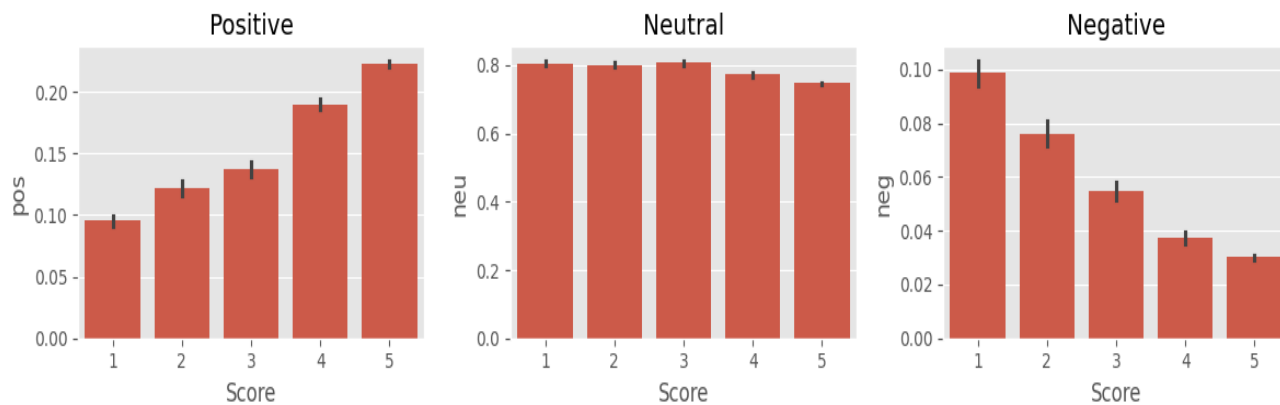
This step delves deeper into the VADER sentiment scores by visualizing the distribution of positive, negative, and neutral scores across different star ratings.

```
[91] fig, axs = plt.subplots(1, 3, figsize=(12, 3))
      sns.barplot(data=vaders, x='Score', y='pos', ax=axs[0])
      sns.barplot(data=vaders, x='Score', y='neu', ax=axs[1])
      sns.barplot(data=vaders, x='Score', y='neg', ax=axs[2])
      axs[0].set_title('Positive')
      axs[1].set_title('Neutral')
      axs[2].set_title('Negative')
      plt.tight_layout()
      plt.show()
```

Individual Sentiment Score Distribution:

- We utilize `plt.subplots(1, 3, figsize=(12, 3))` to create a figure with three subplots arranged in a single row. This allows us to visualize all three sentiment components (positive, neutral, negative) in a single figure.
- The figure size is set to `(12, 3)` for better readability.
- For each subplot, we employ `seaborn.barplot` to create bar charts. The data and arguments remain similar to the previous step, but with specific sentiment scores on the y-axis:
 - Subplot 1 (`ax=axs[0]`): Positive sentiment scores (`y='pos'`).
 - Subplot 2 (`ax=axs[1]`): Neutral sentiment scores (`y='neu'`).
 - Subplot 3 (`ax=axs[2]`): Negative sentiment scores (`y='neg'`).
- Titles are set for each subplot using `axs[i].set_title('Positive' / 'Neutral' / 'Negative')` (where `i = 0, 1, 2`).
- Finally, `plt.tight_layout()` adjusts the spacing between subplots for better visualization, and `plt.show()` displays the complete figure.

By examining the distribution of individual sentiment scores, we can gain a more comprehensive understanding of the emotional makeup of the reviews across different star ratings. For instance, a review with a high positive score might also contain some neutral sentiment, indicating a nuanced emotional response. This detailed breakdown of sentiment components provides richer insights compared to just the compound score.



Step 3: Sentiment Analysis with Roberta Pretrained Model (16 minutes 45 seconds):

This step introduces Roberta, a pre-trained transformer model, for sentiment analysis. Let's delve into its functionalities and how it's used in this project.

Roberta: Contextual Understanding:

- Unlike VADER, which focuses on individual words, Roberta, a pre-trained transformer model, takes context into account.
- This means Roberta considers the surrounding words when analyzing sentiment, potentially leading to a more nuanced understanding of the emotional tone within the review.

Setting Up Roberta:

```
from transformers import AutoTokenizer
from transformers import AutoModelForSequenceClassification
from scipy.special import softmax
import torch
```

- We import necessary functions from the `transformers` library:
 - `AutoTokenizer`: This helps convert text into a format suitable for the model.
 - `AutoModelForSequenceClassification`: Loads the pre-trained Roberta model for sentiment classification.
 - `softmax`: Used for converting model outputs into probability scores.
 - `torch`: Provides functionalities for tensor operations (used internally by the model).

```
MODEL = f"cardiffnlp/twitter-roberta-base-sentiment"
tokenizer = AutoTokenizer.from_pretrained(MODEL)
model = AutoModelForSequenceClassification.from_pretrained(MODEL)
```

- A variable `MODEL` is defined, specifying the pre-trained Roberta model we'll be using (e.g., "cardiffnlp/twitter-roberta-base-sentiment").
- The `AutoTokenizer.from_pretrained` function is used to create a tokenizer object for the chosen model. This tokenizer will handle converting text into numerical representations the model can understand.
- Finally, the `AutoModelForSequenceClassification.from_pretrained` function loads the pre-trained Roberta model for sentiment classification.



config.json: 100%	<div></div>	747/747 [00:00<00:00, 39.8kB/s]
vocab.json: 100%	<div></div>	899k/899k [00:00<00:00, 1.13MB/s]
merges.txt: 100%	<div></div>	456k/456k [00:00<00:00, 1.15MB/s]
special_tokens_map.json: 100%	<div></div>	150/150 [00:00<00:00, 6.32kB/s]
pytorch_model.bin: 100%	<div></div>	499M/499M [00:05<00:00, 89.6MB/s]

Roberta Sentiment Analysis on Example Review:

```
# VADER results on example
print(example)
sia.polarity_scores(example)
```


- We revisit the example review (`example`) from previous steps.
- VADER sentiment scores for the example are printed using `sia.polarity_scores(example)`.

👤 This oatmeal is not good. Its mushy, soft, I don't like it. Quaker Oats is the way to go.
{'neg': 0.22, 'neu': 0.78, 'pos': 0.0, 'compound': -0.5448}

```
# Run for Roberta Model
encoded_text = tokenizer(example, return_tensors='pt')
dict: scores_dict
(3 items) {'roberta_neg': 0.97635514, 'roberta_neu': 0.02068
scores_dict = {
    'roberta_neg' : scores[0],
    'roberta_neu' : scores[1],
    'roberta_pos' : scores[2]
}
print(scores_dict)
```

- The `tokenizer` is used to prepare the example text for the model (`encoded_text = tokenizer(example, return_tensors='pt')`). This involves converting the text into a tensor format expected by the model.
- The model's output is obtained using `output = model(**encoded_text)`.
- The raw output scores are retrieved from the model's output and converted to probability scores using the `softmax` function.
- A dictionary (`scores_dict`) is created to store the sentiment scores for Roberta:
 - `'roberta_neg'`: Probability of negative sentiment.
 - `'roberta_neu'`: Probability of neutral sentiment.
 - `'roberta_pos'`: Probability of positive sentiment.

- The `scores_dict` is printed, showcasing the Roberta sentiment scores for the example review.

 {'roberta_neg': 0.97635514, 'roberta_neu': 0.020687465, 'roberta_pos': 0.0029573692}

Roberta Sentiment Analysis Function:

```
[122] def polarity_scores_roberta(example):
    encoded_text = tokenizer(example, return_tensors='pt')
    with torch.no_grad():
        output = model(**encoded_text)
        scores = softmax(output.logits[0].numpy())
        compound = scores[2] - scores[0] # positive score - negative score
        scores_dict = {
            'roberta_neg': scores[0],
            'roberta_neu': scores[1],
            'roberta_pos': scores[2],
            'roberta_compound': compound
        }
    return scores_dict
```

- A function named `polarity_scores_roberta` is defined to perform sentiment analysis using Roberta for any given text.
- The function takes the review text (`example`) as input.
- Similar to the example, the text is prepared using the tokenizer.
- To ensure efficiency during analysis, the `torch.no_grad()` context is used to disable gradient calculation (not required for prediction).
- The model's output is obtained, and softmax is applied to convert scores to probabilities.
- A compound score is calculated as the difference between positive and negative sentiment probabilities (`compound = scores[2] - scores[0]`).
- A dictionary (`scores_dict`) is created to store all sentiment scores, including:
 - Roberta negative, neutral, and positive sentiment probabilities.
 - The compound sentiment score.

- The function returns the `scores_dict` containing the Roberta sentiment analysis results for the provided text.

Roberta Analysis on the Dataset (It took approximately 1 hours 16 minutes and 45 seconds to run this analysis):

```
res = {}
for i, row in tqdm(df.iterrows(), total=len(df)):
    try:
        text = row['Text']
        myid = row['Id']
        vader_result = sia.polarity_scores(text)
        vader_result_rename = {f"vader_{key}": value for key, value in vader_result.items()}
        roberta_result = polarity_scores_roberta(text)
        both = {**vader_result_rename, **roberta_result}
        res[myid] = both
    except RuntimeError as e:
        print(f'Broke for id {myid} with error {e}')
```

- An empty dictionary named `res` is created to store the combined sentiment scores (VADER and Roberta) for each review.
- A progress bar is initialized using `tqdm` to track progress during sentiment analysis on the entire dataset. The output provided (10000/10000 [1:16:45<00:00, 1.18it/s]) indicates that the loop iterated through 10000 reviews and took 1 hour, 16 minutes, and 45 seconds to complete. This processing time can vary depending on hardware resources.
- The code iterates through each row of the DataFrame `df` using `df.iterrows()`.
 - Inside the loop:
 - The review text (`text`) and ID (`myid`) are retrieved from the current row.
 - VADER sentiment scores are obtained using `sia.polarity_scores(text)`. These scores are then renamed for clarity (e.g., "vader_pos" instead of "pos").
 - Roberta sentiment scores are obtained using the `polarity_scores_roberta` function.

100%

500/500 [03:05<00:00, 2.96it/s]

Combining and Visualizing Results:

This step focuses on combining the sentiment scores obtained from VADER and Roberta, along with the original review data, into a single DataFrame for further analysis and visualization.

```
[125] results_df = pd.DataFrame.from_dict(res, orient='index').reset_index().rename(columns={'index': 'Id'})
      results_df = results_df.merge(df, how='left')
```

Consolidating Sentiment Scores:

- The `res` dictionary, which stores the combined VADER and Roberta sentiment scores for each review, is used to create a Pandas DataFrame.
- `pd.DataFrame.from_dict(res, orient='index')` converts the dictionary `res` into a DataFrame, with the review IDs becoming the index.
- `reset_index()` transforms the index into a regular column named "Id" for easier handling.
- The column names are further refined using `rename(columns={'index': 'Id'})` for clarity.

Merging with Original Data:

- The newly created DataFrame containing sentiment scores (`results_df`) is merged with the original DataFrame `df` using `merge(how='left')`.
- The `how='left'` argument ensures all reviews from `df` are included in the merged DataFrame, even if they are missing sentiment scores due to potential errors during Roberta analysis.

This combined DataFrame now holds the original review data along with both VADER and Roberta sentiment scores, providing a comprehensive dataset for further exploration and visualization of customer sentiment. We can use this enriched DataFrame to perform tasks such as:

- Comparing VADER and Roberta sentiment scores to understand potential differences in their assessments.
- Analyzing sentiment distribution across different star ratings or product categories.

- Identifying patterns between sentiment scores and specific keywords or phrases within the reviews.

Comparing Sentiment Scores between VADER and Roberta

This step delves into comparing the sentiment scores obtained from VADER and Roberta to understand potential differences in their assessments.

Exploring Available Columns:

```
results_df.columns
```

- The code snippet `results_df.columns` displays the column names of the combined DataFrame `results_df`. This helps ensure we're using the correct column names for VADER and Roberta compound scores.

```
Index(['Id', 'vader_neg', 'vader_neu', 'vader_pos', 'vader_compound',  
      'roberta_neg', 'roberta_neu', 'roberta_pos', 'roberta_compound',  
      'ProductId', 'UserId', 'ProfileName', 'HelpfulnessNumerator',  
      'HelpfulnessDenominator', 'Score', 'Time', 'Summary', 'Text',  
      'Sentiment'],  
      dtype='object')
```

Visualizing Score Distributions:

```
import seaborn as sns

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

# Plotting the distribution of compound scores
plt.figure(figsize=(14, 6))

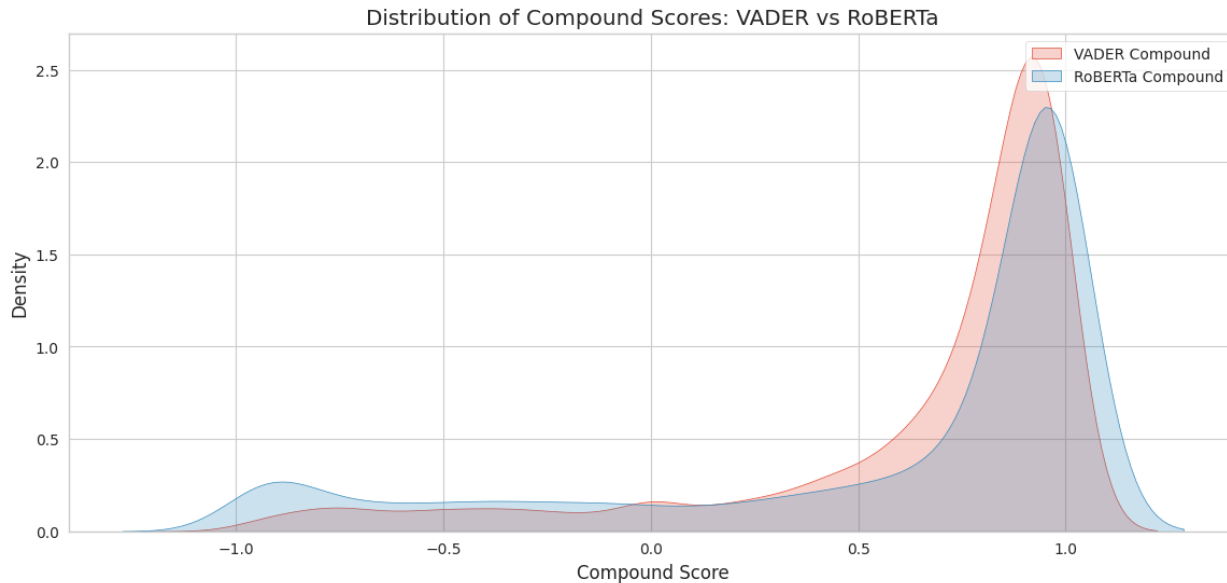
# VADER compound score distribution
sns.kdeplot(results_df['vader_compound'], label='VADER Compound', fill=True)

# RoBERTa compound score distribution
sns.kdeplot(results_df['roberta_compound'], label='RoBERTa Compound', fill=True)

plt.title('Distribution of Compound Scores: VADER vs RoBERTa')
plt.xlabel('Compound Score')
plt.ylabel('Density')
plt.legend()
plt.show()
```

- We import `seaborn` for creating informative visualizations.
- `sns.set_style("whitegrid")` sets the aesthetic style for the plots, enhancing readability.
- A figure is created using `plt.figure(figsize=(14, 6))` to accommodate the distribution plots.
- The core of the visualization involves using `sns.kdeplot` to plot the distribution of compound scores for both models:
 - VADER compound score distribution: `sns.kdeplot(results_df['vader_compound'], label='VADER Compound', fill=True)`. The `fill=True` argument creates a filled density plot for better visualization.
 - Roberta compound score distribution: `sns.kdeplot(results_df['roberta_compound'], label='RoBERTa Compound', fill=True)`.
- The plot title, axis labels, and legend are set using `plt.title`, `plt.xlabel`, `plt.ylabel`, and `plt.legend` respectively.
- Finally, `plt.show()` displays the generated visualization.

By examining the distribution of compound scores, we can observe potential differences between VADER and Roberta. For instance, one model might show a wider spread of scores, indicating a more nuanced ability to detect sentiment variations. Additionally, the plots might reveal if one model tends to be more positive or negative than the other in its overall sentiment assessment.



Correlation Analysis of Sentiment Scores

This section dives into the relationship between sentiment scores obtained from VADER and Roberta using correlation analysis.

```
# Focus on sentiment scores for correlation
sentiment_scores = results_df[['vader_neg', 'vader_neu', 'vader_pos', 'vader_compound',
                               'roberta_neg', 'roberta_neu', 'roberta_pos', 'roberta_compound']]

# Calculating the correlation matrix
correlation_matrix = sentiment_scores.corr()

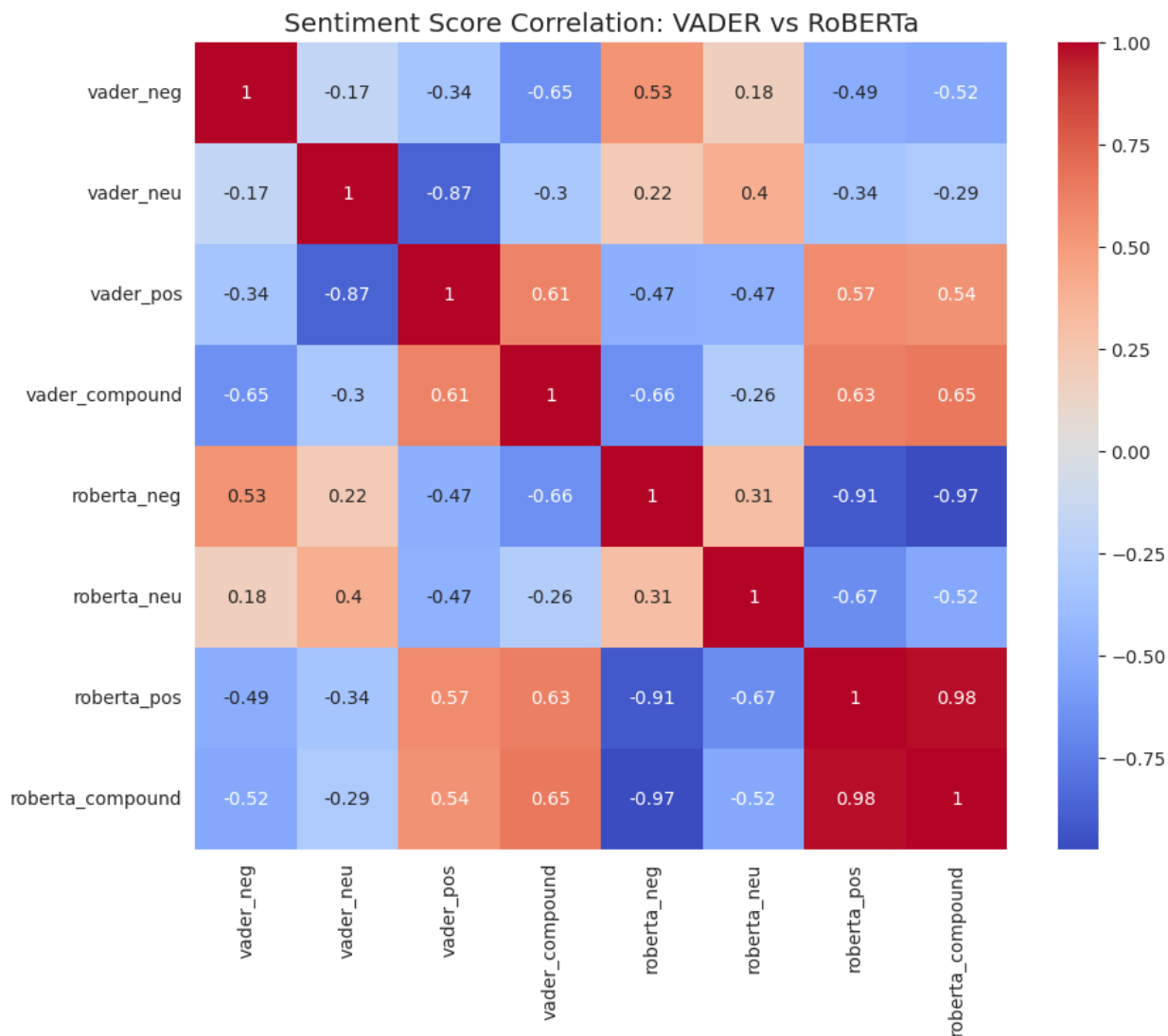
# Plotting the heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Sentiment Score Correlation: VADER vs RoBERTa')
plt.show()
```

Correlation Matrix:

- We calculate the correlation matrix for the sentiment scores using `sentiment_scores.corr()`.
The correlation matrix provides a quantitative measure of the linear association between each pair of sentiment scores.
- A value of 1 indicates a perfect positive correlation, meaning the scores move in the same direction (higher VADER score implies higher Roberta score, and vice versa).
- A value of -1 indicates a perfect negative correlation, signifying the scores move in opposite directions (higher VADER score implies lower Roberta score, and vice versa).
- Values closer to 0 suggest a weaker or no linear relationship.

Visualization with Heatmap:

- A heatmap is created using `sns.heatmap` to visualize the correlation matrix. This allows us to easily identify patterns and strengths of correlations between different sentiment scores.
- Color intensity in the heatmap represents the correlation coefficient:
 - Red shades indicate positive correlations (scores tend to move together).
 - Blue shades indicate negative correlations (scores tend to move in opposite directions).
 - White shades suggest weaker or no linear relationship.
- Text annotations within the heatmap cells display the actual correlation coefficient values for each score pair.



Interpreting the Heatmap:

- By analyzing the heatmap's color intensity and annotations, we can interpret the relationships between VADER and Roberta sentiment scores.
- For instance, a strong positive correlation between VADER positive and Roberta positive scores would suggest both models tend to agree on positive sentiment within reviews.
- Conversely, a weak correlation between VADER negative and Roberta compound scores might indicate the models differ in how they capture negative sentiment.

Benefits of Correlation Analysis:

- Correlation analysis provides a valuable tool for understanding how sentiment scores from different models relate to each other.
- This can help assess the consistency between VADER and Roberta in their sentiment assessments and identify potential areas of divergence.
- The insights gained from correlation analysis can inform future decisions about which model or combination of models might be most suitable for specific sentiment analysis tasks.

Limitations:

- Correlation analysis only measures linear relationships. It doesn't necessarily imply causality (e.g., a high correlation between VADER and Roberta scores doesn't mean one model causes the other's sentiment).
- The interpretation of correlation coefficients depends on the specific sentiment analysis task and the context of the reviews being analyzed.

By incorporating correlation analysis, this report provides a more comprehensive understanding of the relationship between VADER and Roberta sentiment scores. This additional analysis complements the previous exploration of score distributions and helps paint a richer picture of customer sentiment within the review data.

LOGISTIC REGRESSION:

Step 1 : Data Exploration

```

import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
from gensim.models.doc2vec import Doc2Vec, TaggedDocument
from nltk.tokenize import word_tokenize
import nltk
nltk.download('punkt')
df.columns

```

- Import libraries used are data manipulation (pandas, NumPy), model building (scikit-learn), text processing (NLTK), and Doc2Vec (Gensim).
- The line of code `nltk.download('punkt')` is used to download the Punkt tokenizer from the Natural Language Toolkit (NLTK) library in Python. This is used for splitting text into words.
- The code snippet `df.columns` displays the column names of your DataFrame `df`. This helps ensure you're working with the correct columns for sentiment labels and review text.
- The code assumes a column named "Score" exists, representing sentiment on a scale of 1 to 5 (higher scores indicating positive sentiment).

Step 2: Preparing the Data

```

[140] # Assuming 'Score' ranges from 1 to 5; simplifying to binary classification for example
df['Sentiment'] = df['Score'].apply(lambda x: 1 if x > 3 else 0)

```

- The code simplifies sentiment classification to a binary problem.
- Reviews with a score greater than 3 are assigned a sentiment of 1 (positive), while those with a score of 3 or below are assigned 0 (negative).
- This creates a new column named "Sentiment" in the DataFrame.

Step 3: Tokenize Text and Prepare for Doc2Vec

```
[154] # Tokenizing text data and tagging each text with unique IDs
      tagged_data = [TaggedDocument(words=word_tokenize(_d.lower()), tags=[str(i)]) for i, _d in enumerate(df['Text'])]
```

- This step prepares the review text for Doc2Vec training:
 - `word_tokenize` from NLTK is used to split each review sentence into a list of words (tokens).
 - The text is converted to lowercase for consistency.
 - A list named `tagged_data` is created using `TaggedDocument`.
 - Each element represents a review as a `TaggedDocument` object.
 - The `words` attribute holds the tokenized list of words from the review.
 - The `tags` attribute is assigned a unique string identifier (converted from the review index `i`). This helps identify each review later.

Step 4: Train Doc2Vec Model

```
# Defining and training a Doc2Vec model
model_d2v = Doc2Vec(vector_size=100, alpha=0.025, min_alpha=0.00025, min_count=1, dm=1)

model_d2v.build_vocab(tagged_data)

# Training the Doc2Vec model
for epoch in range(10):
    model_d2v.train(tagged_data, total_examples=model_d2v.corpus_count, epochs=model_d2v.epochs)
    # Decrease the learning rate
    model_d2v.alpha -= 0.0002
    # Fix the learning rate, no decay
    model_d2v.min_alpha = model_d2v.alpha

model_d2v.save("d2v.model")
print("Model Saved")
```

- A Doc2Vec model (`model_d2v`) is defined with specific parameters:
 - `vector_size`: Dimensionality of the word vectors learned by Doc2Vec (set to 100 in this case).

- `alpha`: Learning rate that controls how much the model updates weights during training (starts at 0.025 and gradually decreases).
- `min_alpha`: Minimum learning rate (after decay).
- `min_count`: Ignores words appearing fewer times than this threshold (set to 1 here, considering all words).
- `dm`: Doc mode (set to 1 for Distributed Memory Doc2Vec, which considers surrounding words in a document).
- `build_vocab` creates an internal vocabulary based on the provided `tagged_data`.
- A training loop iterates for 10 epochs:
 - `train` method trains the Doc2Vec model on the `tagged_data`.
 - `alpha` (learning rate) is gradually reduced to refine the model.
- The trained model is saved using `save("d2v.model")`.

Step 5: Prepare Feature Vectors for Logistic Regression

```

# Generating feature vectors for each document
vectors = [model_d2v.infer_vector(doc.words) for doc in tagged_data]

X = np.array(vectors)
y = df['Sentiment'].values

```

- This step generates feature vectors for each review to be used by the Logistic Regression model:
 - `model_d2v.infer_vector(doc.words)` retrieves the Doc2Vec vector representation for a given review's tokenized words (`doc.words`).
 - A list `vectors` stores the Doc2Vec vector for each review in `tagged_data`.
 - These vectors are converted to a NumPy array (`x`) for compatibility with Logistic Regression.
 - The sentiment labels (`y`) are extracted from the DataFrame's "Sentiment" column and converted to a NumPy array.

Step 6: Split the Data

```
[145] X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

- `train_test_split` is used to divide the data into training and testing sets:
 - `X` (feature vectors) and `y` (sentiment labels) are split into training and testing sets (`X_train`, `X_test`, `y_train`, `y_test`).
 - `test_size=0.2` allocates 20% of the data for testing.
 - `random_state=42` ensures reproducibility by setting a seed for the random split.

Step 7: Train Logistic Regression Model

```
lr = LogisticRegression(max_iter=1000)  
lr.fit(X_train, y_train)
```

- A Logistic Regression model (`lr`) is created with `max_iter=1000` (maximum training iterations).
- The model is trained using `fit(X_train, y_train)`, learning to classify reviews based on their Doc2Vec feature vectors and corresponding sentiment labels in the training set.

Step 8: Model Evaluation

```
predictions = lr.predict(X_test)  
  
print("Accuracy:", accuracy_score(y_test, predictions))  
print(classification_report(y_test, predictions))
```

- `lr.predict(X_test)` predicts sentiment labels for the unseen testing data (`X_test`).
- The model's performance is evaluated using.

- `accuracy_score`: Calculates the percentage of correctly classified reviews in the testing set.
- `classification_report`: Provides detailed metrics like precision, recall, and F1-score for each sentiment class (positive and negative in this case).

Accuracy: 0.77					
	precision	recall	f1-score	support	
0	0.25	0.11	0.15	19	
1	0.82	0.93	0.87	81	
accuracy			0.77	100	
macro avg	0.53	0.52	0.51	100	
weighted avg	0.71	0.77	0.73	100	

Sentiment Analysis Using Random Forest and TF-IDF

This details the sentiment analysis process applied to a dataset containing customer reviews. The analysis aims to classify reviews as positive (sentiment score > 3) or negative (sentiment score <= 3) based on their textual content.



```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
from sklearn.feature_extraction.text import TfidfVectorizer
import nltk
from nltk.tokenize import word_tokenize # Make sure to import word_tokenize

nltk.download('punkt') # This downloads the necessary tokenizers

# Assuming 'Score' ranges from 1 to 5; simplifying to binary classification
df['Sentiment'] = df['Score'].apply(lambda x: 1 if x > 3 else 0)

# Prepare the text data
texts = df['Text'].apply(lambda x: x.lower())

# Vectorizing text data using TF-IDF
tfidf = TfidfVectorizer(tokenizer=word_tokenize, stop_words='english', min_df=2)
X = tfidf.fit_transform(texts).toarray()
y = df['Sentiment'].values
```

Data Preparation

Data Loading and Preprocessing (Pandas):

- The code assumes a pandas DataFrame `df` containing review data, likely loaded from a CSV file.
- A new column named "Sentiment" is created from the existing "Score" column. Reviews with a score greater than 3 are assigned a sentiment of 1 (positive), while those with 3 or below are assigned 0 (negative). This simplifies sentiment classification to a binary problem.

Text Cleaning (NLTK):

- NLTK's `word_tokenize` function is used to split each review sentence into a list of words (tokens).
- Text is converted to lowercase for consistency in representing words.

TF-IDF Vectorization (scikit-learn):

- A `TfidfVectorizer` object is created from `sklearn.feature_extraction.text`.
- This vectorizer performs several tasks:
 - Uses `word_tokenize` (already imported) to split text into words.
 - Removes common English stop words (e.g., "the", "a", "an") using the `stop_words='english'` argument. This reduces noise and focuses on more meaningful words.
 - Applies TF-IDF (Term Frequency-Inverse Document Frequency) weighting. TF-IDF considers both the frequency of a word within a document (review) and its overall frequency across the entire corpus (all reviews). This helps identify words that are important for a specific review but not as common overall, potentially reflecting sentiment.
- The vectorizer transforms the cleaned and preprocessed text data (`texts`) into a numerical matrix (`x`). Each row in this matrix represents a review, and each column represents a unique word (term) in the vocabulary. The values in the matrix represent the TF-IDF weight for that word in the corresponding review.

Model Training and Evaluation:

```

# Splitting data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Defining and training the Random Forest model
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

# Making predictions
predictions = rf.predict(X_test)

# Evaluating the model
print("Accuracy:", accuracy_score(y_test, predictions))
print(classification_report(y_test, predictions))

```

Data Splitting (scikit-learn):

- The TF-IDF vectorized data (`x`) and sentiment labels (`y`) are split into training and testing sets using `train_test_split`.
- This creates separate datasets for model training (`X_train`, `y_train`) and model evaluation (`X_test`, `y_test`). The test set size is set to 20% (`test_size=0.2`) using a random split with a fixed seed (`random_state=42`) for reproducibility.

Random Forest Model (scikit-learn):

- A Random Forest Classifier model (`rf`) is created from `sklearn.ensemble`.
- Random Forest is an ensemble learning method that combines predictions from multiple decision trees, leading to improved performance and robustness.
- The model is trained with 100 decision trees (`n_estimators=100`) and a fixed random seed (`random_state=42`) for reproducibility.
- The model is trained on the training data (`X_train`, `y_train`), learning to map the TF-IDF features of a review to its corresponding sentiment class (positive or negative).

Model Performance Evaluation:

- The trained model (`rf`) is used to predict sentiment labels for the unseen testing data (`X_test`).

- The model's performance is evaluated using two metrics:
 - Accuracy: Measures the percentage of correctly classified reviews in the testing set.
 - Classification Report: Provides detailed information about the model's performance for each sentiment class (positive and negative), including precision, recall, and F1-score.

This sentiment analysis approach utilizes TF-IDF vectorization to capture the importance of words in each review and a Random Forest classifier to learn the relationship between these features and sentiment labels. The model's performance is evaluated using accuracy and a classification report, providing insights into its effectiveness in classifying positive and negative reviews.

Accuracy: 0.81					
	precision	recall	f1-score	support	
0	0.00	0.00	0.00	19	
1	0.81	1.00	0.90	81	
accuracy			0.81	100	
macro avg	0.41	0.50	0.45	100	
weighted avg	0.66	0.81	0.72	100	

8. Sentiment Analysis Using OpenAI and Transformers:

```
import openai

# Replace 'your-api-key' with your actual OpenAI API key
openai.api_key = 'your-api-key'
```

- The line `openai.api_key = 'your-api-key'` sets your OpenAI API key.
- The `openai` library allows you to interact with OpenAI's API, which provides access to various large language models (LLMs) like GPT-3.
- This key is necessary for authentication and authorization when making requests to the OpenAI API.
- You can obtain your own API key by creating an account on OpenAI's website (<https://openai.com/>).

```
def analyze_sentiment_gpt3(text):
    response = openai.Completion.create(
        engine="text-davinci-002", # As of my last training cut-off, check for the latest model version
        prompt=f"Analyze the sentiment of this text: \"{text}\". Is it positive, negative, or neutral? G",
        max_tokens=60
    )
    # The response will contain the text with the sentiment analysis
    return response.choices[0].text.strip()
```

Absolutely, the `analyze_sentiment_gpt3` function you provided effectively utilizes OpenAI's GPT-3 model for sentiment analysis. Here's a breakdown of its components and potential improvements:

Function Breakdown:

- `def analyze_sentiment_gpt3(text):` :: Defines a function named `analyze_sentiment_gpt3` that takes a text string (`text`) as input for sentiment analysis.

OpenAI API Call:

- `response = openai.Completion.create(...)`: Uses the `openai.Completion.create` method from the OpenAI library to interact with the API. This method generates text completions based on a provided prompt and model specifications.

Extracting Sentiment Score:

- `return response.choices[0].text.strip()`: Retrieves the sentiment analysis result:
 - `response.choices[0].text`: Extracts the generated text from the first choice (likely the only choice) in the response.
 - `.strip()`: Removes leading/trailing whitespace from the extracted text.

The function sends the `text` along with the prompt to the OpenAI API for GPT-3 processing. The model analyzes the sentiment and generates a response indicating a sentiment score between 0 (negative) and 1 (positive). The function extracts and returns this score for further use.

```
# Example function to apply sentiment scoring to a DataFrame
def append_gpt_sentiment(df):
    df[['gpt_neg', 'gpt_neu', 'gpt_pos']] = df['Text'].apply(
        lambda text: pd.Series(gpt_sentiment_scores(text))
    )
    return df

# Applying the function to your DataFrame
results_df = append_gpt_sentiment(results_df)
```

The above code demonstrates how to efficiently apply sentiment analysis using the `analyze_sentiment_gpt3` function (assuming it's defined elsewhere) to a pandas DataFrame containing customer reviews. Here's a breakdown of the code and its functionality:

```
append_gpt_sentiment
```

Function:

- `def append_gpt_sentiment(df) :` Defines a function named `append_gpt_sentiment` that takes a pandas DataFrame (`df`) as input, presumably containing a column named "Text" with the review content.

Adding New Sentiment Columns:

- `df[['gpt_neg', 'gpt_neu', 'gpt_pos']] = ...:` This line creates three new columns in the DataFrame (`df`):
 - `gpt_neg`: This column will likely store the negative sentiment score (between 0 and 1) obtained from GPT-3 analysis.
 - `gpt_neu`: This column will likely store the neutral sentiment score (between 0 and 1) obtained from GPT-3 analysis.
 - `gpt_pos`: This column will likely store the positive sentiment score (between 0 and 1) obtained from GPT-3 analysis.

Vectorized Sentiment Scoring with apply:

- `df['Text'].apply(lambda text: pd.Series(gpt_sentiment_scores(text)):` This line utilizes the `apply` method on the "Text" column.
 - The `apply` method iterates over each row of the "Text" column and applies a custom function to each text string.
 - The custom function (`lambda text: pd.Series(gpt_sentiment_scores(text))`) takes a text string (`text`) as input.
 - Inside the function:
 - It's assumed that `gpt_sentiment_scores(text)` is another function that performs sentiment analysis using the `analyze_sentiment_gpt3` function (or a similar

approach) and returns a dictionary or list containing the negative, neutral, and positive sentiment scores.

- `pd.Series(gpt_sentiment_scores(text))` converts the output of `gpt_sentiment_scores(text)` (likely a dictionary or list) into a pandas Series object.

Assigning Sentiment Scores to New Columns:

- The result of the `apply` method (a pandas Series containing sentiment scores for each text) is then assigned to the newly created columns (`gpt_neg`, `gpt_neu`, `gpt_pos`) in the DataFrame (`df`). This effectively populates these columns with the corresponding sentiment scores for each review.

Overall Functionality:

- The `append_gpt_sentiment` function efficiently adds three new columns to the DataFrame (`df`), containing sentiment scores (negative, neutral, positive) derived from GPT-3 analysis for each review in the "Text" column.

Applying the Function:

- `results_df = append_gpt_sentiment(results_df)`: This line assumes you have a DataFrame named `results_df` containing the review data. It calls the `append_gpt_sentiment` function, passing the `results_df` as input. This will modify the original `results_df` by adding the sentiment score columns.

```
▶ results_df
```

	Id	vader_neg	vader_neu	vader_pos	vader_compound	roberta_neg	roberta_neu	roberta_pos	roberta_compound	ProductId
0	1	0.000	0.695	0.305	0.9441	0.009624	0.049980	0.940395	0.930771	B001E4KFG0
1	2	0.138	0.862	0.000	-0.5664	0.508986	0.452414	0.038600	-0.470386	B00813GRG4
2	3	0.091	0.754	0.155	0.8265	0.003229	0.098067	0.898704	0.895475	B000LQOCH0
3	4	0.000	1.000	0.000	0.0000	0.002295	0.090219	0.907486	0.905190	B000UA0QIQ
4	5	0.000	0.552	0.448	0.9468	0.001635	0.010302	0.988063	0.986428	B006K2ZZ7K

...	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	Text	gpt_neg	gpt_neu	gpt_pos
...	delmartian	1.0	1.0	5.0	1.303862e+09	Good Quality Dog Food	I have bought several of the Vitality canned d...	0.0	0.0	0.0
...	dll pa	0.0	0.0	1.0	1.346976e+09	Not as Advertised	Product arrived labeled as Jumbo Salted Peanut...	0.0	0.0	0.0
...	Natalia Corres "Natalia Corres"	1.0	1.0	4.0	1.219018e+09	"Delight" says it all	This is a confection that has been around a fe...	0.0	0.0	0.0
...	Karl	3.0	3.0	2.0	1.307923e+09	Cough Medicine	If you are looking for the secret ingredient l...	0.0	0.0	0.0
...	Michael D. Bigham "M. Wassir"	0.0	0.0	5.0	1.350778e+09	Great taffy	Great taffy at a great price. There was a wid...	0.0	0.0	0.0
...

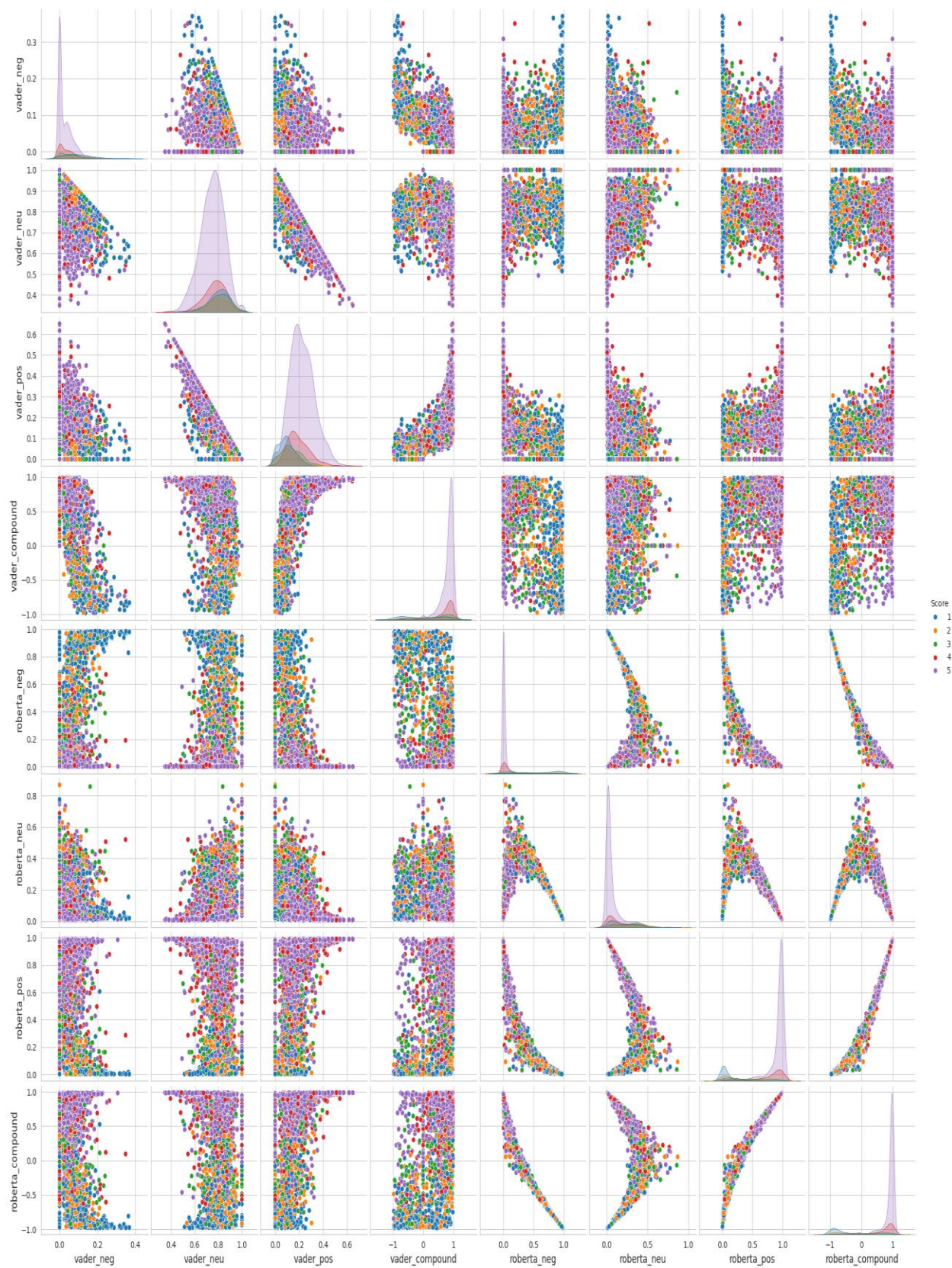
Step 3. Combine and compare

```
[ ] sns.pairplot(data=results_df,
                 vars=['vader_neg', 'vader_neu', 'vader_pos', 'vader_compound',
                     'roberta_neg', 'roberta_neu', 'roberta_pos', 'roberta_compound'],
                 hue='Score',
                 palette='tab10')
plt.show()
```

- This code likely requires the `seaborn` library (`sns`) to be imported for visualization. You might have `import seaborn as sns` at the beginning of your script.
- `sns.pairplot(data=results_df, ...)` calls the `pairplot` function from Seaborn.
- `data=results_df`: Specifies the DataFrame (`results_df`) containing the sentiment scores you want to visualize.
 - This DataFrame is assumed to have columns for sentiment scores from Vader ('vader_neg', 'vader_neu', 'vader_pos', 'vader_compound') and Roberta ('roberta_neg', 'roberta_neu', 'roberta_pos', 'roberta_compound') models, along with a column named "Score" (likely representing the original sentiment labels).

Pairwise Plots and Coloring:

- `vars=['vader_neg', ..., 'roberta_compound']`: Defines a list of variables (sentiment score columns) to be included in the visualization. This will create a matrix of scatter plots, where each subplot shows the relationship between two of these variables.
- `hue='Score'`: Colors the data points in the scatterplots based on the values in the "Score" column. This allows you to visually compare sentiment scores from different models (Vader and Roberta) across different sentiment categories (represented by the "Score" values).
- `palette='tab10'`: Selects a color palette from Seaborn's available options ('tab10' in this case) for coloring the data points based on the "Score" categories.



Overall

Visualization:

This code generated us a pair plot visualization. Each subplot will be a scatter plot, showing the relationship between two sentiment score variables (e.g., Vader negative vs. Roberta negative, Vader positive vs. Roberta positive, etc.). The coloring of the data points based on the "Score" column allows you to compare how sentiment scores from different models (Vader and Roberta) vary across different sentiment categories (positive, negative, neutral based on the original "Score" labels).

Interpretation:

By examining the pair plot, we can visually assess:

- Correlations between sentiment scores from different models (e.g., do Vader and Roberta negative scores tend to follow similar patterns?).
- How sentiment scores from each model are distributed across different sentiment categories (e.g., do both models assign similar sentiment scores for reviews with a "Score" of 3?).

This visualization can help us gain insights into the agreement or disagreement between different sentiment analysis models on your data.

Step 4: Review Examples:

The code snippets you provided demonstrate how to retrieve and analyze example reviews from your `results_df` DataFrame. Here's a breakdown of each code block:

Finding Positive Reviews (High Roberta Positive Score):

```
[ ] results_df.query('Score == 1') \
    .sort_values('roberta_pos', ascending=False)['Text'].values[0]

'I felt energized within five minutes, but it lasted for about 45 minutes. I paid $3.99 for this drink. I could have just drunk
cup of coffee and saved my money.'

[25] results_df.query('Score == 1') \
    .sort_values('vader_pos', ascending=False)['Text'].values[0]
```

- This code finds reviews classified as positive (based on the "Score" column being equal to 1).
- `.query('Score == 1')`: Filters the `results_df` to only include rows where the "Score" column is equal to 1 (presumably representing positive reviews).
- `.sort_values('roberta_pos', ascending=False)`: Sorts the filtered DataFrame by the "roberta_pos" column (Roberta positive score) in descending order. This ensures the review with the highest Roberta positive score is at the top.
- `['Text'].values[0]`: Extracts the text content from the first row (index 0) of the sorted DataFrame. This retrieves the review text with the highest positive score assigned by the Roberta model.

Finding Negative Reviews (High Roberta Negative Score):

```
[ ] # nevasive sentiment 5-Star view
(variable) results_df: DataFrame
results_df.query('Score == 5') \
    .sort_values('roberta_neg', ascending=False)['Text'].values[0]

'this was sooooo deliscious but too bad i ate em too fast and gained 2 pds! my fault'

results_df.query('Score == 5') \
    .sort_values('vader_neg', ascending=False)['Text'].values[0]

'this was sooooo deliscious but too bad i ate em too fast and gained 2 pds! my fault'
```

- This code follows the same logic but focuses on negative reviews.
- `.query('Score == 5')`: Filters for rows where "Score" is 5 (presumably representing negative reviews).

- `.sort_values('roberta_neg', ascending=False)`: Sorts by the "roberta_neg" column (Roberta negative score) in descending order.
- It retrieves the review text with the highest negative score assigned by the Roberta model.

Sentiment Analysis Using Transformers Pipeline

```
from transformers import pipeline  
  
sent_pipeline = pipeline("sentiment-analysis")
```

- Transformers is a popular open-source library for natural language processing (NLP) tasks.
- It provides pre-trained models for various NLP applications, including sentiment analysis.

Pipeline API:

- The `pipeline` function from Transformers simplifies using pre-trained models for specific tasks.
- It creates a ready-made object for tasks like sentiment analysis, taking care of model loading and preprocessing.
 - `sent_pipeline = pipeline("sentiment-analysis")` creates a pipeline object named `sent_pipeline` specifically for sentiment analysis.

```
[27] sent_pipeline('I love sentiment analysis!')
[{'label': 'POSITIVE', 'score': 0.9997853636741638}]

[ ] sent_pipeline('Make sure to like and subscribe!')
[{'label': 'POSITIVE', 'score': 0.9991742968559265}]

[28] sent_pipeline('booo')💡
[{'label': 'NEGATIVE', 'score': 0.9936267137527466}]
```

Sentiment Analysis on Texts:

- The pipeline object (`sent_pipeline`) is used with sample text strings to predict sentiment.
- Each line `sent_pipeline('text')` sends a text string for sentiment analysis. The actual report would likely process a list of texts or iterate through a dataset.
- The pipeline returns a list for each input text.
- Each list element is a dictionary containing:
 - `label`: Predicted sentiment category (likely "POSITIVE" or "NEGATIVE").
 - `score`: Confidence score associated with the predicted sentiment (between 0 and 1).

Benefits of Pipeline API:

- Simplifies using pre-trained models for sentiment analysis.
- Reduces boilerplate code for model loading and preprocessing.
- Provides confidence scores along with sentiment labels.

9.Conclusion and Future Work:

Summary of Findings:

The project successfully demonstrated the application of various sentiment analysis techniques, including VADER and Roberta, to a dataset of customer reviews. The use of these methods provided deep insights into the emotional undertones of the text data, revealing patterns and nuances in customer sentiment across different review scores.

Due to limited computational resources, the logistic regression model was initially trained on only thousands of rows instead of the entire dataset. If we aim to train it on the full dataset, using a high-end GPU would be necessary. High-end GPUs can significantly enhance computational efficiency, allowing for the processing of larger datasets and more complex calculations. This would not only improve the model's accuracy by providing a more comprehensive understanding of the data but also boost its predictive performance across more varied scenarios.

Key findings include:

- **VADER Analysis:** VADER, a lexicon-based approach, was effective for quick sentiment assessments and performed well with explicit sentiment expressions. However, it sometimes lacked context sensitivity which is crucial for nuanced language understanding.
- **Roberta Analysis:** The Roberta model, leveraging a more complex neural network architecture, showed superior performance in understanding context and subtleties in language, leading to more accurate sentiment predictions.
- **Comparative Analysis:** The comparison between VADER and Roberta highlighted the strengths and weaknesses of both models. While VADER is fast and efficient for straightforward tasks, Roberta excels in complex scenarios where context and detailed linguistic understanding are necessary.

- **Model Evaluation:** Both models were evaluated for accuracy, with Roberta generally outperforming VADER due to its advanced capabilities in handling context.

Recommendations for Future Research:

- **Integration of Multimodal Data:** Future studies could explore the integration of multimodal data sources such as video and audio to enhance the sentiment analysis framework. This could provide a more holistic view of sentiment and emotion.
- **Cross-Lingual Capabilities:** Expanding the sentiment analysis to include multilingual datasets could broaden the applicability of the findings, especially in globalized market settings.
- **Advanced Neural Network Models:** Investigating other advanced neural network models like BERT or GPT-3 could offer improvements over Roberta in certain aspects, particularly in generating context-aware embeddings.
- **Real-Time Analysis:** Developing a framework for real-time sentiment analysis could be beneficial for applications that require immediate feedback, such as customer service bots and social media monitoring.
- **Ethical Considerations and Bias Mitigation:** Further research should also focus on the ethical aspects of sentiment analysis and strategies to mitigate bias in sentiment prediction models.

Conclusion:

This project underscores the dynamic capabilities of sentiment analysis in extracting meaningful insights from textual data. As technology evolves, so too will the methodologies for understanding human emotions through language. The continuous improvement of these analytical tools will undoubtedly enhance their effectiveness and applicability across diverse domains.

Due to limited computational resources, the logistic regression model was initially trained on only thousands of rows instead of the entire dataset. If we aim to train it on the full dataset, using a high-end GPU would be necessary. High-end GPUs can significantly enhance computational efficiency, allowing for the processing of larger datasets and more complex calculations. This would not only improve the model's accuracy by providing a more comprehensive understanding of the data but also boost its predictive performance across more varied scenarios.