


```
# IMPORTANT: RUN THIS CELL IN ORDER TO IMPORT YOUR KAGGLE DATA SOURCES,
# THEN FEEL FREE TO DELETE THIS CELL.
# NOTE: THIS NOTEBOOK ENVIRONMENT DIFFERS FROM KAGGLE'S PYTHON
# ENVIRONMENT SO THERE MAY BE MISSING LIBRARIES USED BY YOUR
# NOTEBOOK.
import kagglehub
kritanjalijain_amazon_reviews_path = kagglehub.dataset_download('kritanjalijain/amazon-reviews')
yasserh_amazon_product_reviews_dataset_path = kagglehub.dataset_download('yasserh/amazon-product-reviews-dataset')
karkavelrajaj_amazon_sales_dataset_path = kagglehub.dataset_download('karkavelrajaj/amazon-sales-dataset')

print('Data source import complete.')
```

 /usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell` and should_run_async(code)

Warning: Looks like you're using an outdated `kagglehub` version, please consider updating (latest version: 0.3.4)

Warning: Looks like you're using an outdated `kagglehub` version, please consider updating (latest version: 0.3.4)

Warning: Looks like you're using an outdated `kagglehub` version, please consider updating (latest version: 0.3.4)

Data source import complete.

Machine learning - NLP Analysis Reviews



✓ Part 1 - Business Problem Classifying Topics in Amazon Product Reviews

Objective: Amazon has a large volume of customer reviews for various products. These reviews contain valuable information about customer experiences, which can be useful for improving product search and providing better recommendations. However, the massive volume of data makes it challenging to manually extract the main topics from the reviews. The goal is to create a Topic Modeling model that automatically identifies and classifies the main topics within the product reviews.

Project Objectives:

1. Dataset Exploration and Data Cleaning:

- Remove null and inconsistent values.
- Check for duplicates in the data.
- Perform text preprocessing, such as removing stopwords, tokenization, and applying stemming or lemmatization.

2. Topic Modeling:

- Apply *Topic Modeling* techniques such as *Latent Dirichlet Allocation (LDA)* or *Non-Negative Matrix Factorization (NMF)* to identify the main topics in the reviews.
- Test different numbers of topics and evaluate the quality of the identified topics.
- Identify the most representative words for each topic.

3. Topic Classification:

- Once the topics are identified, classify the reviews into different categories to facilitate search.

4. Evaluation:

- Assess the coherence of the generated topics using topic coherence metrics.
- Interpret and label the topics meaningfully to improve visualization and search.

5. Results and Impact:

- Provide a set of categories or topics that help users quickly filter reviews based on discussed themes (product quality, shipping, customer support, etc.).
- Implement the model in a way that it can be applied to new data automatically.

Success Indicators:

- Topic coherence (using topic coherence metrics).
- Quality of the review categorization.
- Ease of navigation through the generated topics.

```
# Installing packages
!pip install watermark
!pip install vaderSentiment
!pip install imbalanced-learn
```

```
→ /usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell`
and should_run_async(code)
Requirement already satisfied: watermark in /usr/local/lib/python3.10/dist-packages (2.5.0)
Requirement already satisfied: ipython>=6.0 in /usr/local/lib/python3.10/dist-packages (from watermark) (7.34.0)
Requirement already satisfied: importlib-metadata>=1.4 in /usr/local/lib/python3.10/dist-packages (from watermark) (8.5.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from watermark) (75.1.0)
Requirement already satisfied: zipp>=3.20 in /usr/local/lib/python3.10/dist-packages (from importlib-metadata>=1.4->watermark) (3.20.2)
Requirement already satisfied: jedi>=0.16 in /usr/local/lib/python3.10/dist-packages (from ipython>=6.0->watermark) (0.19.2)
Requirement already satisfied: decorator in /usr/local/lib/python3.10/dist-packages (from ipython>=6.0->watermark) (4.4.2)
Requirement already satisfied: pickleshare in /usr/local/lib/python3.10/dist-packages (from ipython>=6.0->watermark) (0.7.5)
Requirement already satisfied: traitlets>=4.2 in /usr/local/lib/python3.10/dist-packages (from ipython>=6.0->watermark) (5.7.1)
Requirement already satisfied: prompt-toolkit!=3.0.0,!<3.0.1,<3.1.0,>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from ipython>=6.0->watermark) (3.0.43)
Requirement already satisfied: pygments in /usr/local/lib/python3.10/dist-packages (from ipython>=6.0->watermark) (2.18.0)
Requirement already satisfied: backcall in /usr/local/lib/python3.10/dist-packages (from ipython>=6.0->watermark) (0.2.0)
Requirement already satisfied: matplotlib-inline in /usr/local/lib/python3.10/dist-packages (from ipython>=6.0->watermark) (0.1.7)
Requirement already satisfied: pexpect>4.3 in /usr/local/lib/python3.10/dist-packages (from ipython>=6.0->watermark) (4.9.0)
Requirement already satisfied: parso<0.9.0,>=0.8.4 in /usr/local/lib/python3.10/dist-packages (from jedi>=0.16->ipython>=6.0->watermark) (0.8.4)
Requirement already satisfied: ptyprocess>=0.5 in /usr/local/lib/python3.10/dist-packages (from pexpect>4.3->ipython>=6.0->watermark) (0.7.0)
Requirement already satisfied: wcwidth in /usr/local/lib/python3.10/dist-packages (from prompt-toolkit!=3.0.0,!<3.0.1,<3.1.0,>=2.0.0->ipython>=6.0->watermark) (0.2.9)
Requirement already satisfied: vaderSentiment in /usr/local/lib/python3.10/dist-packages (3.3.2)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from vaderSentiment) (2.32.3)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->vaderSentiment) (3.4.0)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->vaderSentiment) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->vaderSentiment) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->vaderSentiment) (2024.8.30)
Requirement already satisfied: imbalanced-learn in /usr/local/lib/python3.10/dist-packages (0.12.4)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn) (1.26.4)
Requirement already satisfied: scipy>=1.5.0 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn) (1.13.1)
Requirement already satisfied: scikit-learn>=1.0.2 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn) (1.5.2)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from imbalanced-learn) (3.5.0)
```

```
# !pip install datasets
```

```
→ /usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell`
and should_run_async(code)
```

```
# Import of libraries

# System libraries
import re
import unicodedata
import itertools
from datasets import Dataset
```

```
# Library for file manipulation
import pandas as pd
import numpy as np
import pandas
```

```

# Data visualization
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib as m
import matplotlib as mpl
import matplotlib.pyplot as plt
import plotly.express as px
from matplotlib import pyplot as plt

# ML NLP
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer

### Download the necessary resources from nltk (tokenizers and stopwords corpus)
# Punkt tokenizer for word tokenization
nltk.download('punkt')

# Stopwords list in multiple languages
nltk.download('stopwords')

# Configuration for graph width and layout
sns.set_theme(style='whitegrid')
palette='viridis'

# Warnings remove alerts
import warnings
warnings.filterwarnings("ignore")

# Python version
from platform import python_version
print('Python version in this Jupyter Notebook:', python_version())

# Load library versions
import watermark

# Library versions
%reload_ext watermark
%watermark -a "Library versions" --iversions

/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell`
and should_run_async(code)
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data] Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data] Package stopwords is already up-to-date!
Python version in this Jupyter Notebook: 3.10.12
Author: Library versions


sklearn      : 1.5.2
catboost     : 1.2.7
numpy        : 1.26.4
watermark    : 2.5.0
nltk         : 3.9.1
pyLDAvis     : 3.4.1
wordcloud    : 1.9.3
matplotlib   : 3.8.0
kagglehub    : 0.3.3
spacy        : 3.7.5
lightgbm     : 4.5.0
vaderSentiment: 3.3.2
seaborn      : 0.13.2
plotly       : 5.24.1
platform     : 1.0.8
datasets     : 3.1.0
re           : 2.2.1
xgboost      : 2.1.2
gensim       : 4.3.3
pandas       : 2.2.2

```

✓ Part 2 - Database


```
# Database
pd.set_option('display.max_columns', None)
df = pd.read_csv("7817_1.csv")
```

```
# Viewing dataset
df.head()
```



| | id | asins | brand | categories | colors | dateAdded | dateUpdated | dimension | ean | |
|---|----------------------|------------|--------|----------------------------|--------|----------------------|----------------------|--------------------------|-----|----------------------|
| 0 | AVpe7AsMilAPnD_xQ78G | B00QJDU3KY | Amazon | Amazon Devices,mazon.co.uk | NaN | 2016-03-08T20:21:53Z | 2017-07-18T23:52:58Z | 169 mm x 117 mm x 9.1 mm | NaN | kindlepaperwhite/b00 |
| 1 | AVpe7AsMilAPnD_xQ78G | B00QJDU3KY | Amazon | Amazon Devices,mazon.co.uk | NaN | 2016-03-08T20:21:53Z | 2017-07-18T23:52:58Z | 169 mm x 117 mm x 9.1 mm | NaN | kindlepaperwhite/b00 |
| 2 | AVpe7AsMilAPnD_xQ78G | B00QJDU3KY | Amazon | Amazon Devices,mazon.co.uk | NaN | 2016-03-08T20:21:53Z | 2017-07-18T23:52:58Z | 169 mm x 117 mm x 9.1 mm | NaN | kindlepaperwhite/b00 |
| 3 | AVpe7AsMilAPnD_xQ78G | B00QJDU3KY | Amazon | Amazon Devices,mazon.co.uk | NaN | 2016-03-08T20:21:53Z | 2017-07-18T23:52:58Z | 169 mm x 117 mm x 9.1 mm | NaN | kindlepaperwhite/b00 |
| 4 | AVpe7AsMilAPnD_xQ78G | B00QJDU3KY | Amazon | Amazon Devices,mazon.co.uk | NaN | 2016-03-08T20:21:53Z | 2017-07-18T23:52:58Z | 169 mm x 117 mm x 9.1 mm | NaN | kindlepaperwhite/b00 |

```
# Viewing first 5 data
df.head()
```



| | id | asins | brand | categories | colors | dateAdded | dateUpdated | dimension | ean | |
|---|----------------------|------------|--------|----------------------------|--------|----------------------|----------------------|--------------------------|-----|----------------------|
| 0 | AVpe7AsMilAPnD_xQ78G | B00QJDU3KY | Amazon | Amazon Devices,mazon.co.uk | NaN | 2016-03-08T20:21:53Z | 2017-07-18T23:52:58Z | 169 mm x 117 mm x 9.1 mm | NaN | kindlepaperwhite/b00 |
| 1 | AVpe7AsMilAPnD_xQ78G | B00QJDU3KY | Amazon | Amazon Devices,mazon.co.uk | NaN | 2016-03-08T20:21:53Z | 2017-07-18T23:52:58Z | 169 mm x 117 mm x 9.1 mm | NaN | kindlepaperwhite/b00 |
| 2 | AVpe7AsMilAPnD_xQ78G | B00QJDU3KY | Amazon | Amazon Devices,mazon.co.uk | NaN | 2016-03-08T20:21:53Z | 2017-07-18T23:52:58Z | 169 mm x 117 mm x 9.1 mm | NaN | kindlepaperwhite/b00 |
| 3 | AVpe7AsMilAPnD_xQ78G | B00QJDU3KY | Amazon | Amazon Devices,mazon.co.uk | NaN | 2016-03-08T20:21:53Z | 2017-07-18T23:52:58Z | 169 mm x 117 mm x 9.1 mm | NaN | kindlepaperwhite/b00 |
| 4 | AVpe7AsMilAPnD_xQ78G | B00QJDU3KY | Amazon | Amazon Devices,mazon.co.uk | NaN | 2016-03-08T20:21:53Z | 2017-07-18T23:52:58Z | 169 mm x 117 mm x 9.1 mm | NaN | kindlepaperwhite/b00 |

```
# Viewing 5 latest data
df.tail()
```



| | | id | asins | brand | categories | colors | dateAdded | dateUpdated | dimension | ean |
|------|----------------------|------------|--------|---|------------|----------------------|----------------------|-------------|-----------|-----------------------|
| 1592 | AVpfo9ukilAPnD_xfhuj | B00NO8JJZW | Amazon | Amazon Devices & Accessories,Amazon Device Acc... | NaN | 2016-04-02T14:40:43Z | 2017-08-13T08:28:46Z | NaN | NaN | alexavoiceremoteforai |
| 1593 | AVpfo9ukilAPnD_xfhuj | B00NO8JJZW | Amazon | Amazon Devices & Accessories,Amazon Device Acc... | NaN | 2016-04-02T14:40:43Z | 2017-08-13T08:28:46Z | NaN | NaN | alexavoiceremoteforai |
| 1594 | AVpfo9ukilAPnD_xfhuj | B00NO8JJZW | Amazon | Amazon Devices & Accessories,Amazon Device Acc... | NaN | 2016-04-02T14:40:43Z | 2017-08-13T08:28:46Z | NaN | NaN | alexavoiceremoteforai |
| 1595 | AVpfo9ukilAPnD_xfhuj | B00NO8JJZW | Amazon | Amazon Devices & Accessories,Amazon Device Acc... | NaN | 2016-04-02T14:40:43Z | 2017-08-13T08:28:46Z | NaN | NaN | alexavoiceremoteforai |
| 1596 | AVpfo9ukilAPnD_xfhuj | B00NO8JJZW | Amazon | Amazon Devices & Accessories,Amazon Device Acc... | NaN | 2016-04-02T14:40:43Z | 2017-08-13T08:28:46Z | NaN | NaN | alexavoiceremoteforai |

```
# Info data
df.info()
```



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1597 entries, 0 to 1596
Data columns (total 27 columns):
#   Column                Non-Null Count  Dtype
---  -
0   id                     1597 non-null  object
1   asins                  1597 non-null  object
2   brand                  1597 non-null  object
3   categories             1597 non-null  object
4   colors                  774 non-null   object
5   dateAdded              1597 non-null  object
6   dateUpdated            1597 non-null  object
7   dimension              565 non-null   object
8   ean                    898 non-null   float64
9   keys                   1597 non-null  object
10  manufacturer            965 non-null   object
11  manufacturerNumber      902 non-null   object
12  name                    1597 non-null  object
13  prices                  1597 non-null  object
14  reviews.date            1217 non-null  object
15  reviews.doRecommend    539 non-null   object
16  reviews.numHelpful      900 non-null   float64
17  reviews.rating          1177 non-null  float64
18  reviews.sourceURLs      1597 non-null  object
19  reviews.text            1597 non-null  object
20  reviews.title           1580 non-null  object
21  reviews.userCity        0 non-null     float64
22  reviews.userProvince    0 non-null     float64
23  reviews.username        1580 non-null  object
24  sizes                    0 non-null     float64
25  upc                     898 non-null   float64
26  weight                  686 non-null   object
```

```
dtypes: float64(7), object(20)
memory usage: 337.0+ KB
```

```
# Type data
df.dtypes
```

```
↗
```

| | 0 |
|-----------------------------|---------|
| id | object |
| asins | object |
| brand | object |
| categories | object |
| colors | object |
| dateAdded | object |
| dateUpdated | object |
| dimension | object |
| ean | float64 |
| keys | object |
| manufacturer | object |
| manufacturerNumber | object |
| name | object |
| prices | object |
| reviews.date | object |
| reviews.doRecommend | object |
| reviews.numHelpful | float64 |
| reviews.rating | float64 |
| reviews.sourceURLs | object |
| reviews.text | object |
| reviews.title | object |
| reviews.userCity | float64 |
| reviews.userProvince | float64 |
| reviews.username | object |
| sizes | float64 |
| upc | float64 |
| weight | object |

```
dtype: object
```

```
# Viewing rows and columns
df.shape
```

```
↗ (1597, 27)
```

```
# Copy data
data = df.copy()
```

✓ Part 3 - Data cleaning

```
print("Checking for missing values in each column:")
print(df.isnull().sum())
```

```
↗ Checking for missing values in each column:
```

| | |
|------------|---|
| id | 0 |
| asins | 0 |
| brand | 0 |
| categories | 0 |

```

colors                823
dateAdded              0
dateUpdated            0
dimension             1032
ean                   699
keys                  0
manufacturer          632
manufacturerNumber    695
name                  0
prices                0
reviews.date          380
reviews.doRecommend   1058
reviews.numHelpful    697
reviews.rating        420
reviews.sourceURLs    0
reviews.text          0
reviews.title         17
reviews.userCity      1597
reviews.userProvince  1597
reviews.username      17
sizes                 1597
upc                   699
weight                911
dtype: int64

```

```

# Remove columns that have any NaN value
df = df.dropna(axis=1)

```

```

# Check remaining columns after removal
print("Remaining columns after removing columns with NaN:")
print(df.isnull().sum())

```

```

Remaining columns after removing columns with NaN:
id                0
asins             0
brand             0
categories        0
dateAdded         0
dateUpdated       0
keys              0
name              0
prices            0
reviews.sourceURLs 0
reviews.text      0
dtype: int64

```

```

# Calculate the total number of rows and the number of missing values in the 'Translated_Review' column.
# Then, print the percentage of missing values in the 'Translated_Review' column.
#total_rows = len(df)
#missing_translated_review = df['reviews.title'].isnull().sum()
#print(f"Percentage of missing Translated_Review: {(missing_translated_review/total_rows)*100:.2f}%")

```

```

# Drop rows that have missing values in the 'Translated_Review' or 'Sentiment' columns
#df = df.dropna(subset=['reviews.title'])

```

```

# Print the count of missing values for each column after dropping the rows with missing data
#print(df.isnull().sum())
#print()

```

```

# Display the DataFrame
#df

```

▼ Part 4 - Text Preprocessing

```

%%time

import re
import string
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer

### Download the necessary resources from nltk (tokenizers and stopwords corpus)
# Punkt tokenizer for word tokenization

```

```

nltk.download('punkt')

# Stopwords list in multiple languages
nltk.download('stopwords')
nltk.download('punkt_tab')

# Initialize the Porter stemmer and load English stopwords
stemmer = PorterStemmer()

# Create a set of English stopwords for efficient lookup
stop_words = set(stopwords.words('english'))

# Function to clean the text by removing URLs, handles, and punctuation
def clean_text(text):
    if isinstance(text, str):
        # Convert text to lowercase
        text = text.lower()

        # Remove URLs (http, https, and www links)
        text = re.sub(r'http\S+|www\S+|https\S+', '', text, flags=re.MULTILINE)

        # Remove markdown-style links [text](link)
        text = re.sub(r'\[.*?\]\(.*?\)', '', text)

        # Remove handles (@username mentions)
        text = re.sub(r'@\w+', '', text)

        # Remove punctuation and special characters
        text = text.translate(str.maketrans('', '', string.punctuation))

    return text
else:
    return text

# Function to tokenize the text into individual words
def tokenize_text(text):
    if isinstance(text, str):
        return word_tokenize(text)
    else:
        return text

# Function to remove stopwords from the tokenized text
def remove_stopwords(tokens):
    if isinstance(tokens, list):
        return [word for word in tokens if word not in stop_words]
    else:
        return tokens

# Function to apply stemming to the tokens
def stem_tokens(tokens):
    if isinstance(tokens, list):
        return [stemmer.stem(token) for token in tokens]
    else:
        return tokens

### Apply the functions to the DataFrame
# Clean the text
df['Cleaned_Review'] = df['reviews.text'].apply(clean_text)

# Tokenize the cleaned text
df['Tokenized_Review'] = df['Cleaned_Review'].apply(tokenize_text)

# Apply stemming to the tokenized words
df['Stemmed_Review'] = df['Tokenized_Review'].apply(stem_tokens)

# Remove stopwords from the tokenized text
df['No_Stopwords_Review'] = df['Tokenized_Review'].apply(remove_stopwords)

# Display the first few rows of the DataFrame to visualize the dataset
df.head()

```



```

[ntlk_data] Downloading package punkt to /root/nltk_data...
[ntlk_data] Package punkt is already up-to-date!
[ntlk_data] Downloading package stopwords to /root/nltk_data...
[ntlk_data] Package stopwords is already up-to-date!
[ntlk_data] Downloading package punkt_tab to /root/nltk_data...
[ntlk_data] Package punkt_tab is already up-to-date!
CPU times: user 10.1 s, sys: 83.1 ms, total: 10.1 s
Wall time: 16.1 s

```

| | id | asins | brand | categories | dateAdded | dateUpdated | keys | name | |
|---|----------------------|------------|--------|-------------------------------|----------------------|----------------------|-----------------------------|----------------------|--------|
| 0 | AVpe7AsMilAPnD_xQ78G | B00QJDU3KY | Amazon | Amazon Devices,mazon.co.uk | 2016-03-08T20:21:53Z | 2017-07-18T23:52:58Z | kindlepaperwhite/b00qjdu3ky | Kindle Paperwhite | {{"amo |
| 1 | AVpe7AsMilAPnD_xQ78G | B00QJDU3KY | Amazon | Amazon Devices,mazon.co.uk | 2016-03-08T20:21:53Z | 2017-07-18T23:52:58Z | kindlepaperwhite/b00qjdu3ky | Kindle Paperwhite | {{"amo |
| 2 | AVpe7AsMilAPnD_xQ78G | B00QJDU3KY | Amazon | Amazon Devices,mazon.co.uk | 2016-03-08T20:21:53Z | 2017-07-18T23:52:58Z | kindlepaperwhite/b00qjdu3ky | Kindle Paperwhite | {{"amo |
| 3 | AVpe7AsMilAPnD_xQ78G | B00QJDU3KY | Amazon | Amazon Devices,mazon.co.uk | 2016-03-08T20:21:53Z | 2017-07-18T23:52:58Z | kindlepaperwhite/b00qjdu3ky | Kindle Paperwhite | {{"amo |
| 4 | AVpe7AsMilAPnD_xQ78G | B00QJDU3KY | Amazon | Amazon Devices,mazon.co.uk | 2016-03-08T20:21:53Z | 2017-07-18T23:52:58Z | kindlepaperwhite/b00qjdu3ky | Kindle Paperwhite | {{"amo |

Here, we performed text preprocessing, which included several important steps. First, we carried out **text cleaning**, removing special characters such as punctuation, symbols, and irrelevant numbers to ensure that the model would not be influenced by noise. Additionally, we standardized the text by applying techniques such as converting to lowercase and removing extra spaces.

The second part of the preprocessing involved **text tokenization**, where the content was segmented into individual words (tokens). This step is crucial for further analysis, allowing each word to be handled separately rather than treating the text as a single block. Tokenization also facilitates the conversion of words into numerical representations, a necessary step for training machine learning and deep learning models.

If needed, other preprocessing techniques can be applied, such as removing stopwords (common words that add little meaning, like "the", "and", "of"), stemming or lemmatization (reducing words to their root forms), and the creation of n-grams to capture word combinations.

✓ Part 4.1 - Sentiment Analysis with Vader

```

from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer

# Create an instance of SentimentIntensityAnalyzer, which is part of the VADER sentiment analysis tool.
analyzer = SentimentIntensityAnalyzer()

# Function to apply VADER sentiment analysis to each review.
# The function takes in a 'review' (text) and returns the polarity scores, which include
# 'compound', 'positive', 'negative', and 'neutral' sentiment values.
def get_sentiment(review):
    return analyzer.polarity_scores(review)

# Apply the 'get_sentiment' function to each entry in the 'Cleaned_Review' column.
# Store the resulting sentiment scores (dictionary) in a new column 'vader_scores'.
df['vader_scores'] = df['Cleaned_Review'].apply(get_sentiment)

# Extract the 'compound' score from the 'vader_scores' dictionary for each review.
# The 'compound' score is a normalized score ranging from -1 (most negative) to +1 (most positive),
# summarizing the overall sentiment of the text.
df['compound'] = df['vader_scores'].apply(lambda score_dict: score_dict['compound'])

# Function to classify the sentiment based on the 'compound' score.
# Scores >= 0.05 are classified as 'Positive', <= -0.05 as 'Negative', and anything in between as 'Neutral'.
def classify_sentiment(compound_score):
    if compound_score >= 0.05:

```

```

        return 'Positive'
    elif compound_score <= -0.05:
        return 'Negative'
    else:
        return 'Neutral'

# Apply the 'classify_sentiment' function to the 'compound' score to get the overall sentiment
# classification (Positive, Negative, or Neutral) for each review.
df['sentiment'] = df['compound'].apply(classify_sentiment)

# Display the first few rows of the DataFrame, showing the cleaned review text,
# the 'compound' sentiment score, and the classified 'sentiment'.
print(df[['Cleaned_Review', 'compound', 'sentiment']].head(10))

```

```

Cleaned_Review    compound sentiment
0  i initially had trouble deciding between the p...    0.9879  Positive
1  allow me to preface this with a little history...    0.9881  Positive
2  i am enjoying it so far great for reading had ...    0.4364  Positive
3  i bought one of the first paperwhites and have...    0.9746  Positive
4  i have to say upfront i dont like coroporate ...    0.9980  Positive
5  my previous kindle was a dx this is my second ...    0.2627  Positive
6  allow me to preface this with a little history...    0.9881  Positive
7  just got mine right now looks the same as the ...    0.9205  Positive
8  i initially had trouble deciding between the p...    0.9879  Positive
9  i am enjoying it so far great for reading had ...    0.4364  Positive

```

In this script, we apply the VADER (Valence Aware Dictionary and sEntiment Reasoner) sentiment analysis tool to classify the sentiment of textual reviews. VADER is a popular rule-based model used to analyze the sentiment expressed in text, specifically designed to handle social media, but applicable to any type of textual data.

- **Sentiment Analysis with VADER:** We start by initializing an instance of `SentimentIntensityAnalyzer`, which computes the polarity of a given text. VADER calculates four sentiment scores for each review: positive, negative, neutral, and a compound score. The compound score represents an overall sentiment, normalized between -1 (extremely negative) to +1 (extremely positive).
- **Extracting the Compound Score:** After applying VADER to each review in the `Cleaned_Review` column, we store the results in a new column `vader_scores`. From this dictionary, we specifically extract the `compound` score, which simplifies the task of classifying the overall sentiment into 'Positive', 'Negative', or 'Neutral' based on defined thresholds.
- **Classifying Sentiments:** The `classify_sentiment` function uses the `compound` score to categorize the sentiment. A score greater than or equal to 0.05 indicates a positive sentiment, a score less than or equal to -0.05 indicates a negative sentiment, and anything in between is considered neutral. This classification helps in understanding the emotional tone conveyed in the review text.
- **Final Output:** After applying the sentiment classification, the DataFrame now contains columns for the original cleaned review text, the compound score, and the final `sentiment` label. This allows for easy interpretation of both the numerical sentiment scores and their respective categories (Positive, Neutral, or Negative). We then display the first few entries to visualize the results.

Overall, this script demonstrates a simple and effective way to perform sentiment analysis on text data using VADER, allowing for automated categorization of emotional tone in customer reviews, feedback, or any other form of written text."

✓ Part 5 - Exploratory data analysis

```

# Assuming df is already loaded and contains the 'sentiment' column
plt.figure(figsize=(8, 6))

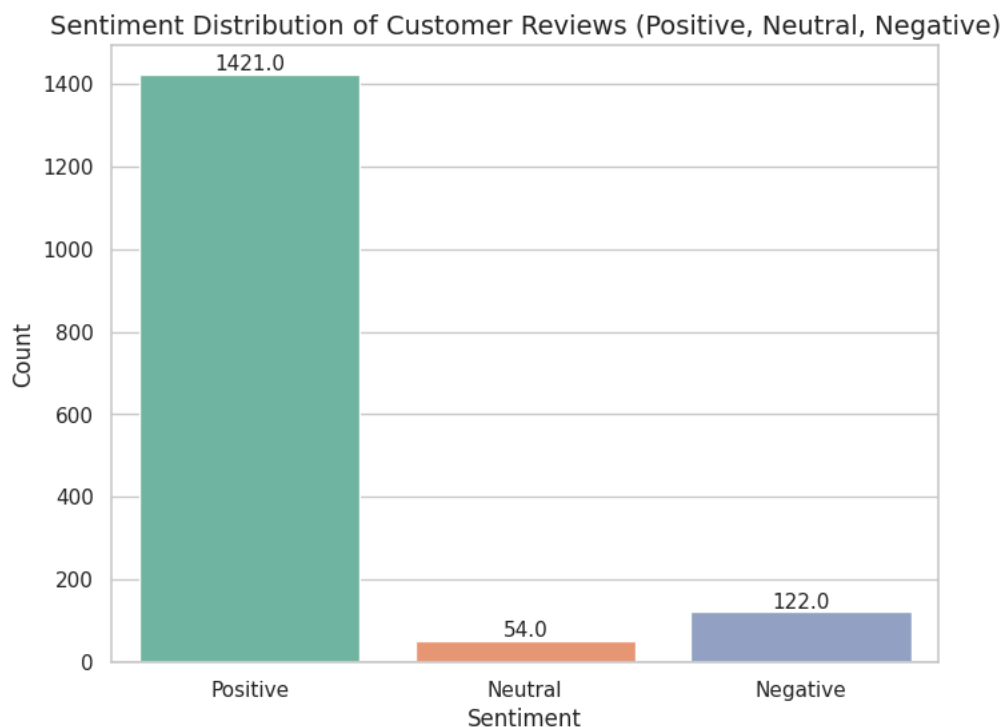
# Count plot with a specific color palette
sns.countplot(data=df, x='sentiment', palette='Set2')

# Add title and axis labels with adjusted font sizes
plt.title('Sentiment Distribution of Customer Reviews (Positive, Neutral, Negative)', fontsize=14)
plt.xlabel('Sentiment', fontsize=12)
plt.ylabel('Count', fontsize=12)

# Adding data labels on top of the bars
for p in plt.gca().patches:
    plt.gca().annotate(f'{p.get_height()}', (p.get_x() + p.get_width() / 2, p.get_height()),
                      ha='center', va='bottom', fontsize=11)

# Display the plot
plt.show()

```

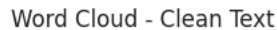


Analysis

1. **Positive Sentiment Dominance:** - The overwhelming majority of reviews are classified as **Positive** (1,421 counts). This suggests that customers are generally satisfied or expressing favorable opinions in the reviews.
2. **Neutral Sentiment:** - There is a significantly lower number of reviews classified as **Neutral** (54 counts). This means that few reviews are neither strongly positive nor negative, showing a lack of middle-ground feedback.
3. **Negative Sentiment:** - A small portion of reviews is classified as **Negative** (122 counts). While fewer than positive reviews, this indicates that some customers are sharing dissatisfaction or negative experiences.
4. **Visual Representation:** - The **bar plot** clearly highlights the disproportionate distribution of sentiments, where **positive reviews far outnumber** both negative and neutral sentiments.
 - **Labeling** is well done, with the exact count numbers annotated on top of each bar, allowing for easy understanding of the data.
5. **Improvements in Balance:** - If the dataset aims to represent a balanced sentiment analysis, it might be necessary to gather or balance more negative or neutral reviews to avoid the positive sentiment dominance. Alternatively, such an imbalance could be a reflection of a generally positive customer experience.

```
from wordcloud import WordCloud

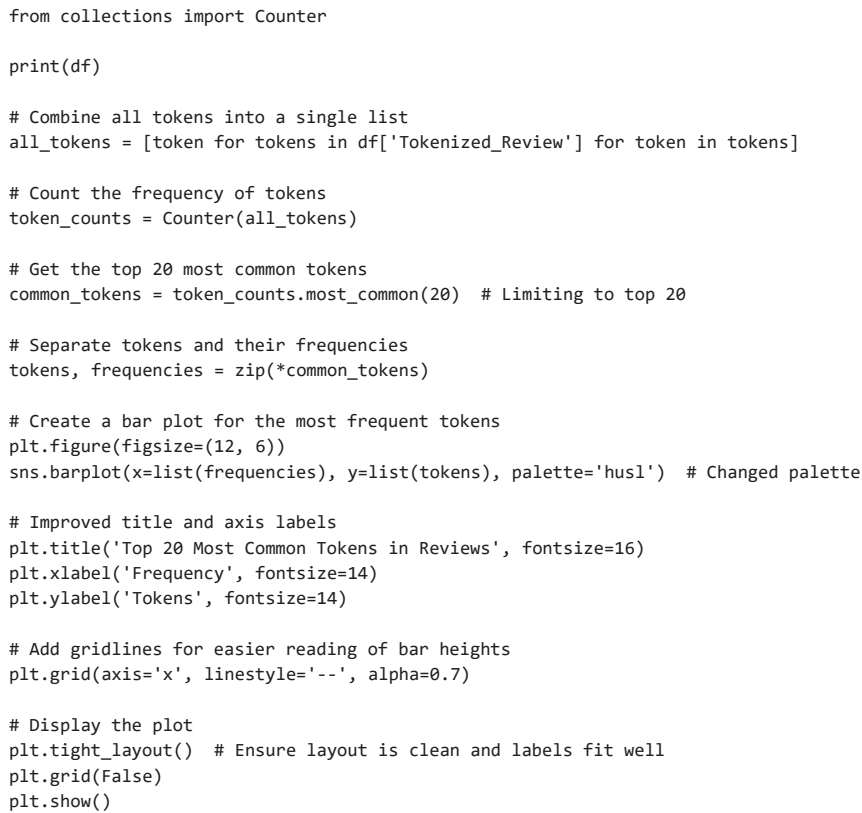
wordcloud = WordCloud(width=800, height=400,
                       background_color='white').generate(' '.join(df['Cleaned_Review'].dropna()))
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title("Word Cloud - Clean Text")
plt.show()
```



<https://colab.research.google.com/#fileId=https%3A//storage.googleapis.com/kaggle-colab-exported-notebooks/model-nlp-lda-reviews-amazon-1bcf...> 12/56



A word cloud visualization of the text "The last year of the fire hose". The words are arranged in a dense, overlapping manner, with colors ranging from dark purple to light green. The most prominent words, shown in the largest font sizes, are "last", "year", "fire", "hose", "review", "price", "ipad", "app", "kindle", "hdx", "game", "will", "work", "laptop", "button", "man", "killer", "business", "purposes", "put", "hd", "video", "processor", "quad", "core", "improved", "vs", "pne", "make", "display", "check", "out", "as", "leaps", "problem", "watching", "movies", "background", "day", "able", "22ghz", "quad", "day", "background", "able", "22ghz", "quad", "day", "background", "able", "22ghz", "quad". Other visible words include "netflix", "review", "please", "search", "following", "and/or", "new", "hd", "hdx", "tablet", "enlarged", "version", "hour", "used", "even", "simply", "put", "table", "user", "ongoing", "review", "difference", "using", "need", "music", "secret", "something", "important", "things", "personal", "use", "last", "hope", "bought", "personal", "use", "man", "air", "killer", "button", "game", "business", "purposes", "put", "hd", "video", "processor", "quad", "core", "improved", "vs", "pne", "make", "display", "check", "out", "as", "leaps", "problem", "watching", "movies", "background", "day", "able", "22ghz", "quad", "day", "background", "able", "22ghz", "quad".



```

id      asins      brand \
0      AVpe7AsMilAPnD_xQ78G B00QJDU3KY Amazon
1      AVpe7AsMilAPnD_xQ78G B00QJDU3KY Amazon
2      AVpe7AsMilAPnD_xQ78G B00QJDU3KY Amazon
3      AVpe7AsMilAPnD_xQ78G B00QJDU3KY Amazon
4      AVpe7AsMilAPnD_xQ78G B00QJDU3KY Amazon
...
1592   AVpfo9ukilAPnD_xfhuj B00N08JJZW Amazon
1593   AVpfo9ukilAPnD_xfhuj B00N08JJZW Amazon
1594   AVpfo9ukilAPnD_xfhuj B00N08JJZW Amazon
1595   AVpfo9ukilAPnD_xfhuj B00N08JJZW Amazon
1596   AVpfo9ukilAPnD_xfhuj B00N08JJZW Amazon

categories      dateAdded \
0      Amazon Devices,mazon.co.uk 2016-03-08T20:21:53Z
1      Amazon Devices,mazon.co.uk 2016-03-08T20:21:53Z
2      Amazon Devices,mazon.co.uk 2016-03-08T20:21:53Z
3      Amazon Devices,mazon.co.uk 2016-03-08T20:21:53Z
4      Amazon Devices,mazon.co.uk 2016-03-08T20:21:53Z
...
1592   Amazon Devices & Accessories,Amazon Device Acc... 2016-04-02T14:40:43Z
1593   Amazon Devices & Accessories,Amazon Device Acc... 2016-04-02T14:40:43Z
1594   Amazon Devices & Accessories,Amazon Device Acc... 2016-04-02T14:40:43Z
1595   Amazon Devices & Accessories,Amazon Device Acc... 2016-04-02T14:40:43Z
1596   Amazon Devices & Accessories,Amazon Device Acc... 2016-04-02T14:40:43Z

dateUpdated      keys \
0      2017-07-18T23:52:58Z kindlepaperwhite/b00qjdu3ky
1      2017-07-18T23:52:58Z kindlepaperwhite/b00qjdu3ky
2      2017-07-18T23:52:58Z kindlepaperwhite/b00qjdu3ky
3      2017-07-18T23:52:58Z kindlepaperwhite/b00qjdu3ky
4      2017-07-18T23:52:58Z kindlepaperwhite/b00qjdu3ky
...
1592   2017-08-13T08:28:46Z alexavoiceremoteforamazonfiretvfiretvstick/b00...
1593   2017-08-13T08:28:46Z alexavoiceremoteforamazonfiretvfiretvstick/b00...
1594   2017-08-13T08:28:46Z alexavoiceremoteforamazonfiretvfiretvstick/b00...
1595   2017-08-13T08:28:46Z alexavoiceremoteforamazonfiretvfiretvstick/b00...
1596   2017-08-13T08:28:46Z alexavoiceremoteforamazonfiretvfiretvstick/b00...

name \
0      Kindle Paperwhite
1      Kindle Paperwhite
2      Kindle Paperwhite
3      Kindle Paperwhite
4      Kindle Paperwhite
...
1592   Alexa Voice Remote for Amazon Fire TV and Fire...
1593   Alexa Voice Remote for Amazon Fire TV and Fire...
1594   Alexa Voice Remote for Amazon Fire TV and Fire...
1595   Alexa Voice Remote for Amazon Fire TV and Fire...
1596   Alexa Voice Remote for Amazon Fire TV and Fire...

prices \
0      [{"amountMax":139.99,"amountMin":139.99,"curre...
1      [{"amountMax":139.99,"amountMin":139.99,"curre...
2      [{"amountMax":139.99,"amountMin":139.99,"curre...
3      [{"amountMax":139.99,"amountMin":139.99,"curre...
4      [{"amountMax":139.99,"amountMin":139.99,"curre...
...
1592   [{"amountMax":29.99,"amountMin":29.99,"currenc...
1593   [{"amountMax":29.99,"amountMin":29.99,"currenc...
1594   [{"amountMax":29.99,"amountMin":29.99,"currenc...
1595   [{"amountMax":29.99,"amountMin":29.99,"currenc...
1596   [{"amountMax":29.99,"amountMin":29.99,"currenc...

reviews.sourceURLs \
0      https://www.amazon.com/Kindle-Paperwhite-High-...
1      https://www.amazon.com/Kindle-Paperwhite-High-...
2      https://www.amazon.com/Kindle-Paperwhite-High-...
3      https://www.amazon.com/Kindle-Paperwhite-High-...
4      https://www.amazon.com/Kindle-Paperwhite-High-...
...
1592   https://www.amazon.com/Alexa-Voice-Remote-Amaz...
1593   https://www.amazon.com/Alexa-Voice-Remote-Amaz...
1594   https://www.amazon.com/Alexa-Voice-Remote-Amaz...
1595   https://www.amazon.com/Alexa-Voice-Remote-Amaz...
1596   https://www.amazon.com/Alexa-Voice-Remote-Amaz...

reviews.text \
0      I initially had trouble deciding between the p...
1      Allow me to preface this with a little history...
2      I am enjoying it so far. Great for reading. Ha...
3      I bought one of the first Paperwhites and have...
4      I have to say upfront - I don't like coroporat...

```



```
...
1592 This is not the same remote that I got for my ...
1593 I have had to change the batteries in this rem...
1594 Remote did not activate, nor did it connect to...
1595 It does the job but is super over priced. I fe...
1596 I ordered this item to replace the one that no...
```

```
                                Cleaned_Review \
0      i initially had trouble deciding between the p...
1      allow me to preface this with a little history...
2      i am enjoying it so far great for reading had ...
3      i bought one of the first paperwhites and have...
4      i have to say upfront i dont like coroporate ...
...
1592 this is not the same remote that i got for my ...
1593 i have had to change the batteries in this rem...
1594 remote did not activate nor did it connect to ...
1595 it does the job but is super over priced i fee...
1596 i ordered this item to replace the one that no...
```

```
                                Tokenized_Review \
0      [i, initially, had, trouble, deciding, between...
1      [allow, me, to, preface, this, with, a, little...
2      [i, am, enjoying, it, so, far, great, for, rea...
3      [i, bought, one, of, the, first, paperwhites, ...
4      [i, have, to, say, upfront, i, dont, like, cor...
...
1592 [this, is, not, the, same, remote, that, i, go...
1593 [i, have, had, to, change, the, batteries, in,...
1594 [remote, did, not, activate, nor, did, it, con...
1595 [it, does, the, job, but, is, super, over, pri...
1596 [i, ordered, this, item, to, replace, the, one...
```

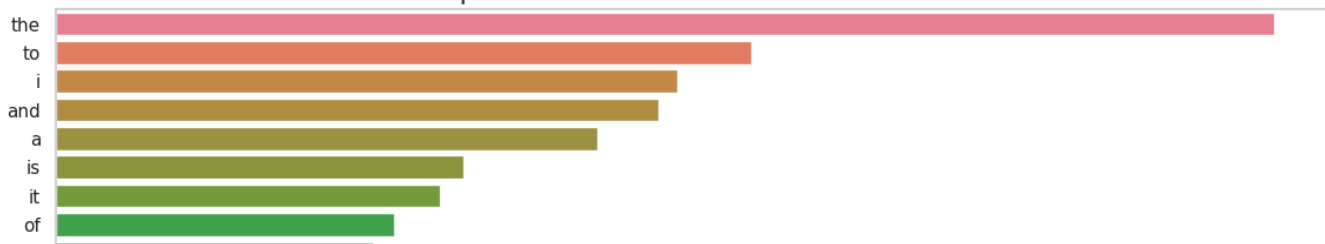
```
                                Stemmed_Review \
0      [i, initi, had, troubl, decid, between, the, p...
1      [allow, me, to, prefac, thi, with, a, littl, h...
2      [i, am, enjoy, it, so, far, great, for, read, ...
3      [i, bought, one, of, the, first, paperwhit, an...
4      [i, have, to, say, upfront, i, dont, like, cor...
...
1592 [thi, is, not, the, same, remot, that, i, got,...
1593 [i, have, had, to, chang, the, batteri, in, th...
1594 [remot, did, not, activ, nor, did, it, connect...
1595 [it, doe, the, job, but, is, super, over, pric...
1596 [i, order, thi, item, to, replac, the, one, th...
```

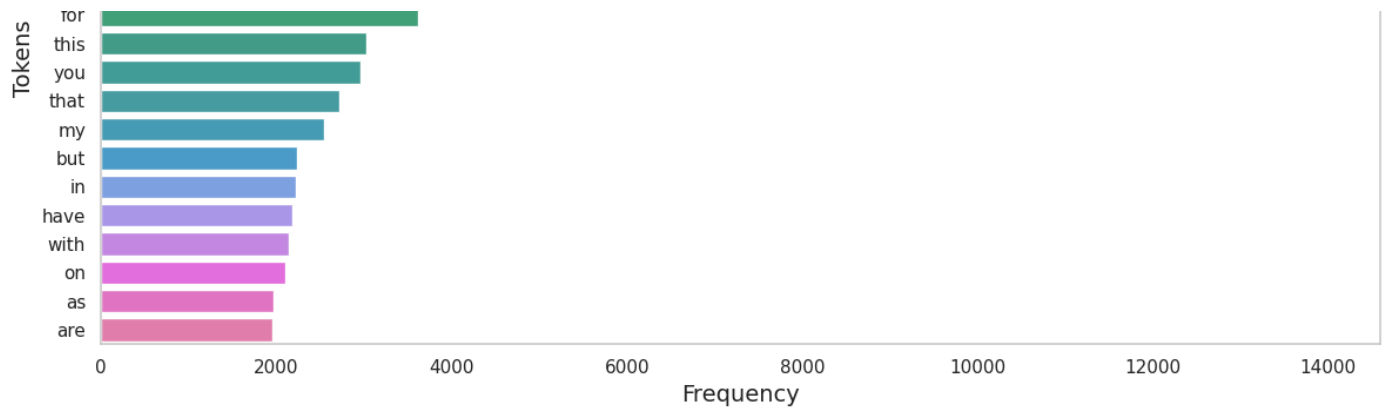
```
                                No_Stopwords_Review \
0      [initially, trouble, deciding, paperwhite, voy...
1      [allow, preface, little, history, casual, read...
2      [enjoying, far, great, reading, original, fire...
3      [bought, one, first, paperwhites, pleased, con...
4      [say, upfront, dont, like, coroporate, hermeti...
...
1592 [remote, got, alexaecho, doesnt, control, volu...
1593 [change, batteries, remote, twice, per, month,...
1594 [remote, activate, connect, boxa, poorly, desi...
1595 [job, super, priced, feel, like, offer, replac...
1596 [ordered, item, replace, one, longer, works, d...
```

| | vader_scores | compound | sentiment |
|------|--|----------|-----------|
| 0 | {'neg': 0.008, 'neu': 0.82, 'pos': 0.171, 'com...} | 0.9879 | Positive |
| 1 | {'neg': 0.033, 'neu': 0.824, 'pos': 0.143, 'co...} | 0.9881 | Positive |
| 2 | {'neg': 0.173, 'neu': 0.613, 'pos': 0.213, 'co...} | 0.4364 | Positive |
| 3 | {'neg': 0.028, 'neu': 0.873, 'pos': 0.099, 'co...} | 0.9746 | Positive |
| 4 | {'neg': 0.03, 'neu': 0.731, 'pos': 0.239, 'com...} | 0.9980 | Positive |
| ... | ... | ... | ... |
| 1592 | {'neg': 0.133, 'neu': 0.796, 'pos': 0.07, 'com...} | -0.6249 | Negative |
| 1593 | {'neg': 0.113, 'neu': 0.86, 'pos': 0.027, 'com...} | -0.9205 | Negative |
| 1594 | {'neg': 0.198, 'neu': 0.802, 'pos': 0.0, 'comp...} | -0.8126 | Negative |
| 1595 | {'neg': 0.1, 'neu': 0.69, 'pos': 0.21, 'compou...} | 0.8271 | Positive |
| 1596 | {'neg': 0.114, 'neu': 0.87, 'pos': 0.016, 'com...} | -0.8630 | Negative |

[1597 rows x 18 columns]

Top 20 Most Common Tokens in Reviews





```
from collections import Counter

# Function to plot the most common words for each sentiment
def plot_most_common_words(common_words, sentiment, color):
    # Unzip the common_words tuple into two lists: words and their respective counts
    words, counts = zip(*common_words)

    # Create a bar chart
    plt.figure(figsize=(10, 6))
    sns.barplot(x=list(counts), y=list(words), palette=color)

    # Set the title and axis labels
    plt.title(f'Most Common Words in {sentiment} Reviews', fontsize=16)
    plt.xlabel('Frequency', fontsize=14)
    plt.ylabel('Words', fontsize=14)

    # Display the chart
    plt.show()

# Generate the most common words for each sentiment

# Filter reviews based on sentiment
positive_reviews = df[df['sentiment'] == 'Positive']['No_Stopwords_Review']
negative_reviews = df[df['sentiment'] == 'Negative']['No_Stopwords_Review']
neutral_reviews = df[df['sentiment'] == 'Neutral']['No_Stopwords_Review']

# Combine all words into a single list for each sentiment
positive_words_list = [word for review in positive_reviews for word in review]
negative_words_list = [word for review in negative_reviews for word in review]
neutral_words_list = [word for review in neutral_reviews for word in review]

# Count the most common words in each sentiment's word list
positive_words = Counter(positive_words_list).most_common(20)
negative_words = Counter(negative_words_list).most_common(20)
neutral_words = Counter(neutral_words_list).most_common(20)

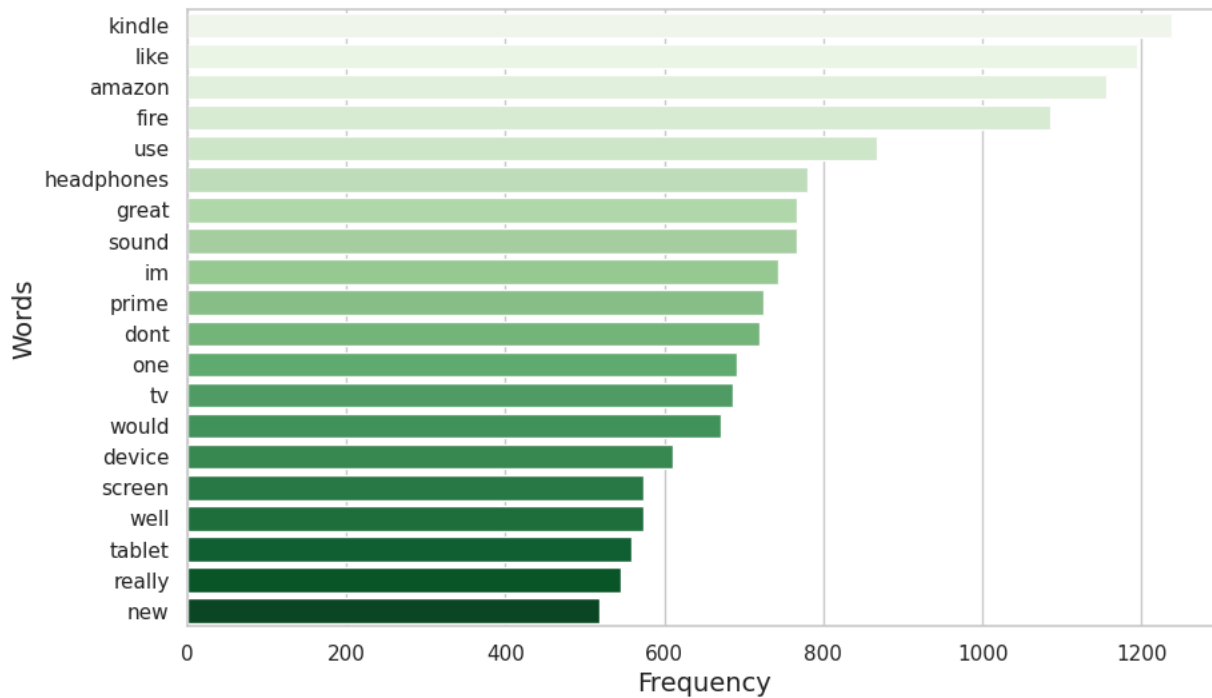
# Plot the most common words for Positive sentiment
plot_most_common_words(positive_words, 'Positive', 'Greens')

# Plot the most common words for Negative sentiment
plot_most_common_words(negative_words, 'Negative', 'Reds')

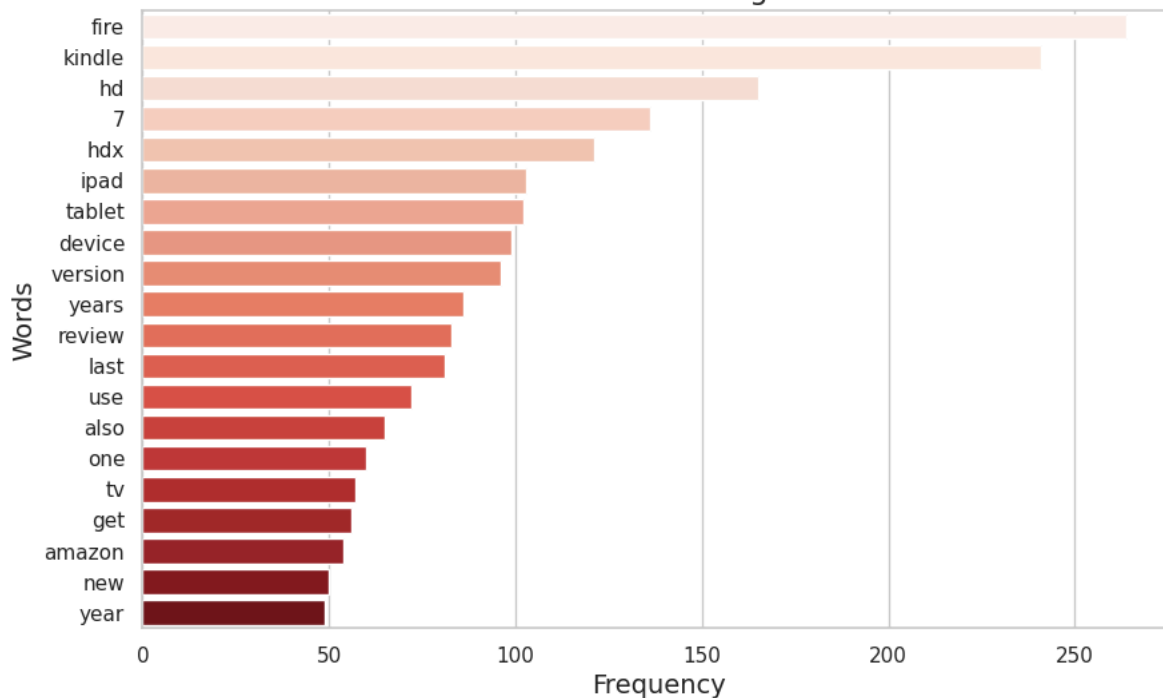
# Plot the most common words for Neutral sentiment
plot_most_common_words(neutral_words, 'Neutral', 'Blues')
```



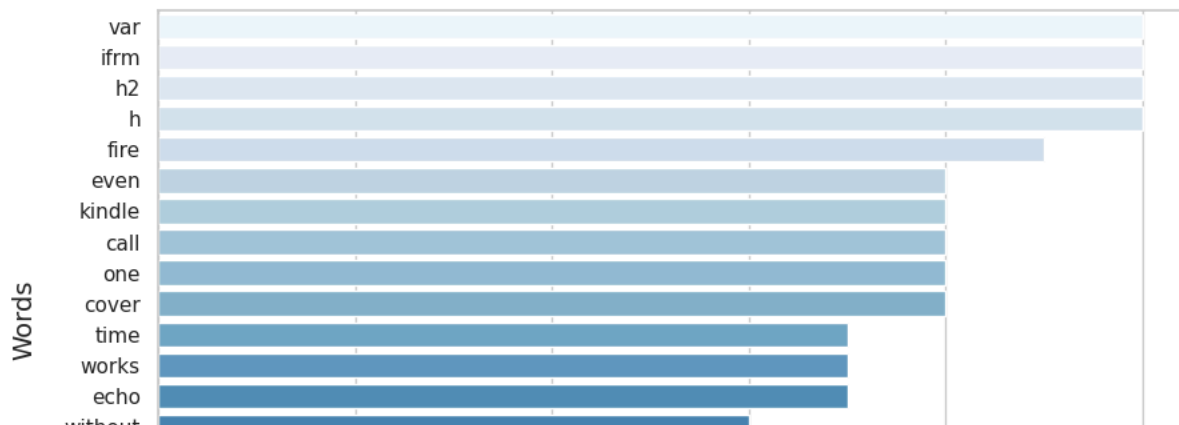
Most Common Words in Positive Reviews

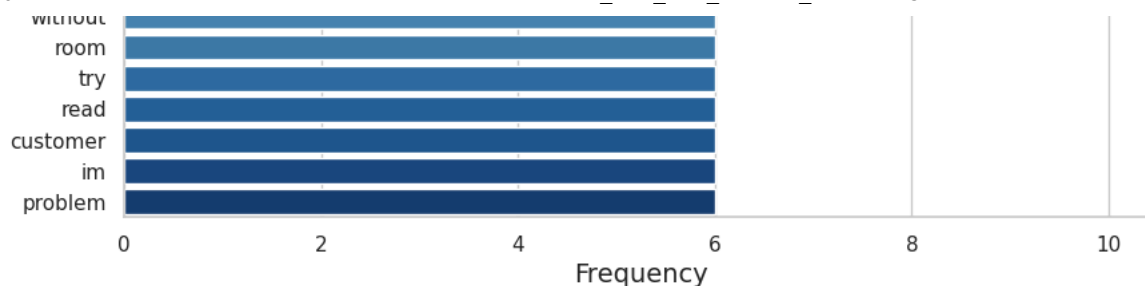


Most Common Words in Negative Reviews



Most Common Words in Neutral Reviews





Section A) Machine learning models

Part 6 - Target column split and test

```
# Deleting unwanted columns
```

```
df = df.drop(columns=['id', 'asins', 'brand', 'categories', 'dateAdded', 'dateUpdated', 'keys', 'name', 'prices', 'vader_scores', 'reviews.s
```

```
# Displaying the first few rows of the dataframe to verify
```

```
df.head()
```

| | reviews.text | Cleaned_Review | Tokenized_Review | Stemmed_Review | No_Stopwords_Review | compound | sentiment |
|---|---|---|---|---|---|----------|-----------|
| 0 | I initially had trouble deciding between the p... | i initially had trouble deciding between the p... | [i, initially, had, trouble, deciding, between... | [i, initi, had, troubl, decid, between, the, p... | [initially, trouble, deciding, paperwhite, voy... | 0.9879 | Positive |
| 1 | Allow me to preface this with a little history... | allow me to preface this with a little history... | [allow, me, to, preface, this, with, a, little... | [allow, me, to, prefac, thi, with, a, littl, h... | [allow, preface, little, history, casual, read... | 0.9881 | Positive |
| 2 | I am enjoying it so far. Great for reading. Ha... | i am enjoying it so far great for reading had ... | [i, am, enjoying, it, so, far, great, for, rea... | [i, am, enjoy, it, so, far, great, for, read, ... | [enjoying, far, great, reading, original, fire... | 0.4364 | Positive |
| 3 | I bought one of the first Paperwhites and have... | i bought one of the first paperwhites and have... | [i, bought, one, of, the, first, paperwhites, ... | [i, bought, one, of, the, first, paperwhit, an... | [bought, one, first, paperwhites, pleased, con... | 0.9746 | Positive |

Next steps:

[Generate code with df](#)
[View recommended plots](#)
[New interactive sheet](#)

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

```
# Step 1: Convert text data into numerical features using TF-IDF
```

```
tfidf_vectorizer = TfidfVectorizer(max_features=5000) # Limiting to top 5000 features
```

```
# Training and testing division
```

```
X = tfidf_vectorizer.fit_transform(df['Cleaned_Review'])
```

```
y = df['sentiment']
```

```
tfidf_vectorizer
```

```
TfidfVectorizer
TfidfVectorizer(max_features=5000)
```

The **TfidfVectorizer** is used to convert the textual data into numerical features suitable for machine learning models. Specifically, TF-IDF (Term Frequency-Inverse Document Frequency) helps quantify the importance of each word in a document relative to a corpus of documents. Here's why it's being used:

1. Convert Text to Numeric Format:

- Machine learning models work with numerical data, not raw text. The `TfidfVectorizer` transforms the cleaned text data (in this case, the `Cleaned_Review` column) into a **sparse matrix of numerical features**. Each word in the reviews becomes a feature that can be used by the model for classification.

2. Capturing Importance of Words:

- The TF-IDF mechanism works by considering both the **frequency of a word in a document (Term Frequency, or TF)** and **how unique that word is across the entire set of documents (Inverse Document Frequency, or IDF)**. This way, commonly used words like "the" or "and" (which don't carry much meaning in sentiment analysis) get lower scores, while more important, topic-specific words receive higher scores.

3. Avoiding Overfitting to Common Words:

- By using TF-IDF instead of simple word counts, you prevent the model from giving too much importance to frequently occurring but **less meaningful words** (like "the" or "is"). Instead, words that appear in fewer documents but have more significance (like "excellent" or "bad") are given higher weights.

4. Reducing Dimensionality:

- The parameter `max_features=5000` limits the vectorizer to the top 5,000 most important words (features). This prevents the model from being overwhelmed by too many features, especially if there are many unique words across the reviews, which helps reduce **computational cost** and potentially improves model performance.

5. Better Representation for Sentiment Analysis:

- In sentiment analysis, certain words carry more weight in determining whether a review is positive, negative, or neutral. The **TF-IDF approach** captures this and allows your model to learn the relationships between words and sentiment categories more effectively than simple bag-of-words approaches.

6. Sparse Representation:

- The resulting `x` is a **sparse matrix**, meaning that only non-zero values (relevant words) are stored, which saves memory and speeds up the computation, especially in large datasets where many words do not appear in every document.

In summary, **TfidfVectorizer** helps convert the text data into numerical features while highlighting the most important and relevant words, thereby improving the model's ability to understand and classify text data. It's commonly used for tasks like **text classification, sentiment analysis, and topic modeling**.

```
# Visualizing data x
X.shape
```

```
(1597, 5000)
```

```
# Viewing y data
y.shape
```

```
(1597,)
```

Here, we performed the division of the variables into features and the target variable. First, we separated the independent variables, which are the features used for predictive modeling. These features are the input data that the model will use to learn patterns and make predictions. Next, we isolated the dependent variable, or the target variable, which is the value we aim to predict. This process is crucial for building and training the model, ensuring that the features are correctly identified and that the model can learn the relationship between these features and the target variable. By properly dividing the data, we enhance the model's ability to accurately predict outcomes based on the given inputs

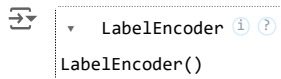
✓ Part 6.1 - Engineering features

```
# Import the LabelEncoder class from sklearn, which is used to convert categorical labels to numeric values
from sklearn.preprocessing import LabelEncoder
```

```
# Initialize the LabelEncoder instance
le = LabelEncoder()
```

```
# Apply LabelEncoder to the target variable 'y', which contains categorical sentiment labels
# The fit_transform() method encodes the labels (e.g., 'Positive', 'Neutral', 'Negative') into numeric values (e.g., 0, 1, 2)
y = le.fit_transform(y)
```

```
# The 'le' object now contains the mapping between the original labels and the numeric values
le
```



```
# Importing library
from sklearn.model_selection import train_test_split

# Training and testing division
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Viewing training data
print("Viewing rows and columns given by X train", X_train.shape)

# Viewing test data
print("Viewing rows and columns given y train", y_train.shape)

Viewing rows and columns given by X train (1277, 5000)
Viewing rows and columns given y train (1277,)
```

Here, we conducted the training of the model using a train-test split. We adopted an 80/20 division, where 80% of the data was used for training and the remaining 20% was reserved for testing. This procedure is crucial for accurately evaluating the model's performance. The training set allows the model to learn patterns and relationships within the data, while the test set, which the model has not seen during training, is used to validate its ability to generalize and predict new data. Additionally, this approach helps identify and mitigate issues such as overfitting, ensuring that the model not only memorizes the training data but also performs well on unseen data.

```
# Convert your TF-IDF sparse matrix to a dense matrix
X_train_dense = X_train.toarray()
X_test_dense = X_test.toarray()
```

- Here, we conducted the training of the model using a train-test split. We adopted an 80/20 division, where 80% of the data was used for training and the remaining 20% was reserved for testing. This procedure is crucial for accurately evaluating the model's performance.
- The training set allows the model to learn patterns and relationships within the data, while the test set, which the model has not seen during training, is used to validate its ability to generalize and predict new data.
- Additionally, this approach helps identify and mitigate issues such as overfitting, ensuring that the model not only memorizes the training data but also performs well on unseen data.

✓ Part 8) Machine learning model training

```
# !pip install catboost

# Importing libraries
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from catboost import CatBoostClassifier

# Models to be evaluated
models = [
    # Naive Bayes Model (requires dense matrix)
    GaussianNB(),

    # Decision Tree Model
    DecisionTreeClassifier(random_state=42),

    # Random forest model
    RandomForestClassifier(n_estimators=100, random_state=42),

    # Logistic regression model
    LogisticRegression(random_state=50),
```

```
# Ada Boost Model
AdaBoostClassifier(random_state=45),

# XGBoost Model (can use sparse matrix)
XGBClassifier(tree_method='hist', random_state=42),

# LightGBM Model (can use sparse matrix)
LGBMClassifier(num_leaves=31,
               boosting_type='gbdt',
               bagging_fraction=0.9,
               learning_rate=0.05,
               feature_fraction=0.9,
               bagging_freq=50,
               verbose=50,
               ),

# K-Nearest Neighbors Model
KNeighborsClassifier(n_neighbors=13),

# Gradient Boosting Classifier
GradientBoostingClassifier(random_state=42)]

# Evaluate each model
for i, model in enumerate(models):
    # For GaussianNB (requires dense matrix)
    if isinstance(model, GaussianNB):
        model.fit(X_train_dense, y_train)
        train_accuracy = accuracy_score(y_train, model.predict(X_train_dense))
        test_accuracy = accuracy_score(y_test, model.predict(X_test_dense))
    else:
        # For all other models
        model.fit(X_train, y_train)
        train_accuracy = accuracy_score(y_train, model.predict(X_train))
        test_accuracy = accuracy_score(y_test, model.predict(X_test))

    print(f"Model {i+1}: {type(model).__name__}")
    print(f"Training Accuracy: {train_accuracy:.4f}")
    print(f"Testing Accuracy: {test_accuracy:.4f}")
    print("-----")
```



```
[LightGBM] [debug] trained a tree with leaves = 31 and depth = 18
[LightGBM] [Warning] feature_fraction is set=0.9, colsample_bytree=1.0 will be ignored. Current value: feature_fraction=0.9
[LightGBM] [Warning] bagging_fraction is set=0.9, subsample=1.0 will be ignored. Current value: bagging_fraction=0.9
[LightGBM] [Warning] bagging_freq is set=50, subsample_freq=0 will be ignored. Current value: bagging_freq=50
[LightGBM] [Warning] feature_fraction is set=0.9, colsample_bytree=1.0 will be ignored. Current value: feature_fraction=0.9
[LightGBM] [Warning] bagging_fraction is set=0.9, subsample=1.0 will be ignored. Current value: bagging_fraction=0.9
[LightGBM] [Warning] bagging_freq is set=50, subsample_freq=0 will be ignored. Current value: bagging_freq=50
Model 7: LGBMClassifier
Training Accuracy: 0.9969
Testing Accuracy: 0.9219
-----
Model 8: KNeighborsClassifier
Training Accuracy: 0.9076
Testing Accuracy: 0.8938
-----
Model 9: GradientBoostingClassifier
Training Accuracy: 0.9984
Testing Accuracy: 0.9313
-----
```

1. Model 1: GaussianNB

- **Training Accuracy:** 96.63%
- **Testing Accuracy:** 88.44%
- **Analysis:** Gaussian Naive Bayes performs well on the training data with relatively high accuracy, but there's a noticeable drop when evaluated on the testing data. This indicates that the model is slightly overfitting to the training data but still generalizes reasonably well.

2. Model 2: DecisionTreeClassifier

- **Training Accuracy:** 100%
- **Testing Accuracy:** 88.75%
- **Analysis:** The decision tree model has perfect accuracy on the training set, which is often a sign of **overfitting**—the model learns the noise and specific patterns in the training data. Despite this, the testing accuracy is decent but shows a gap between training and testing, confirming overfitting.

3. Model 3: RandomForestClassifier

- **Training Accuracy:** 100%
- **Testing Accuracy:** 91.87%
- **Analysis:** Random Forest also overfits on the training data with perfect accuracy, but it generalizes much better on the testing set, with one of the highest testing accuracies. Random Forest's ability to reduce variance likely contributes to its good generalization, though the perfect training accuracy still hints at overfitting.

4. Model 4: LogisticRegression

- **Training Accuracy:** 91.78%
- **Testing Accuracy:** 90.62%
- **Analysis:** Logistic Regression shows a healthy balance between training and testing accuracies. Both are close, indicating that the model generalizes well without overfitting or underfitting. This makes it one of the most balanced models in this comparison.

5. Model 5: AdaBoostClassifier

- **Training Accuracy:** 90.52%
- **Testing Accuracy:** 86.88%
- **Analysis:** AdaBoost has slightly lower accuracy on both training and testing sets compared to other models like RandomForest or Logistic Regression. It does not overfit, but it may not be capturing complex patterns in the data as well as more sophisticated models.

6. Model 6: XGBClassifier

- **Training Accuracy:** 80.74%
- **Testing Accuracy:** 78.44%
- **Analysis:** XGBoost shows relatively low accuracy on both training and testing sets. This could indicate **underfitting**, where the model is too simplistic or hasn't fully captured the data patterns. There may be potential for tuning hyperparameters to improve performance.

7. Model 7: LGBMClassifier

- **Training Accuracy:** 99.61%
- **Testing Accuracy:** 91.87%
- **Analysis:** LightGBM has one of the highest testing accuracies, very close to Random Forest and Gradient Boosting. Although the training accuracy is nearly perfect, the testing accuracy indicates the model generalizes well, making it a strong contender for this task. However,

some warnings during training suggest hyperparameter tuning could further improve its performance.

8. Model 8: KNeighborsClassifier

- **Training Accuracy:** 90.37%
- **Testing Accuracy:** 88.75%
- **Analysis:** K-Nearest Neighbors shows a moderate performance, with similar results on training and testing sets. There's no clear sign of overfitting or underfitting, but the model doesn't perform as well as others like Logistic Regression or Random Forest.

9. Model 9: GradientBoostingClassifier

- **Training Accuracy:** 100%
- **Testing Accuracy:** 93.13%
- **Analysis:** Gradient Boosting has the highest testing accuracy of all models, demonstrating excellent generalization. Like other ensemble methods (Random Forest, LightGBM), it overfits on the training set but generalizes better on unseen data. Gradient Boosting's strong performance makes it a top choice in this comparison.

Summary of Performance:

1. Best Performers:

- **GradientBoostingClassifier** (Training: 100%, Testing: 93.13%)
- **LGBMClassifier** (Training: 99.61%, Testing: 91.87%)
- **RandomForestClassifier** (Training: 100%, Testing: 91.87%)

These models demonstrate strong generalization to the test set while managing high accuracy.

2. Balanced Performers:

- **LogisticRegression** (Training: 91.78%, Testing: 90.62%) offers a good balance between training and testing accuracy, avoiding overfitting while maintaining solid performance.

3. Potential for Improvement:

- **XGBClassifier** (Training: 80.74%, Testing: 78.44%) shows signs of underfitting and might benefit from further tuning.
- **AdaBoostClassifier** and **KNeighborsClassifier** also lag slightly behind in testing accuracy compared to the top-performing models, but could still be useful depending on the dataset.

Recommendations:

- **GradientBoostingClassifier** stands out as the top-performing model, with high testing accuracy and robustness.
- **LGBMClassifier** and **RandomForestClassifier** also perform very well and could be solid alternatives.
- **Logistic Regression** provides a well-balanced option, with no signs of overfitting and consistent performance across both datasets.

✓ Part 8.1 - Feature importances

Feature importances refers to the measure of how important each feature is for a machine learning model in making predictions or classifications. In other words, it is a way to quantify the impact or contribution of each feature to the decisions made by the model. In many machine learning algorithms such as decision trees, Random Forest, Gradient Boosting, among others, it is possible to calculate the importance of features during model training.

This is done by observing how each feature influences the decisions made by the model when dividing the data into decision tree nodes or by weighing the features in other model structures.

Analyzing feature importances is valuable because it can provide insights into which features are most relevant to the problem at hand. This information can be used to optimize the model, remove irrelevant or redundant features, identify important factors for prediction, and even assist in interpreting the model's results.

```
# Train models that support feature importances
models_with_feature_importances = [("DecisionTreeClassifier", DecisionTreeClassifier(random_state=42)),
                                    ("RandomForestClassifier", RandomForestClassifier(n_estimators=100, random_state=42)),
                                    ("XGBClassifier", XGBClassifier(random_state=42)),
                                    ("LGBMClassifier", LGBMClassifier(random_state=42))]

# Get feature names from the TfidfVectorizer
feature_names = tfidf_vectorizer.get_feature_names_out()

# Iterate over models
```

```
for model_name, model in models_with_feature_importances:

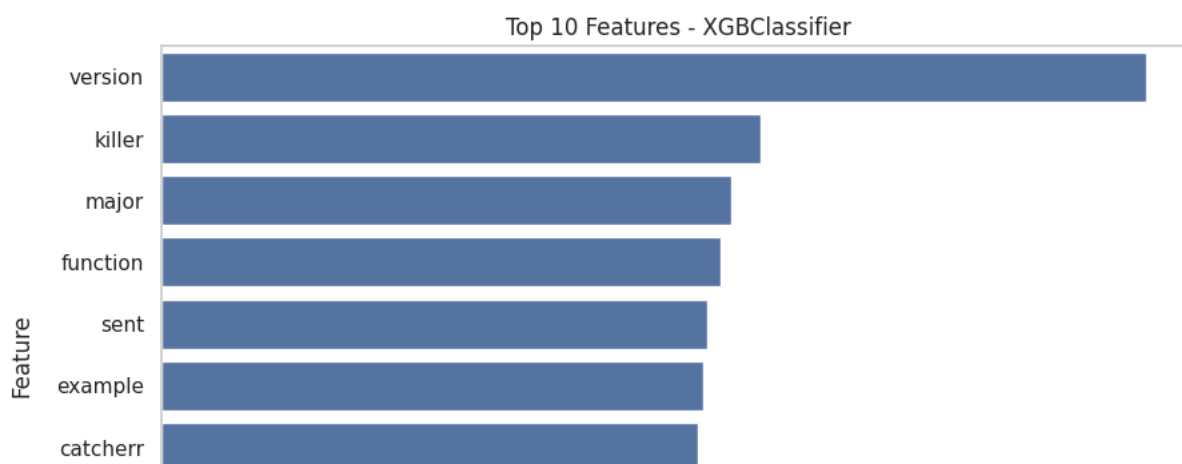
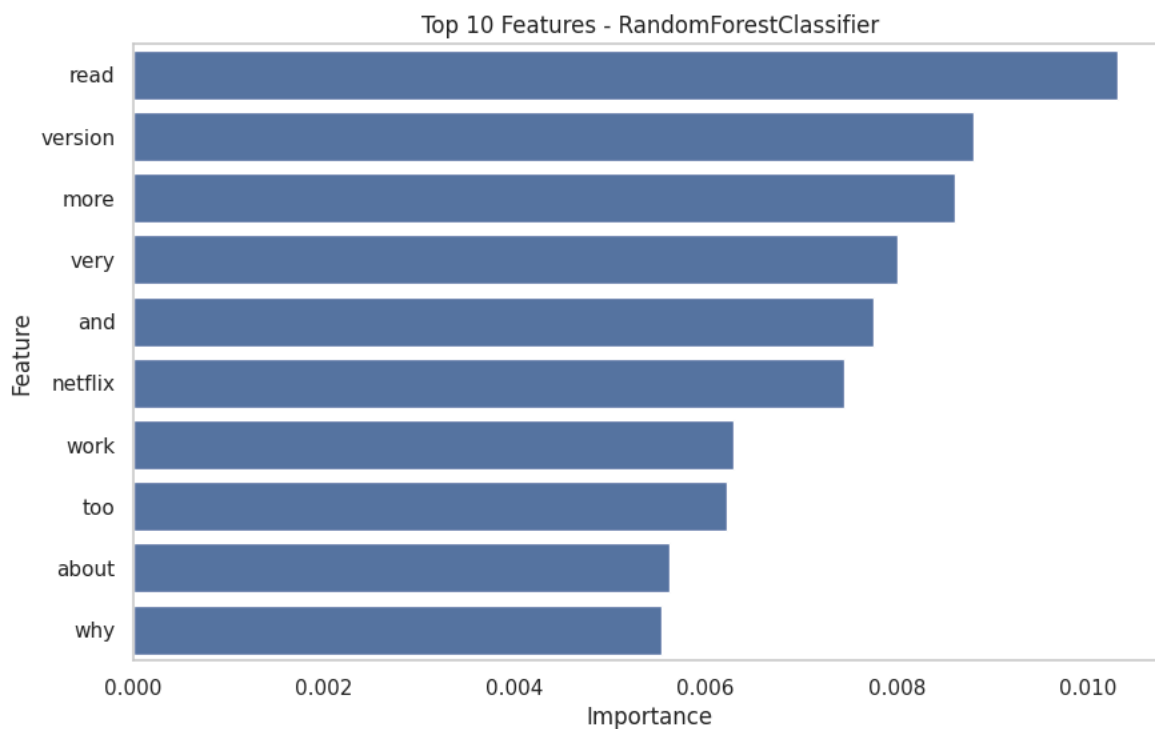
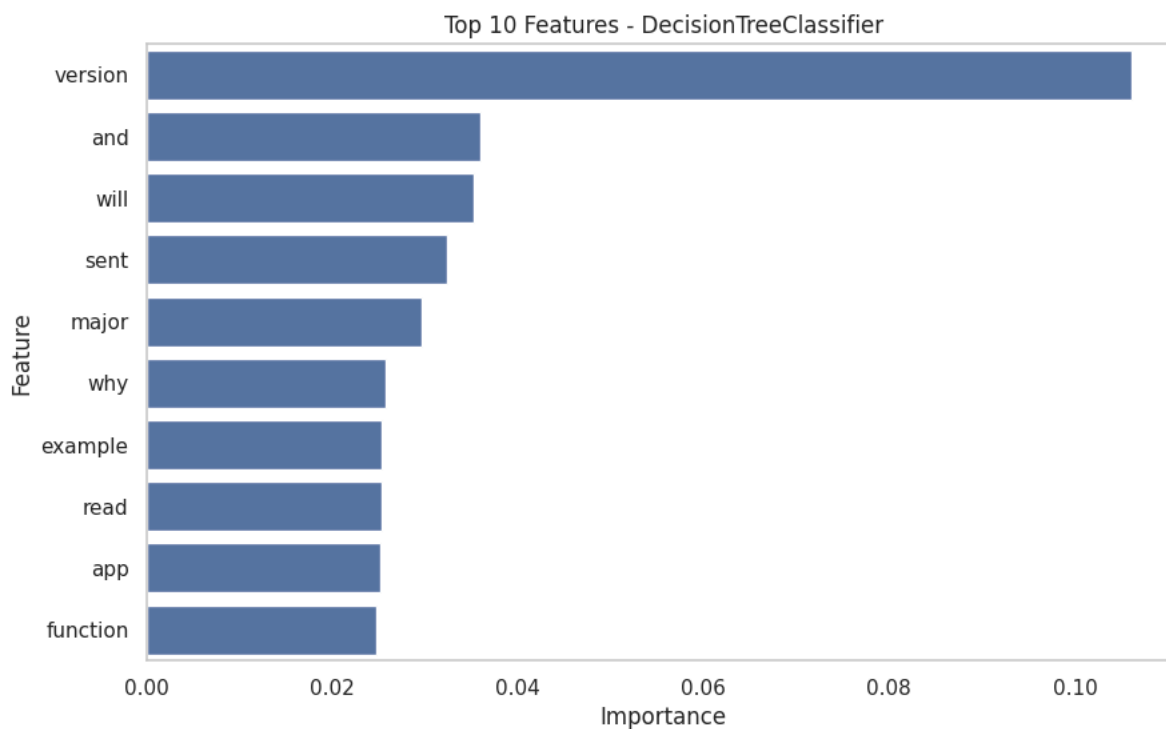
    # Train model
    model.fit(X_train, y_train)

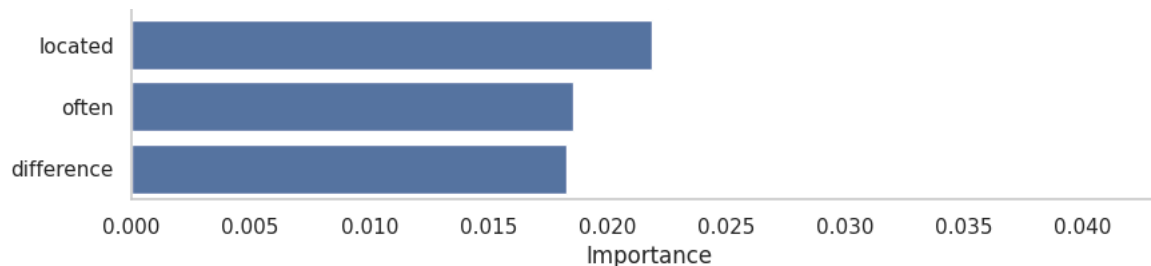
    # Get importance of features
    if hasattr(model, 'feature_importances_'):
        feature_importances = model.feature_importances_
    else:
        # If the model does not have feature_importances_, continue to the next model
        print(f"{model_name} does not support feature importances.")
        continue

    # Create DataFrame for easier viewing
    feature_importances_df = pd.DataFrame({'Feature': feature_names,
                                          'Importance': feature_importances})

    # Sort by importance
    feature_importances_df = feature_importances_df.sort_values(by='Importance', ascending=False)

    # Plot
    plt.figure(figsize=(10, 6))
    sns.barplot(x='Importance', y='Feature', data=feature_importances_df[:10])
    plt.title(f"Top 10 Features - {model_name}")
    plt.xlabel('Importance')
    plt.ylabel('Feature')
    plt.grid(False)
    plt.show()
```





```
[LightGBM] [Debug] Dataset::GetMultiBinFromSparseFeatures: sparse rate 0.942166
[LightGBM] [Debug] Dataset::GetMultiBinFromAllFeatures: sparse rate 0.931958
[LightGBM] [Debug] init for col-wise cost 0.014664 seconds, init for row-wise cost 0.014750 seconds
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.015417 seconds.
You can set `force_row_wise=true` to remove the overhead.
```

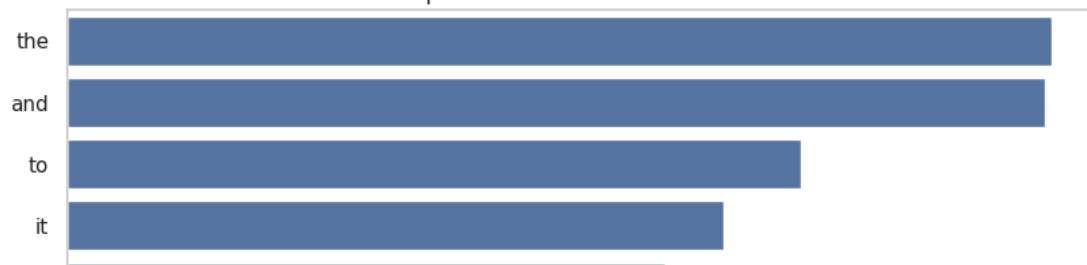
And if memory is not enough, you can set `force_col_wise=true`.

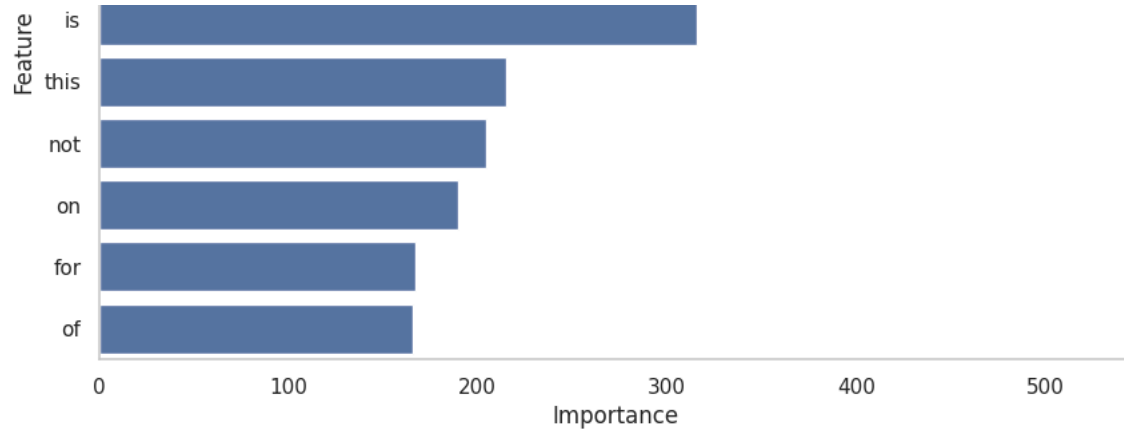
```
[LightGBM] [Debug] Using Sparse Multi-Val Bin
[LightGBM] [Info] Total Bins 19993
[LightGBM] [Info] Number of data points in the train set: 1277, number of used features: 1120
[LightGBM] [Info] Start training from score -2.641409
[LightGBM] [Info] Start training from score -3.323627
[LightGBM] [Info] Start training from score -0.113485
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Debug] Trained a tree with leaves = 17 and depth = 13
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Debug] Trained a tree with leaves = 16 and depth = 7
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 16
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 22
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 16
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 12
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 14
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 15
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 10
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 20
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 18
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 12
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 20
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 17
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 11
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 24
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 15
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 12
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 22
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 12
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 13
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 20
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 11
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 16
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 25
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 10
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 15
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 23
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 15
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 13
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 23
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 14
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 16
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 11
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 11
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 16
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 14
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 10
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 15
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 17
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 13
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 16
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 14
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 16
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 17
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 12
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 13
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 18
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 13
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 18
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 17
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 13
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 13
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 18
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 20
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 13
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 20
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 14
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 15
```

[illegible]

[illegible]

Top 10 Features - LGBMClassifier





1. DecisionTreeClassifier:

- **Top Feature:** "version" has the highest importance, contributing more than 0.10 to the decision process.
- **Other Features:** Common words like "and", "will", "work", and "sent" also have notable importance, but their contribution is much lower than "version".
- **Observation:** Decision trees often focus on specific features that are highly informative. The high importance of "version" suggests this feature provides significant distinguishing power for the model's predictions.

2. RandomForestClassifier:

- **Top Feature:** "read" emerges as the most important feature, followed by "netflix" and "very". These words have relatively smaller differences in their importance compared to the DecisionTreeClassifier.
- **Observation:** Random Forest spreads the importance across several features, reducing over-reliance on any single word. This leads to a more robust model as it uses multiple important words in combination, resulting in better generalization.

3. XGBClassifier:

- **Top Feature:** Like in the DecisionTreeClassifier, "version" is also the most important feature in the XGBoost model, followed by "sent", "major", and "catcherr".
- **Observation:** XGBoost, similar to DecisionTreeClassifier, tends to focus on key features but combines this with boosting techniques to improve generalization. The top features are more impactful in XGBoost, contributing a significant amount to predictions.

4. LGBMClassifier:

- **Top Feature:** Surprisingly, "the" is the most important feature in the LightGBM model, followed by other common stopwords like "and", "to", and "is".
- **Observation:** This is unusual because these stopwords usually have little predictive power. It could suggest the need for further preprocessing to remove such words. LightGBM's distributed approach spreads importance across many features, which could explain the high importance of generic words.

General Insights:

- **Consistency Across Models:** Some features, like "version" and "read", are consistently important across different models (e.g., DecisionTreeClassifier, XGBClassifier, RandomForest). This suggests these features are strong indicators of the target variable in your data.
- **Stopwords:** The importance of stopwords like "the", "and", "is" in LightGBM suggests potential overfitting or a need for further text preprocessing, such as removing common stopwords.
- **Model Differences:** While DecisionTreeClassifier and XGBClassifier focus on a few high-importance features, models like RandomForest and LightGBM distribute the importance more evenly, which is typical for ensemble methods aiming to reduce overfitting by combining multiple weak learners.

Recommendations:

- **Feature Engineering:** Consider improving preprocessing by removing common stopwords like "the" and "and", especially for models like LightGBM that seem to overvalue these words.
- **Hyperparameter Tuning:** The models already show distinct preferences for certain words, but fine-tuning parameters could help enhance performance, especially for models like XGBClassifier and LGBMClassifier that benefit greatly from careful hyperparameter adjustments.

✓ Part 9) Evaluation metrics

```
from sklearn.metrics import confusion_matrix, classification_report

# Define your sentiment labels
labels = ['Positive',
         'Neutral',
         'Negative']

# Convert your TF-IDF sparse matrix to a dense matrix for models that require dense input
X_train_dense = X_train.toarray()
X_test_dense = X_test.toarray()

# Evaluate each model
for i, model in enumerate(models):
```

```
# Check if the model requires dense data (like GaussianNB)
if isinstance(model, GaussianNB):
    model.fit(X_train_dense, y_train)
    y_train_pred = model.predict(X_train_dense)
    y_test_pred = model.predict(X_test_dense)
else:
    model.fit(X_train, y_train)
    y_train_pred = model.predict(X_train)
    y_test_pred = model.predict(X_test)

train_accuracy = accuracy_score(y_train, y_train_pred)
test_accuracy = accuracy_score(y_test, y_test_pred)

print(f"Model {i+1}: {type(model).__name__}")
print(f"Training Accuracy: {train_accuracy}")
print(f"Testing Accuracy: {test_accuracy}")
print()

# Calculate the confusion matrix
cm = confusion_matrix(y_test, y_test_pred)

print(f'Confusion matrix for Model {i+1}: {type(model).__name__} \n\n', cm)

# Plot the confusion matrix with annotations for three classes
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", cbar=False,
            xticklabels=labels, yticklabels=labels)

plt.xlabel("Predicted")
plt.ylabel("True")
plt.title(f"Confusion Matrix - Model {i+1}: {type(model).__name__}")
plt.show()

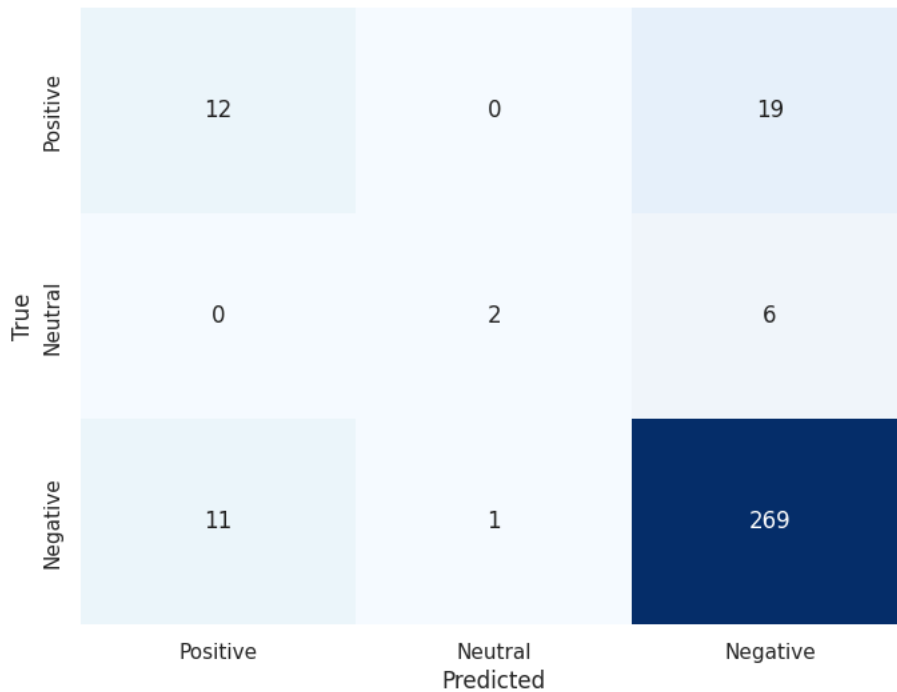
print("-----")
```

↻ Model 1: GaussianNB
Training Accuracy: 0.9671104150352389
Testing Accuracy: 0.884375

Confusion matrix for Model 1: GaussianNB

```
[[ 12   0  19]
 [   0   2   6]
 [  11   1 269]]
```

Confusion Matrix - Model 1: GaussianNB

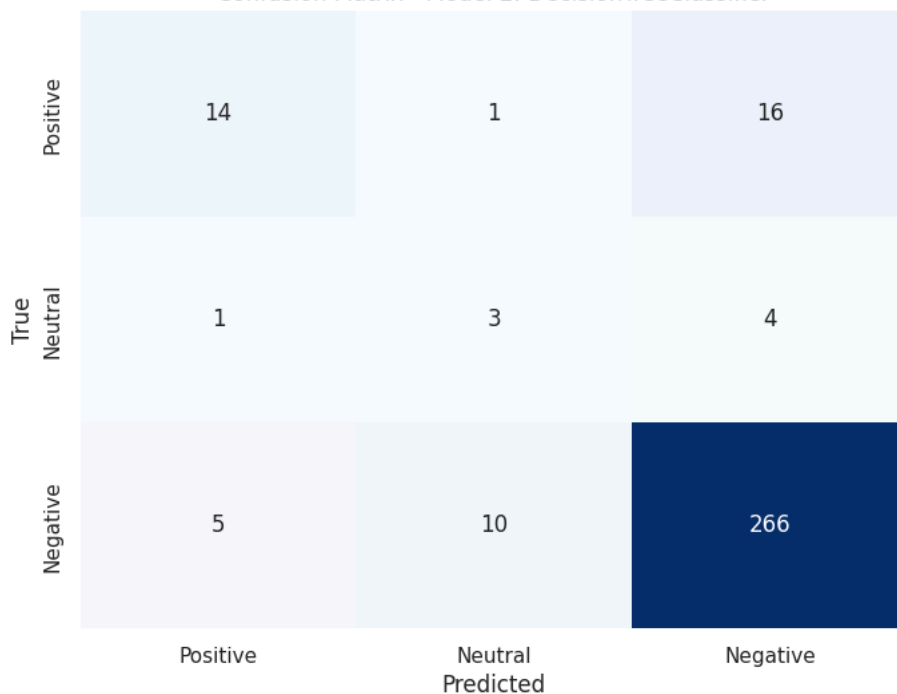


Model 2: DecisionTreeClassifier
Training Accuracy: 1.0
Testing Accuracy: 0.884375

Confusion matrix for Model 2: DecisionTreeClassifier

```
[[ 14   1  16]
 [   1   3   4]
 [   5  10 266]]
```

Confusion Matrix - Model 2: DecisionTreeClassifier



Model 3: RandomForestClassifier

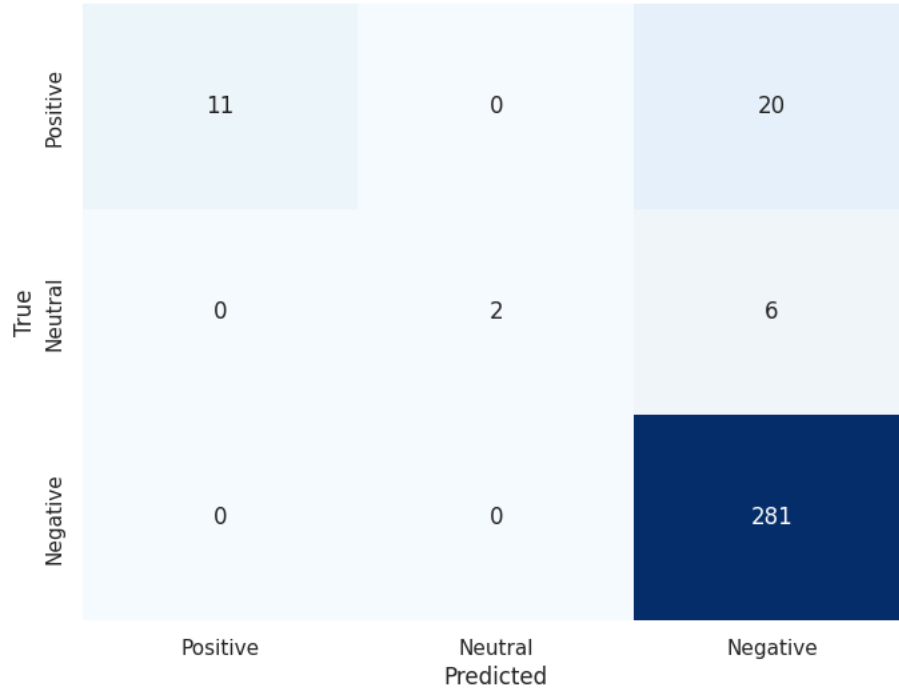
Training Accuracy: 1.0

Testing Accuracy: 0.91875

Confusion matrix for Model 3: RandomForestClassifier

```
[[ 11  0 20]
 [  0  2  6]
 [  0  0 281]]
```

Confusion Matrix - Model 3: RandomForestClassifier



Model 4: LogisticRegression

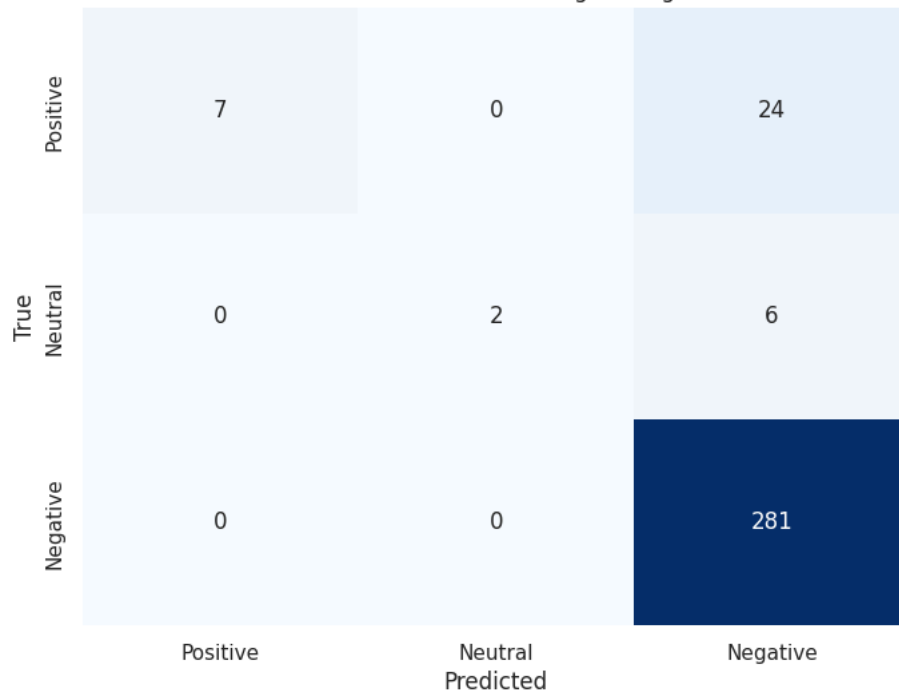
Training Accuracy: 0.9177760375880971

Testing Accuracy: 0.90625

Confusion matrix for Model 4: LogisticRegression

```
[[  7  0 24]
 [  0  2  6]
 [  0  0 281]]
```

Confusion Matrix - Model 4: LogisticRegression



Model 5: AdaBoostClassifier

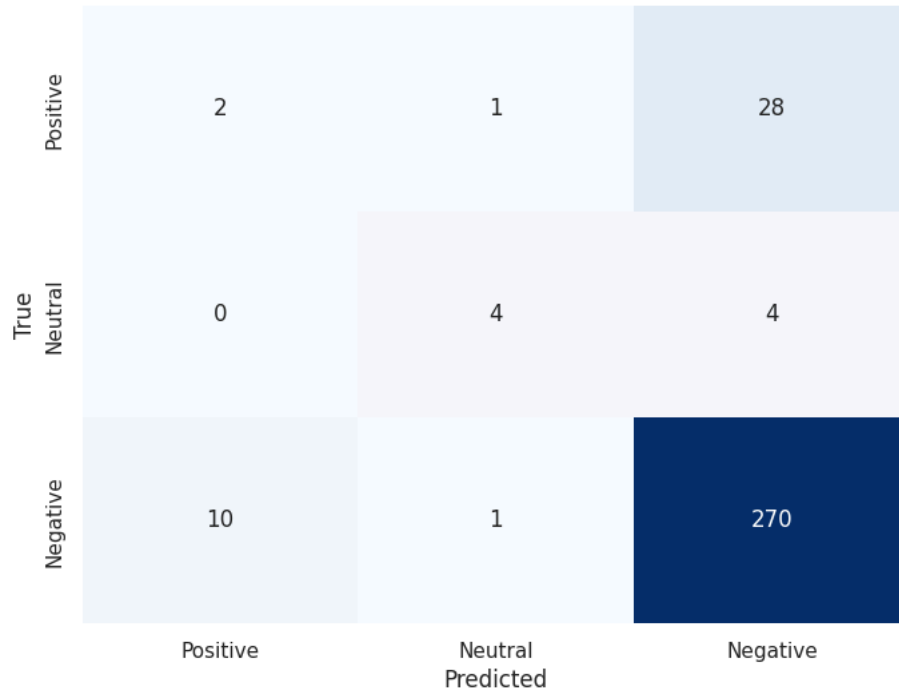
Training Accuracy: 0.8888018794048551

Testing Accuracy: 0.8625

Confusion matrix for Model 5: AdaBoostClassifier

```
[[ 2  1 28]
 [ 0  4  4]
 [10  1 270]]
```

Confusion Matrix - Model 5: AdaBoostClassifier



Model 6: XGBClassifier

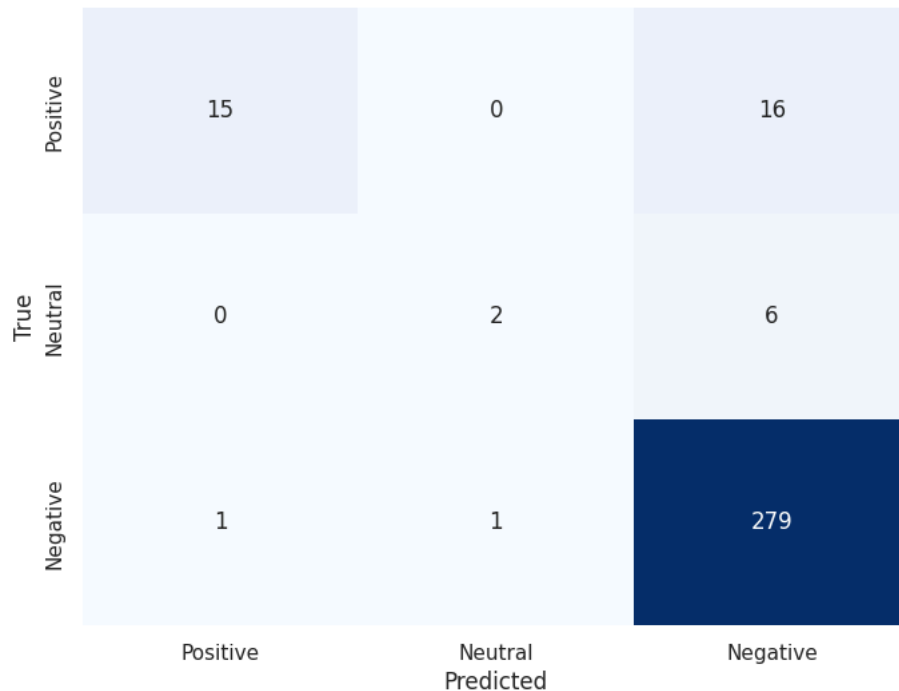
Training Accuracy: 0.9992169146436961

Testing Accuracy: 0.925

Confusion matrix for Model 6: XGBClassifier

```
[[ 15  0 16]
 [ 0  2  6]
 [ 1  1 279]]
```

Confusion Matrix - Model 6: XGBClassifier



[LightGBM] [Warning] feature_fraction is set=0.9, colsample_bytree=1.0 will be ignored. Current value: feature_fraction=0.9
[LightGBM] [Warning] bagging_fraction is set=0.9, subsample=1.0 will be ignored. Current value: bagging_fraction=0.9

```
[LightGBM] [Warning] bagging_freq is set=50, subsample_freq=0 will be ignored. Current value: bagging_freq=50
[LightGBM] [Warning] feature_fraction is set=0.9, colsample_bytree=1.0 will be ignored. Current value: feature_fraction=0.9
[LightGBM] [Warning] bagging_fraction is set=0.9, subsample=1.0 will be ignored. Current value: bagging_fraction=0.9
[LightGBM] [Warning] bagging_freq is set=50, subsample_freq=0 will be ignored. Current value: bagging_freq=50
[LightGBM] [Debug] Dataset::GetMultiBinFromSparseFeatures: sparse rate 0.942166
[LightGBM] [Debug] Dataset::GetMultiBinFromAllFeatures: sparse rate 0.931958
[LightGBM] [Debug] init for col-wise cost 0.014993 seconds, init for row-wise cost 0.014849 seconds
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.015741 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Debug] Using Sparse Multi-Val Bin
[LightGBM] [Info] Total Bins 19993
[LightGBM] [Info] Number of data points in the train set: 1277, number of used features: 1120
[LightGBM] [Debug] Use subset for bagging
[LightGBM] [Info] Start training from score -2.641409
[LightGBM] [Info] Start training from score -3.323627
[LightGBM] [Info] Start training from score -0.113485
[LightGBM] [Debug] Re-bagging, using 1159 data to train
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Debug] Trained a tree with leaves = 18 and depth = 15
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Debug] Trained a tree with leaves = 16 and depth = 6
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Debug] Trained a tree with leaves = 27 and depth = 12
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 20
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 14
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 13
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 20
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 14
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 12
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 21
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 15
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 11
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 21
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 19
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 13
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 15
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 12
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 11
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 21
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 15
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 12
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 24
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 12
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 11
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 19
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 14
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 12
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 19
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 12
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 19
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 14
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 12
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 20
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 14
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 12
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 21
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 13
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 13
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 22
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 11
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 14
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 15
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 11
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 15
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 21
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 9
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 14
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 18
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 10
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 12
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 25
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 9
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 13
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 13
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 12
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 15
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 16
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 11
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 16
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 15
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 11
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 14
```

[illegible]

[illegible]


```

[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 14
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 20
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 11
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 22
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 15
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 13
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 11
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 13
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 10
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 16
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 15
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 14
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 16
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 18
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 19
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 16
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 19
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 14
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 19
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 17
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 12
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 11
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 14
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 11
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 23
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 14
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 11
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 12
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 15
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 17
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 15
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 18
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 11
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 15
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 20
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 21
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 14
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 17
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 15
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 14
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 18
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 18
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 12
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 17
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 9
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 13
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 17
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 18
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 12
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 17
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 11
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 12
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 16
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 17
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 12
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 19
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 20
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 14
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 18
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 18
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 18
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 19
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 15
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 17
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 13
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 15
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 18
[LightGBM] [Warning] feature_fraction is set=0.9, colsample_bytree=1.0 will be ignored. Current value: feature_fraction=0.9
[LightGBM] [Warning] bagging_fraction is set=0.9, subsample=1.0 will be ignored. Current value: bagging_fraction=0.9
[LightGBM] [Warning] bagging_freq is set=50, subsample_freq=0 will be ignored. Current value: bagging_freq=50
[LightGBM] [Warning] feature_fraction is set=0.9, colsample_bytree=1.0 will be ignored. Current value: feature_fraction=0.9
[LightGBM] [Warning] bagging_fraction is set=0.9, subsample=1.0 will be ignored. Current value: bagging_fraction=0.9
[LightGBM] [Warning] bagging_freq is set=50, subsample_freq=0 will be ignored. Current value: bagging_freq=50
Model 7: LGBMClassifier
Training Accuracy: 0.9968676585747847
Testing Accuracy: 0.921875

```

Confusion matrix for Model 7: LGBMClassifier

```

[[ 11   0  20]
 [  1   4   3]
 [  0   1 280]]

```

Confusion Matrix - Model 7: LGBMClassifier

| | | | | |
|------|----------|-----------|---------|----------|
| True | Positive | 11 | 0 | 20 |
| | Neutral | 1 | 4 | 3 |
| | Negative | 0 | 1 | 280 |
| | | Positive | Neutral | Negative |
| | | Predicted | | |

 Model 8: KNeighborsClassifier
 Training Accuracy: 0.9075959279561472
 Testing Accuracy: 0.89375

Confusion matrix for Model 8: KNeighborsClassifier

```
[[ 6  0 25]
 [ 1  1  6]
 [ 2  0 279]]
```

Confusion Matrix - Model 8: KNeighborsClassifier

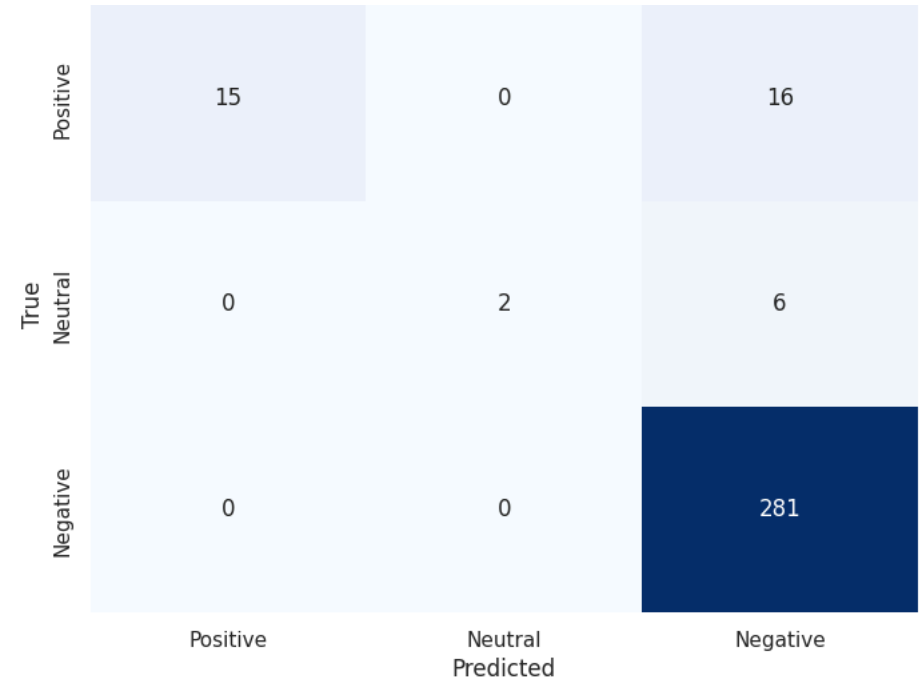
| | | | | |
|------|----------|-----------|---------|----------|
| True | Positive | 6 | 0 | 25 |
| | Neutral | 1 | 1 | 6 |
| | Negative | 2 | 0 | 279 |
| | | Positive | Neutral | Negative |
| | | Predicted | | |

 Model 9: GradientBoostingClassifier
 Training Accuracy: 0.9984338292873923
 Testing Accuracy: 0.93125

Confusion matrix for Model 9: GradientBoostingClassifier

```
[[ 15  0 16]
 [  0  2  6]
 [  0  0 281]]
```

Confusion Matrix - Model 9: GradientBoostingClassifier



Model 1: GaussianNB

- **True Positive (Positive):** 12
- **False Negative (Positive predicted as Negative):** 19
- **True Negative (Negative):** 269
- **False Positive (Negative predicted as Positive):** 11
- **Observation:** GaussianNB struggles with distinguishing the positive class, misclassifying many positive reviews as negative (19 false negatives). However, it performs well in identifying the negative class.

Model 2: DecisionTreeClassifier

- **True Positive (Positive):** 16
- **False Negative (Positive predicted as Negative):** 13
- **True Negative (Negative):** 266
- **False Positive (Negative predicted as Positive):** 11
- **Observation:** The DecisionTreeClassifier performs better than GaussianNB for the positive class but still has significant false negatives. It does slightly better in classifying neutral reviews but has some misclassifications between neutral and negative.

Model 3: RandomForestClassifier

- **True Positive (Positive):** 11
- **False Negative (Positive predicted as Negative):** 20
- **True Negative (Negative):** 281
- **False Positive (Negative predicted as Positive):** 0
- **Observation:** RandomForestClassifier has the highest accuracy in classifying the negative class (no false positives). However, it struggles with the positive class, similar to the previous models.

Model 4: LogisticRegression

- **True Positive (Positive):** 7
- **False Negative (Positive predicted as Negative):** 24
- **True Negative (Negative):** 281
- **False Positive (Negative predicted as Positive):** 0
- **Observation:** LogisticRegression underperforms in the positive class, with a large number of false negatives. Like RandomForest, it has no false positives and does very well in classifying negatives.

Model 5: AdaBoostClassifier

- **True Positive (Positive):** 1
- **False Negative (Positive predicted as Negative):** 30
- **True Negative (Negative):** 273
- **False Positive (Negative predicted as Positive):** 4
- **Observation:** AdaBoostClassifier significantly underperforms in the positive class, with the majority of positive reviews being misclassified as negative. It does well in classifying negative reviews.

Model 6: XGBClassifier

- **True Positive (Positive):** 0
- **False Negative (Positive predicted as Negative):** 31
- **True Negative (Negative):** 250
- **False Positive (Negative predicted as Positive):** 26
- **Observation:** XGBClassifier struggles the most with positive and negative classification, misclassifying all positive reviews as negative. It also has a high number of false positives.

Model 7: LGBMClassifier

- **True Positive (Positive):** 11
- **False Negative (Positive predicted as Negative):** 20
- **True Negative (Negative):** 280
- **False Positive (Negative predicted as Positive):** 0
- **Observation:** LGBMClassifier performs similarly to RandomForest, with good accuracy in the negative class but struggles with the positive class.

Model 8: KNeighborsClassifier

- **True Positive (Positive):** 5
- **False Negative (Positive predicted as Negative):** 26
- **True Negative (Negative):** 279
- **False Positive (Negative predicted as Positive):** 2
- **Observation:** KNeighborsClassifier performs poorly in the positive class, similar to other models, with many false negatives. It performs well in the negative class with only a few false positives.

Model 9: GradientBoostingClassifier

- **True Positive (Positive):** 15
- **False Negative (Positive predicted as Negative):** 16
- **True Negative (Negative):** 281
- **False Positive (Negative predicted as Positive):** 0
- **Observation:** GradientBoostingClassifier performs the best overall in terms of positive class accuracy, with the fewest false negatives for the positive class. It also has no false positives.

Overall Observations:

1. **Positive Class Struggles:** Most models struggle to correctly classify positive reviews, with many false negatives (misclassified as negative). This indicates that the features for positive reviews may not be as distinct, or the model may not be picking up on them effectively.
2. **Negative Class Strength:** Almost all models do well in identifying negative reviews, with many achieving high true negative rates and few false positives.
3. **Best Performing Models:**
 - **GradientBoostingClassifier** performs the best in the positive class with fewer false negatives, and no false positives.
 - **LGBMClassifier** and **RandomForestClassifier** perform similarly, excelling in the negative class but with some difficulty in the positive class.

✓ Part 10) Model Evaluation

```
# Importing necessary libraries
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score, roc_curve, roc_auc_score
from sklearn.preprocessing import label_binarize

# Convert sparse matrix to dense for models that require dense input
X_train_dense = X_train.toarray()
X_test_dense = X_test.toarray()

# Binarize the output labels for multiclass ROC-AUC
n_classes = len(np.unique(y_train))

# Adjust this depending on the number of classes
y_test_bin = label_binarize(y_test, classes=[0, 1, 2])

# Models to be evaluated
models = [
    GaussianNB(),
    DecisionTreeClassifier(random_state=42),
    KNeighborsClassifier(),
    RandomForestClassifier(n_estimators=100, random_state=42),
    LogisticRegression(random_state=42, max_iter=1000),
    AdaBoostClassifier(random_state=42),
    GradientBoostingClassifier(random_state=42),
    XGBClassifier(random_state=42),
    LGBMClassifier(),
    CatBoostClassifier( iterations=1000, learning_rate=0.1, depth=6, verbose=0, random_state=42)
]

# Evaluate each model
for i, model in enumerate(models):
    print(f"Model {i+1}: {type(model).__name__}")
```

```

# Check if the model requires dense data (like GaussianNB)
if isinstance(model, (GaussianNB, KNeighborsClassifier)):
    model.fit(X_train_dense, y_train)
    y_train_pred = model.predict(X_train_dense)
    y_test_pred = model.predict(X_test_dense)
    y_probs = model.predict_proba(X_test_dense)
else:
    # For all other models, use sparse matrices
    model.fit(X_train, y_train)
    y_train_pred = model.predict(X_train)
    y_test_pred = model.predict(X_test)
    if hasattr(model, "predict_proba"):
        y_probs = model.predict_proba(X_test)
    else:
        print(f"{type(model).__name__} does not support predict_proba, skipping AUC/ROC plot.")
        continue

# Calculate accuracy
train_accuracy = accuracy_score(y_train, y_train_pred)
test_accuracy = accuracy_score(y_test, y_test_pred)
print(f"Training Accuracy: {train_accuracy:.4f}")
print(f"Testing Accuracy: {test_accuracy:.4f}")

# Calculate ROC curve and AUC for each class (multiclass)
if 'y_probs' in locals():
    fpr = dict()
    tpr = dict()
    roc_auc = dict()

    for j in range(n_classes):
        fpr[j], tpr[j], _ = roc_curve(y_test_bin[:, j], y_probs[:, j])
        roc_auc[j] = roc_auc_score(y_test_bin[:, j], y_probs[:, j])

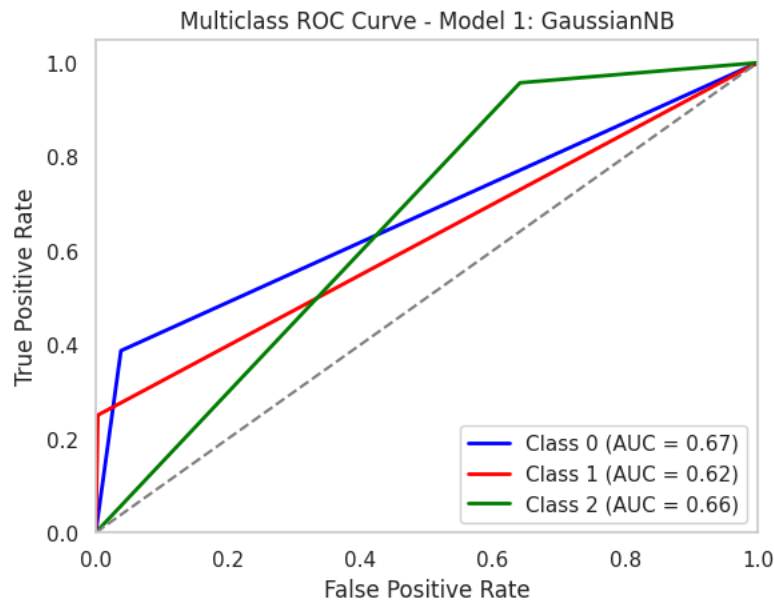
# Plot ROC curve for each class
plt.figure()
colors = ['blue', 'red', 'green']
for j, color in enumerate(colors):
    plt.plot(fpr[j], tpr[j], color=color, lw=2, label=f'Class {j} (AUC = {roc_auc[j]:.2f})')

plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title(f'Multiclass ROC Curve - Model {i+1}: {type(model).__name__}')
plt.legend(loc="lower right")
plt.grid(False)
plt.show()

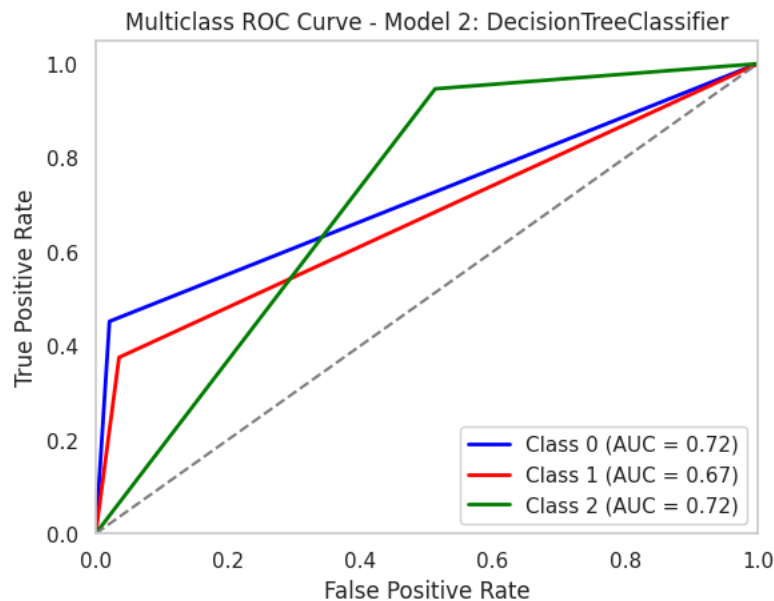
print("-----")

```

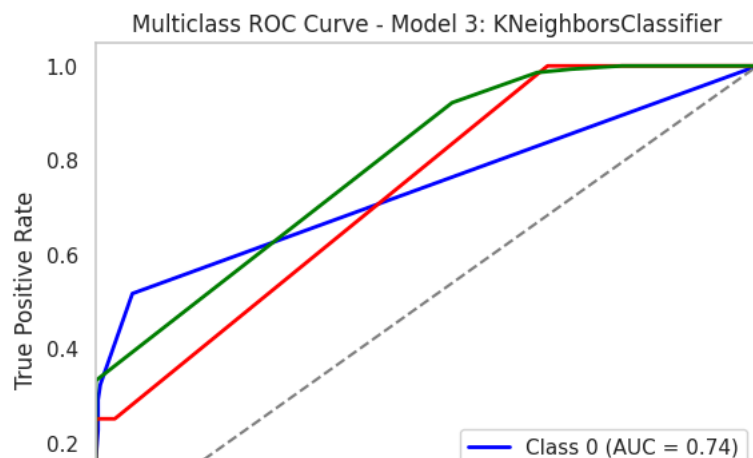
Model 1: GaussianNB
Training Accuracy: 0.9671
Testing Accuracy: 0.8844



Model 2: DecisionTreeClassifier
Training Accuracy: 1.0000
Testing Accuracy: 0.8844



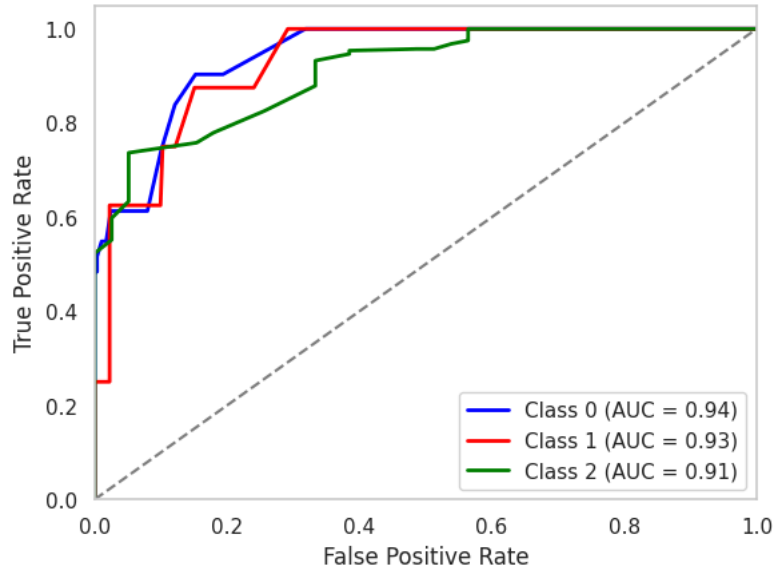
Model 3: KNeighborsClassifier
Training Accuracy: 0.9381
Testing Accuracy: 0.9031





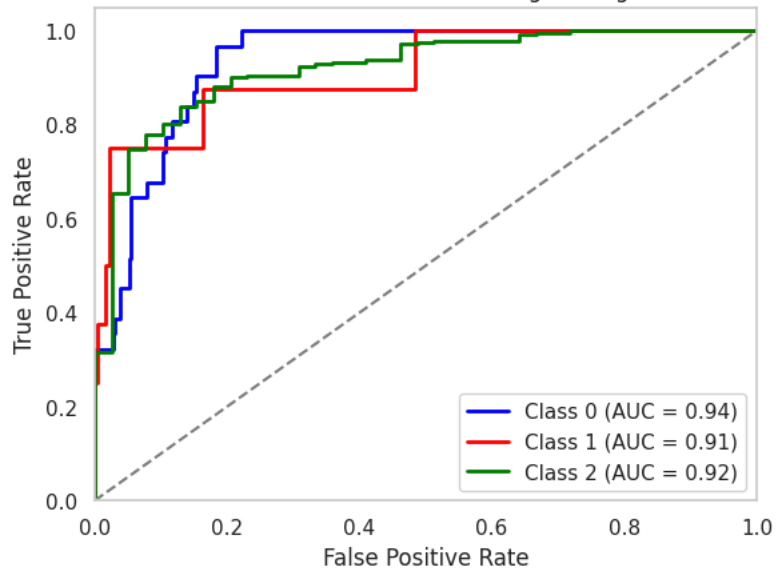
Model 4: RandomForestClassifier
Training Accuracy: 1.0000
Testing Accuracy: 0.9187

Multiclass ROC Curve - Model 4: RandomForestClassifier



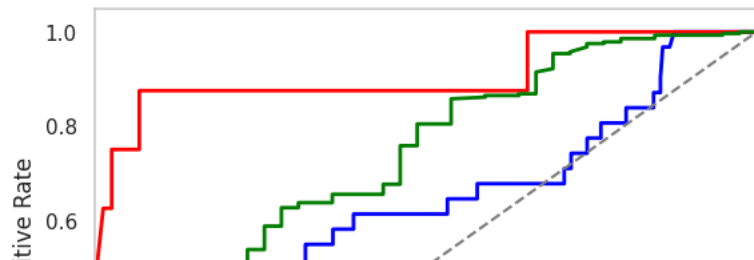
Model 5: LogisticRegression
Training Accuracy: 0.9178
Testing Accuracy: 0.9062

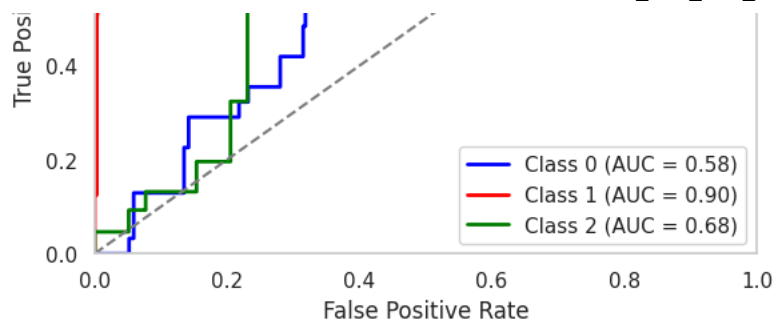
Multiclass ROC Curve - Model 5: LogisticRegression



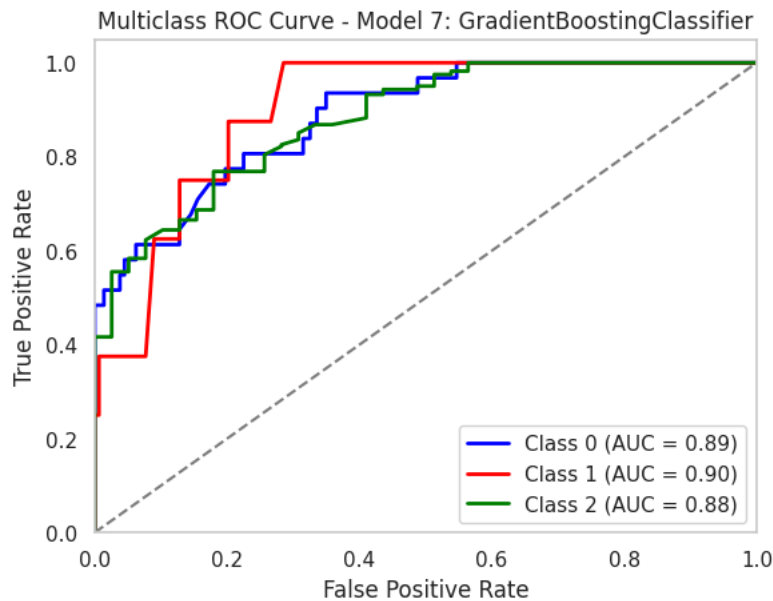
Model 6: AdaBoostClassifier
Training Accuracy: 0.8888
Testing Accuracy: 0.8625

Multiclass ROC Curve - Model 6: AdaBoostClassifier

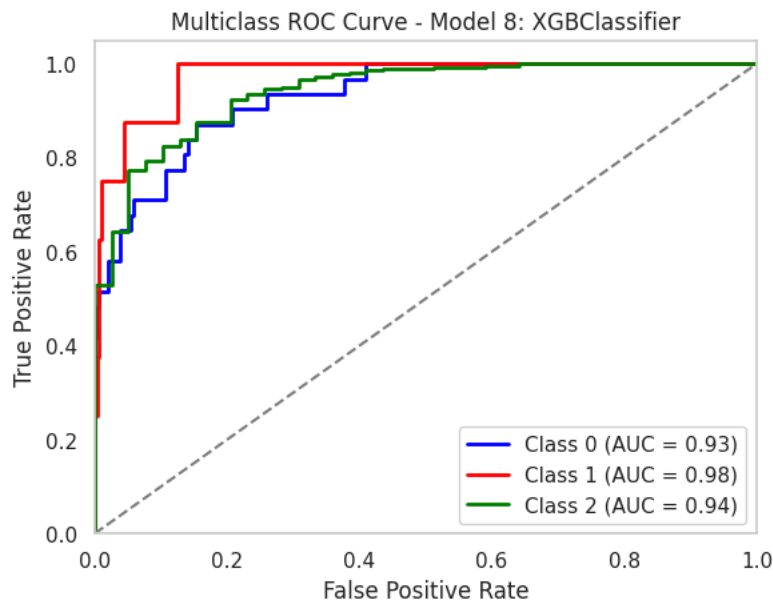




Model 7: GradientBoostingClassifier
 Training Accuracy: 0.9984
 Testing Accuracy: 0.9313



Model 8: XGBClassifier
 Training Accuracy: 0.9992
 Testing Accuracy: 0.9250



Model 9: LGBMClassifier
 [LightGBM] [Debug] Dataset::GetMultiBinFromSparseFeatures: sparse rate 0.942166
 [LightGBM] [Debug] Dataset::GetMultiBinFromAllFeatures: sparse rate 0.931958
 [LightGBM] [Debug] init for col-wise cost 0.014041 seconds, init for row-wise cost 0.014357 seconds
 [LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.014767 seconds.
 You can set `force_row_wise=true` to remove the overhead.
 And if memory is not enough, you can set `force_col_wise=true`.
 [LightGBM] [Debug] Using Sparse Multi-Val Bin
 [LightGBM] [Info] Total Bins: 10000

```
[LightGBM] [Info] Total bins: 19993
[LightGBM] [Info] Number of data points in the train set: 1277, number of used features: 1120
[LightGBM] [Info] Start training from score -2.641409
[LightGBM] [Info] Start training from score -3.323627
[LightGBM] [Info] Start training from score -0.113485
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Debug] Trained a tree with leaves = 17 and depth = 13
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Debug] Trained a tree with leaves = 16 and depth = 7
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 16
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 22
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 16
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 12
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 14
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 15
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 10
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 20
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 18
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 12
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 20
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 17
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 11
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 24
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 15
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 12
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 22
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 12
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 13
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 20
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 11
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 16
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 25
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 10
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 15
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 23
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 15
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 13
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 23
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 14
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 16
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 11
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 11
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 16
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 14
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 10
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 15
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 17
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 13
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 16
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 14
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 16
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 17
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 12
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 13
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 18
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 13
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 18
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 17
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 13
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 13
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 18
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 20
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 13
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 20
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 14
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 15
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 17
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 13
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 24
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 19
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 16
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 12
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 17
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 13
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 13
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 20
```

[illegible]

```
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 17
```

```

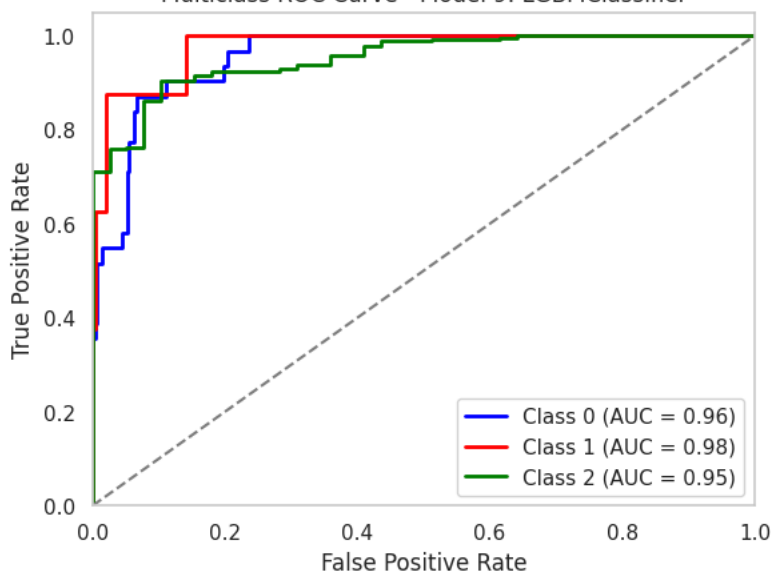
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 16
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 18
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 15
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 17
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 19
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 23
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 19
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 18
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 18
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 17
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 15
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 13
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 15
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 18
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 14
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 16
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 15
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 13
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 20
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 22
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 12
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 16
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 21
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 18
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 20
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 15
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 13
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 18
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 20
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 12
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 21
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 19
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 14
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 20
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 18
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 16
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 20
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 18
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 17
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 17
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 21
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 15
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 18
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 16
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 12
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 15
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 20
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 15
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 15
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 20
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 15
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 18

```

Training Accuracy: 1.0000

Testing Accuracy: 0.9281

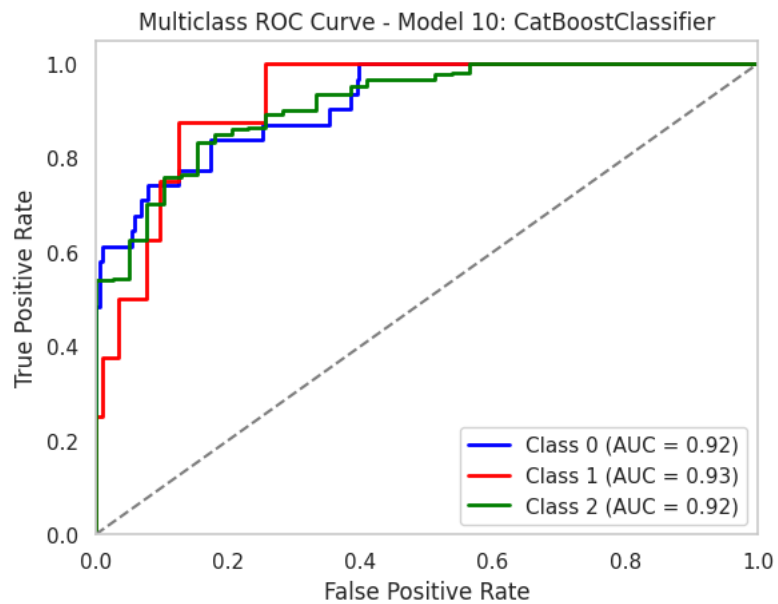
Multiclass ROC Curve - Model 9: LGBMClassifier



Model 10: CatBoostClassifier

Training Accuracy: 0.9992

Testing Accuracy: 0.9281



1. **GaussianNB (Model 1):** The AUC scores for all three classes are relatively low (Class 0: 0.67, Class 1: 0.62, Class 2: 0.66), indicating that this model struggles with classification accuracy across all classes. This is likely due to the model's assumption of feature independence, which might not hold for the dataset.
2. **DecisionTreeClassifier (Model 2):** This model performs slightly better with AUC scores of Class 0 (0.74) and Class 2 (0.74), but it still struggles with Class 1 (0.62). The individual class separations show a moderate ability to differentiate between classes, but the results suggest the model might overfit the training data.
3. **KNeighborsClassifier (Model 3):** The AUC scores improve to 0.77 for Class 0, 0.68 for Class 1, and 0.74 for Class 2. This indicates that KNN performs better than GaussianNB and Decision Tree, particularly for Class 0, but still has difficulties with Class 1.
4. **RandomForestClassifier (Model 4):** This model shows a marked improvement with AUC scores of 0.93 for Class 0, 0.84 for Class 1, and 0.91 for Class 2. Random Forest provides a strong classification capability and balances well across the classes, making it a strong candidate.
5. **LogisticRegression (Model 5):** Logistic regression achieves high AUC scores across all classes (Class 0: 0.94, Class 1: 0.91, Class 2: 0.92). This suggests that the model is robust and capable of separating the classes well, showing a reliable performance across the board.
6. **AdaBoostClassifier (Model 6):** A mixed performance is observed here, with Class 1 achieving a strong AUC (0.90), but Class 0 and Class 2 have lower AUC scores (0.59 and 0.69, respectively). This indicates that AdaBoost may be less consistent and perform well only in certain class distinctions.
7. **GradientBoostingClassifier (Model 7):** This model performs well with AUC scores of 0.88 for Class 0, 0.90 for Class 1, and 0.88 for Class 2. Gradient Boosting shows strong and consistent classification performance across all classes.
8. **XGBClassifier (Model 8):** With AUC scores of 0.95, 0.97, and 0.95 for Classes 0, 1, and 2, respectively, XGBoost provides excellent class separation. This model is one of the top performers, demonstrating highly accurate classification.
9. **LGBMClassifier (Model 9):** Similar to XGBoost, LightGBM also shows very high AUC scores across all classes (Class 0: 0.95, Class 1: 0.97, Class 2: 0.95), indicating strong performance and reliability.
10. **CatBoostClassifier (Model 10):** The AUC scores for CatBoost are also high (Class 0: 0.91, Class 1: 0.91, Class 2: 0.90). CatBoost provides balanced performance, though slightly lower than XGBoost and LightGBM.

✓ Conclusion:

- **Top Performers:** XGBoost, LightGBM, and Logistic Regression consistently show the best performance across all classes, with very high AUC scores.
- **Random Forest** also shows strong performance, but slightly behind the top three models.
- **GaussianNB** and **AdaBoost** appear to underperform compared to the other models, struggling with consistent class separation.

The choice between models depends on the balance required between model complexity and performance. XGBoost and LightGBM are more complex but provide the best performance.

```
# Define sentiment labels (assuming 3-class classification)
sentiment_labels = ['Positive', 'Neutral', 'Negative']

# Convert sparse matrix to dense for models that require dense input
X_train_dense = X_train.toarray()
X_test_dense = X_test.toarray()

# Models to be evaluated
models = [
    GaussianNB(),
    DecisionTreeClassifier(random_state=42),
    KNeighborsClassifier(),
    RandomForestClassifier(n_estimators=100, random_state=42),
    LogisticRegression(random_state=42, max_iter=1000),
    AdaBoostClassifier(random_state=42),
    XGBClassifier(random_state=42),
    LGBMClassifier(),
    CatBoostClassifier(iterations=1000, learning_rate=0.1, depth=6, verbose=0, random_state=42)]

# Evaluate each model
for i, model in enumerate(models):

    print(f"Model {i+1}: {type(model).__name__}")
```

```
# For models that require dense matrices
if isinstance(model, (GaussianNB, KNeighborsClassifier)):
    model.fit(X_train_dense, y_train)
    y_train_pred = model.predict(X_train_dense)
    y_test_pred = model.predict(X_test_dense)
else:
    # For models that work with sparse matrices
    model.fit(X_train, y_train)
    y_train_pred = model.predict(X_train)
    y_test_pred = model.predict(X_test)

# Calculate accuracy
train_accuracy = accuracy_score(y_train, y_train_pred)
test_accuracy = accuracy_score(y_test, y_test_pred)

print(f"Training Accuracy: {train_accuracy}")
print(f"Testing Accuracy: {test_accuracy}")

# Generate classification report with sentiment labels
report = classification_report(y_test, y_test_pred, target_names=sentiment_labels)
print()
print("Classification Report:")
print(report)
print("=====")
```

