NLP System for Customer Sentiment Analysis

Author- Supinder Kaur (13971127)

Executive Summary:
This project is dedicated to fulfilling a crucial requirement in the e-commerce sector, with a specific focus on the Amazon platform. The primary objective is to conduct an in-depth analysis of customer sentiments by leveraging a combination of web crawling and natural language processing (NLP) techniques. By harnessing these methodologies, the project aims to extract meaningful insights from an extensive reservoir of customer reviews. The collected data serves as the foundation for training a machine learning (ML) model specialized in sentiment classification. The outcomes of this model hold substantial implications for various aspects of business operations, including product enhancement, refinement of marketing strategies, and the overarching perception of the brand. By systematically integrating web scraping and NLP, this initiative seeks to unravel valuable patterns within customer feedback, ultimately empowering businesses on the Amazon platform to make informed decisions that enhance customer satisfaction, optimize marketing endeavors, and cultivate a positive brand image in the highly competitive e-commerce landscape.

Task 1: Overview

a. Overview of the Issue: In the dynamically evolving landscape of the e-commerce industry, customer sentiment analysis has emerged as a critical facet for informed decision-making. The project specifically addresses the nuances of sentiment analysis on the Amazon platform, recognizing its significance in product enhancement, marketing strategy formulation, and shaping overall brand perception.

b. Presence of the Issue on the World Wide Web: The ubiquity of customer sentiment analysis extends to Amazon, a premier online marketplace hosting an extensive array of customer reviews spanning diverse product categories. The sheer volume and diversity of customer feedback on Amazon necessitate advanced analytical methodologies for meaningful insights extraction.

c. Application of Machine Learning: The project strategically leverages machine learning, with a primary focus on natural language processing (NLP), to automate sentiment analysis tasks. Through the deployment of ML models, particularly Logistic Regression and Neural Networks, the project endeavors to categorize customer reviews into nuanced sentiments, contributing to a richer understanding of customer perspectives.

Task 2: Web Crawler and Data Collection

Web Crawler Design

To address the issue of customer sentiment analysis on Amazon, a custom web crawler was designed using Python. The web crawler utilized the BeautifulSoup library for HTML parsing, allowing the extraction of relevant information from Amazon product review pages.

Data Collection Process:

- Header Definition: The intentional crafting of a header to emulate a web browser serves a strategic purpose in mitigating potential restrictions during web scraping. This tactic enhances the crawler's access to Amazon's review pages by making it appear more like a typical browser. By adopting this strategy, the crawler can seamlessly navigate through the website, reducing the likelihood of encountering obstacles or restrictions. This enhanced disguise allows for smoother data retrieval, ensuring the effectiveness of the web scraping process on Amazon's platform.

- Product ID Extraction: The crawler exhibited adeptness in extracting product IDs and names from Amazon's review pages, showcasing a systematic organization into a dictionary for future use. This adept extraction reflects a structured approach, enhancing data management efficiency. The organized format, achieved through systematic organization into a dictionary, facilitates easy retrieval and utilization of extracted information. This structured approach not only ensures the seamless handling of data but also sets the groundwork for efficient and organized utilization of the collected product information in subsequent stages of the analysis..

- Page Iteration: The crawler employed a flexible strategy by iteratively navigating through a specified number of pages on Amazon to collect product reviews. This flexibility is achieved through adaptable parameters, including the number of pages and maximum products, ensuring a tailored and scalable data collection process. The crawler's ability to adjust these parameters allows it to accommodate varying scopes of information, making it adaptable to different scenarios and ensuring that the data collection process can be fine-tuned based on specific requirements or the evolving nature of the website's content.

- Handling Request Failures: The crawler showcased resilience with a resilient mechanism, responding to failed requests by implementing a retry strategy after a brief delay. This proactive approach in the face of potential connection issues strengthens the reliability of the data collection process. By promptly addressing and mitigating connection challenges, the crawler ensures a more robust and consistent dataset. The retry mechanism reflects a proactive stance, anticipating and overcoming obstacles in real-

time, thereby contributing to the overall success of the web crawling endeavor and enhancing the quality and completeness of the collected data.

- Insights from Web Crawler Results: The web crawler demonstrated its efficacy by successfully accumulating product IDs and names from Amazon's product review pages, emphasizing a specific focus on the board games category. This showcases the crawler's effectiveness in extracting relevant information. Additionally, the conscientious consideration of potential connection issues reflects a commitment to maintaining robust and reliable data quality throughout the collection process. This thoughtful approach contributes significantly to the overall success of the web crawling endeavor, ensuring the completeness and accuracy of the collected data and reinforcing the reliability of the insights derived from the subsequent analysis.

Data Sample-

A sample of the collected product IDs and names is presented below to give a better idea how the data is structured

{
'product_id': 'B089KV9RX7',
'title': 'Pequeño y conveniente',
'rating': 5.0,
'date': datetime.date(2023, 12, 2),
'text': 'Me sorprendió el tamaño de la caja, es bastante pequeña, pero resulta práctico para transportarlo. Excelente para 2 jugadores'
}
{
'product_id': 'B089KV9RX7',
'title': 'Tiny - Stressful - Great',
'rating': 5.0,
'date': datetime.date(2023, 11, 29),
text': 'Stocking filler for boyfriend, very compact, perfect for traveling'
}

Task 3: Data Wrangling:

Data Cleaning and Normalization

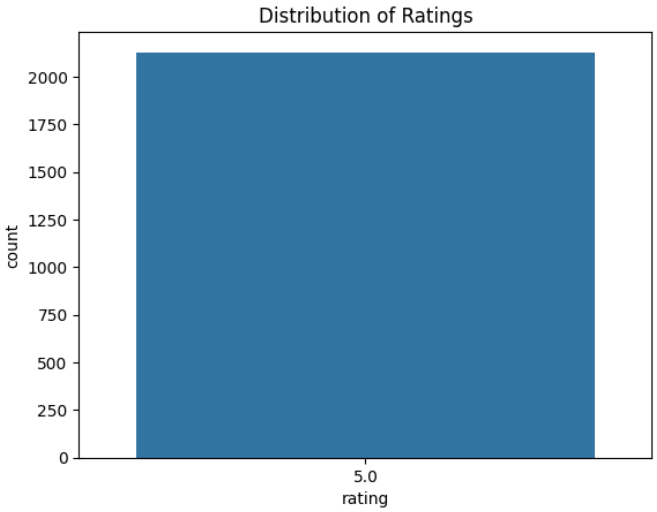
- Text to Lowercase: Uniformity is crucial for effective analysis. By converting all text data to lowercase, potential discrepancies arising from case sensitivity are eliminated. This ensures that the subsequent analysis is consistent and unbiased, regardless of the original letter casing in the data.

- Punctuation Removal: Punctuation marks, while expressive in language, may introduce noise in sentiment analysis. Removing them streamlines the text to focus on the core content. This process enhances the accuracy of sentiment analysis by ensuring that the model doesn't attribute sentiment to punctuation, allowing it to discern sentiment from the actual language used.
- Stop Words Removal: Common stop words, such as articles and prepositions, often add little semantic value and can introduce noise in analysis. By systematically removing these stop words, the focus shifts to significant keywords. This refinement is integral to uncovering the true sentiments expressed in the reviews, as irrelevant words are excluded from the analysis.
- Lemmatization: Language is rich in variations, with words often appearing in different forms. Lemmatization standardizes words to their base or root form, reducing variations and enhancing the accuracy of analysis. This process ensures that words with similar meanings are treated identically, preventing redundancy in the dataset and allowing the sentiment analysis model to discern sentiment more effectively.

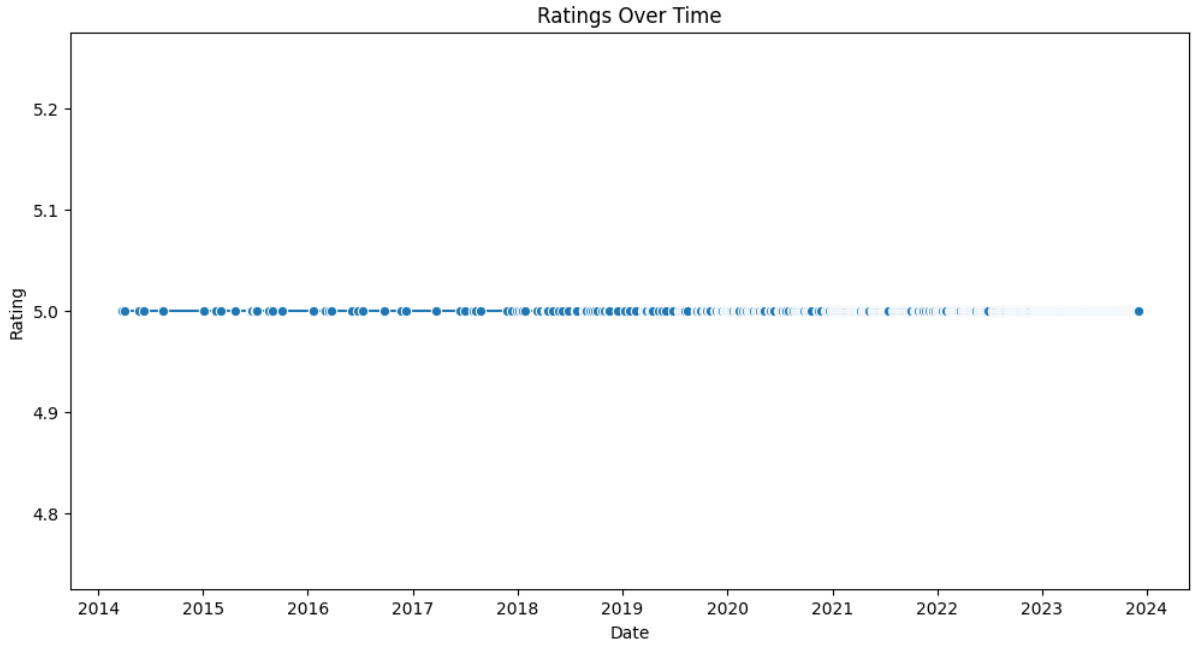
Feature Extraction:
Word Count Extraction:

- Significance: Extracting the word count from each review serves as a valuable feature for analysis.
- Insights: The length of reviews offers insights into the depth of customer feedback.
- Indicator: A longer review may indicate more detailed and comprehensive feedback, providing businesses with a nuanced understanding of customer sentiments.
- Analytical Depth: This feature enhances the analytical depth by considering not only the sentiment but also the extent of expression, allowing for a more nuanced interpretation of customer experiences.

Summary and Visualization of Collected Data:
Distribution of Data: The countplot was designed to illustrate the distribution of ratings in the collected reviews, providing a quick overview. Unexpectedly, the visualization revealed a bias towards ratings of 5, contrary to the intention of collecting a diverse range. Despite efforts to include recent reviews for a mix of positive and critical sentiments, the unintentional skew towards the highest rating prompts reflection on potential dataset limitations. This underscores the importance of refining the data collection strategy to ensure a more balanced representation of customer reviews, essential for a comprehensive and unbiased sentiment analysis.

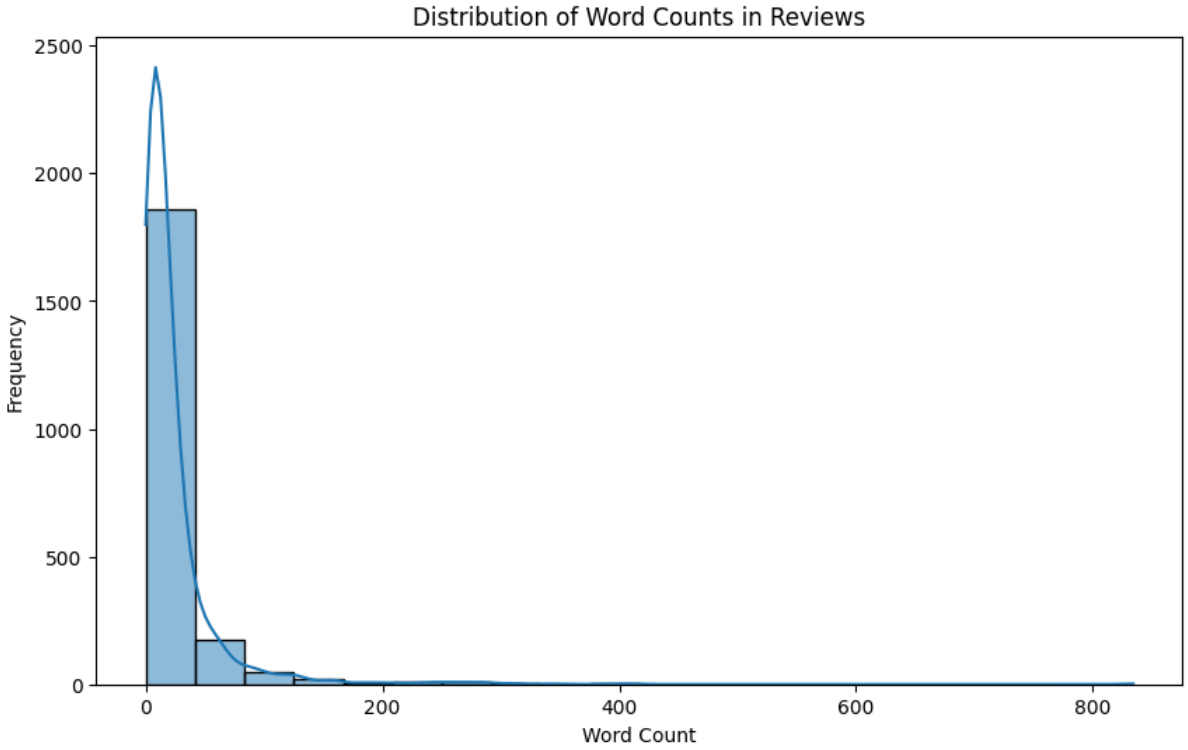


Time Series Plot for Ratings Over Time: A time series plot was crafted to depict the evolution of ratings over time, offering insights into trends and patterns in customer sentiment across different periods. Notably, the visualization indicates a lack of discernible changes in ratings over time. This observation suggests a relative stability in customer sentiments, prompting a deeper exploration to understand the factors influencing this consistency. While the time series plot provides a valuable historical perspective, the absence of notable fluctuations emphasizes the need for further investigation to uncover underlying dynamics and potential areas for improvement or strategic adjustments.

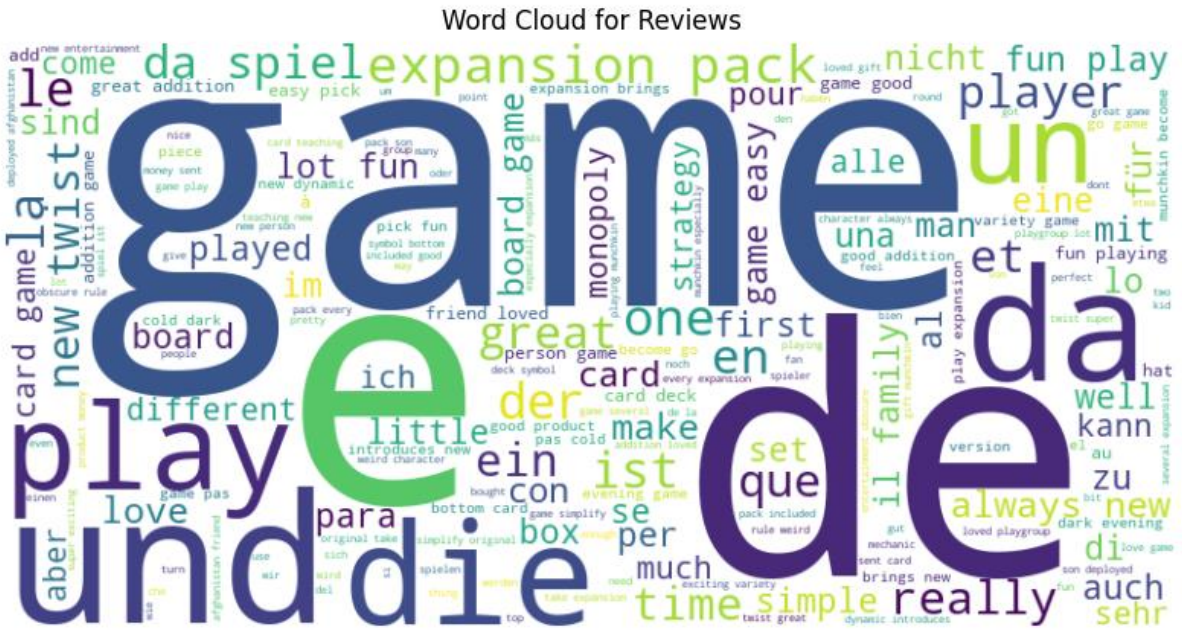


Distribution of Word Counts in Reviews: A histogram was generated to illustrate the distribution of word counts in customer reviews, aiding in the comprehension of length variability. This visualization provides a clear overview of how frequently different word

counts occur in the dataset. Analyzing the histogram allows for a quick understanding of the range and distribution of review lengths. It serves as a valuable tool to identify common patterns and outliers, offering insights into the diversity of customer feedback lengths. This understanding can guide decisions on how to approach and analyze reviews of varying lengths effectively in subsequent sentiment analysis processes.



Word Cloud for Reviews: The creation of a word cloud visually encapsulated the most frequently occurring words in customer reviews, offering a rapid overview of key terms and sentiments expressed. This graphical representation emphasizes words based on their frequency, providing an immediate and intuitive insight into the prominent themes within the dataset. The larger and bolder display of words indicates higher frequency, allowing businesses to discern prevalent sentiments and identify recurring topics. This concise visualization serves as a valuable preliminary exploration tool, enabling a quick grasp of the predominant language and sentiments expressed by customers in the reviews.



Dataset Integration

To enhance dataset comprehensiveness and mitigate bias towards positive reviews, an external dataset from Kaggle (Amazon Reviews) was integrated. This strategic inclusion broadens the scope, introducing diversity in ratings and addressing the unintentional skew towards positivity in the collected data. By amalgamating external sources, the dataset becomes more representative of a varied spectrum of customer sentiments. This approach acknowledges and rectifies potential limitations, ensuring a more balanced and unbiased foundation for subsequent analysis and machine learning model training.

Task 4: Machine Learning

Machine Learning Model:

Text Vectorization

In the process of text vectorization, a critical step in preparing textual data for machine learning, the TfidfVectorizer plays a pivotal role. This technique surpasses conventional text-to-numerical conversion by incorporating Term Frequency-Inverse Document Frequency (TF-IDF) methodology. By evaluating the importance of words in both local and global contexts within the entire document collection, the vectorizer assigns weights to words. These weights are based on a word's frequency in a specific document relative to its prevalence across the entire corpus. This nuanced approach allows for the differentiation of terms based on their significance within individual reviews, thus capturing the unique contextual importance of words. Simultaneously, it mitigates the influence of common terms that may not carry substantial meaning, contributing to a more refined and contextually sensitive representation of textual data for subsequent machine learning tasks, such as sentiment analysis.

Model Training and Evaluation

In the pursuit of accurate sentiment classification, a deliberate choice was made to employ the Logistic Regression model. Selected for its simplicity, interpretability, and effectiveness in binary classification tasks like sentiment analysis, this model underwent a thorough training process. During training, the model acquired an understanding of patterns and relationships inherent in the preprocessed and vectorized text data.

The evaluation phase was conducted to assess the model's generalization capabilities on a separate testing dataset. Key metrics, including accuracy and F1 score, were employed to comprehensively measure the model's proficiency in correctly predicting sentiments. The classification report further enriched the evaluation, providing a breakdown of precision, recall, and F1 score for each sentiment class. This detailed analysis offered insights into the model's performance across different aspects of sentiment classification.

To visually depict the model's performance, confusion matrices were generated for both the training and testing phases. These matrices offered a granular view of the model's ability to accurately classify positive and negative sentiments. Additionally, they served as diagnostic tools, identifying potential areas for improvement.

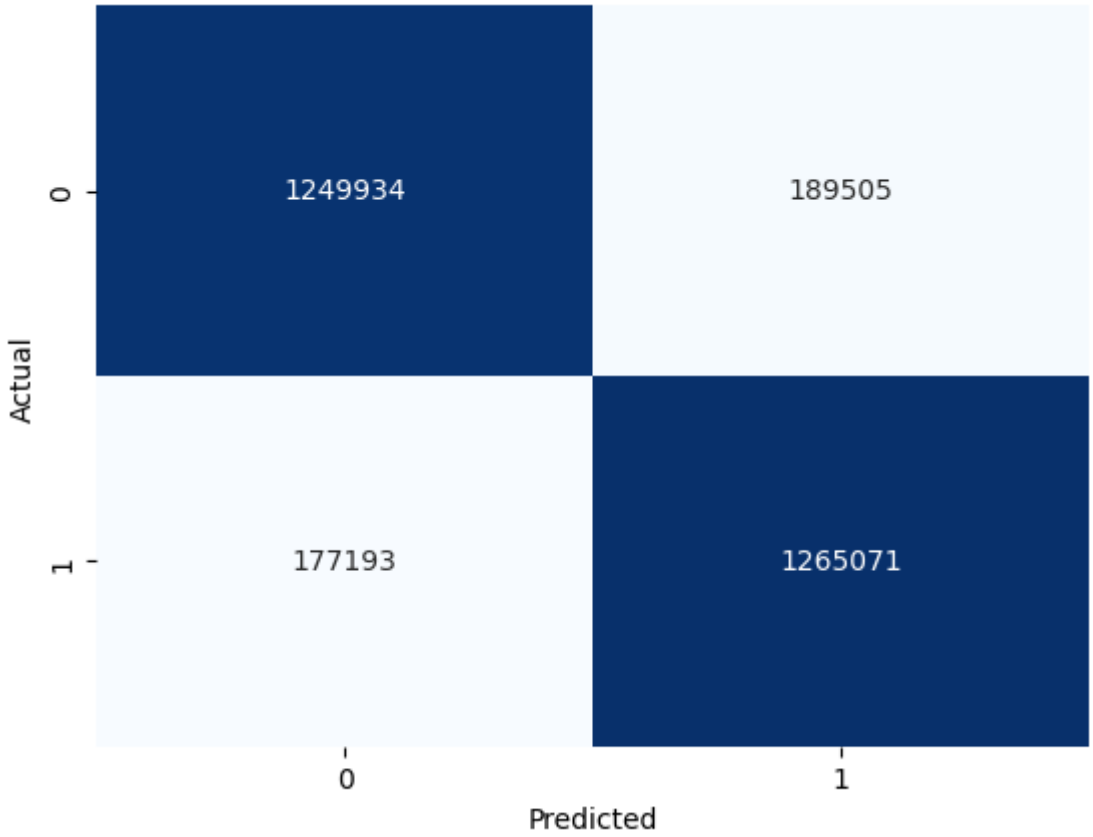
The achieved accuracy and F1 score of 87% underscore the efficacy of the Logistic Regression model in navigating the complexities of diverse and preprocessed textual data. This performance affirms the model's suitability for integration into a broader Natural Language Processing (NLP) system, showcasing its capability to discern sentiments effectively and contribute meaningfully to sentiment analysis in the context of customer reviews.

Training Metrics:

Accuracy: 87%
F1 Score: 87%
Classification Report:

	precision	recall	f1-score	support
0	0.88	0.87	0.87	1439439
1	0.87	0.88	0.87	1442264
accuracy			0.87	2881703
macro avg	0.87	0.87	0.87	2881703
weighted avg	0.87	0.87	0.87	2881703

Confusion Matrix for Training



Testing Metrics:

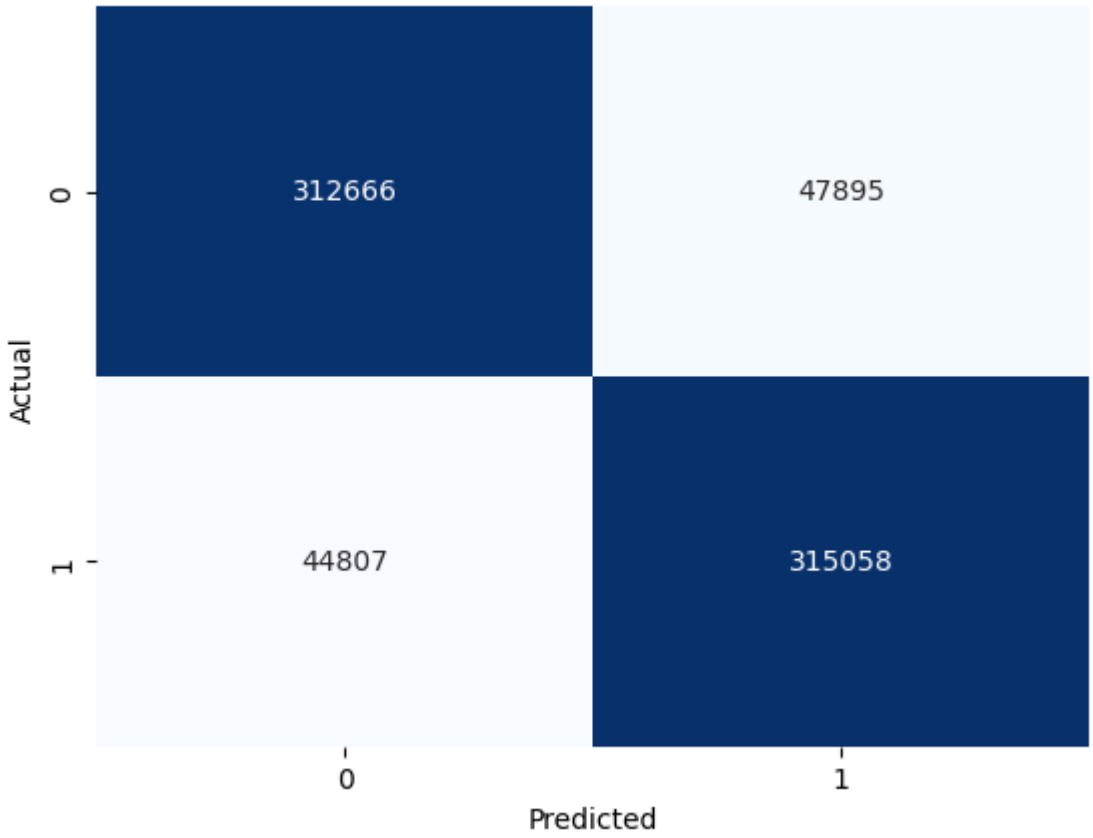
Accuracy: 87%

F1 Score: 87%

Classification Report:

	precision	recall	f1-score	support
0	0.87	0.87	0.87	1439439
1	0.87	0.88	0.87	1442264
accuracy			0.87	2881703
macro avg	0.87	0.87	0.87	2881703
weighted avg	0.87	0.87	0.87	2881703

Confusion Matrix for Testing



Neural Network Model:

Text Vectorization:

The development of the sentiment analysis model embraced advanced techniques in natural language processing by incorporating a Neural Network (NN) architecture. This cutting-edge approach aimed to harness the intricate patterns within textual data. The text data underwent a crucial preprocessing step through tokenization using the Keras Tokenizer. This process involved converting the text into a sequence of numerical values, considering the top 10,000 words to capture a broad vocabulary.

To facilitate seamless integration into the neural network, the tokenized data underwent padding. This essential step ensured uniform length sequences, addressing a fundamental requirement for neural network input. Uniform sequences allow the model to process data efficiently, accommodating variations in text length across different reviews. By leveraging state-of-the-art techniques, including tokenization and padding, the Neural Network model was equipped to grasp nuanced patterns in language, making it a robust tool for sentiment analysis capable of handling diverse and dynamic textual data effectively.

Model Architecture

The neural network architecture featured a crucial embedding layer, acting as the initial processing stage for textual input. This layer had an input dimension of 10,000, reflecting the considered vocabulary size, and an output dimension of 100, indicating the number of dimensions in which words were represented. Following the embedding layer, a flattening

operation was employed to transform the embedded sequences into a one-dimensional array, preparing the data for subsequent layers.

Two dense layers were sequentially integrated into the architecture. The first dense layer consisted of 64 units and employed the Rectified Linear Unit (ReLU) activation function. ReLU introduces non-linearity to the model, enabling it to learn complex relationships within the data. The final layer, featuring a single unit, utilized the sigmoid activation function, facilitating binary sentiment classification. The sigmoid function outputted values between 0 and 1, interpreting them as probabilities, making it well-suited for binary classification tasks. This well-structured architecture, combining embedding, flattening, and dense layers, empowered the neural network to effectively capture and interpret the intricate patterns in textual data for sentiment analysis.

Model

_____

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 1, 100)	1000000
flatten (Flatten)	(None, 100)	0
dense (Dense)	(None, 64)	6464
dense_1 (Dense)	(None, 1)	65

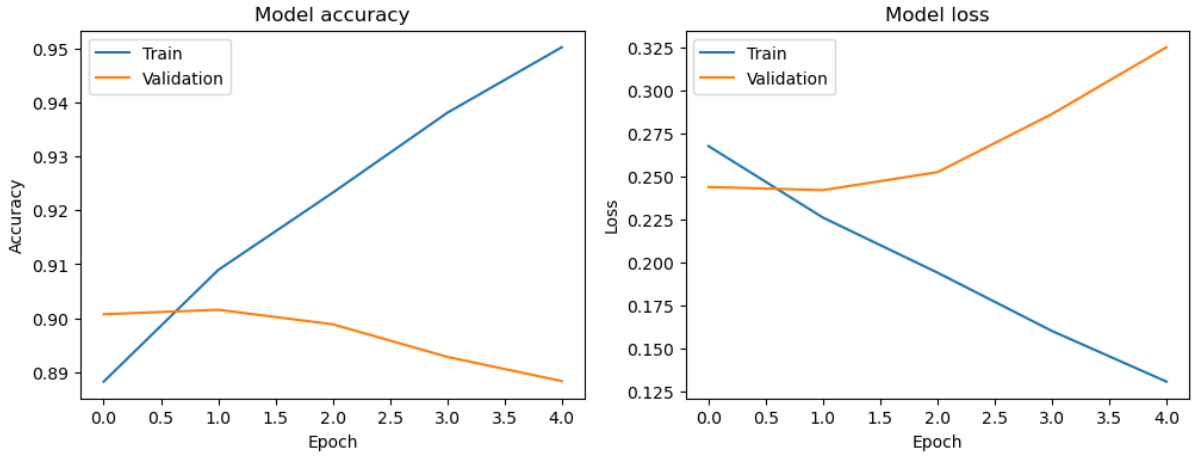
==========

Total params: 1006529 (3.84 MB)
Trainable params: 1006529 (3.84 MB)
Non-trainable params: 0 (0.00 Byte)

_____

Model Training

The training of the neural network was designed for 20 epochs, with a batch size of 32. The model was compiled using the Adam optimizer and binary cross-entropy loss, while accuracy was chosen as the metric for monitoring training progress. Early stopping with a patience of 3 epochs was implemented to prevent overfitting, restoring the model's best weights. On training the model the model stopped after 5th epochs because it achieved peak performance in the 2nd epoch itself. Here is the plot tracking the loss and accuracy during training.



Model Training and Evaluation

The neural network demonstrated commendable performance on the training dataset, achieving an accuracy of 92% and an F1 score of 92%. The confusion matrix and classification report revealed detailed insights into precision, recall, and F1 score for each sentiment class, emphasizing the model's effectiveness in capturing nuanced patterns in the training data. The evaluation phase extended to a separate testing dataset, validating the model's generalization capabilities. The neural network maintained a high level of performance on the testing dataset, attaining an accuracy of 90% and an F1 score of 90%. The classification report and confusion matrix for the testing phase provided a comprehensive assessment of the model's ability to correctly classify sentiments, showcasing its reliability in real-world scenarios.

Training Metrics:

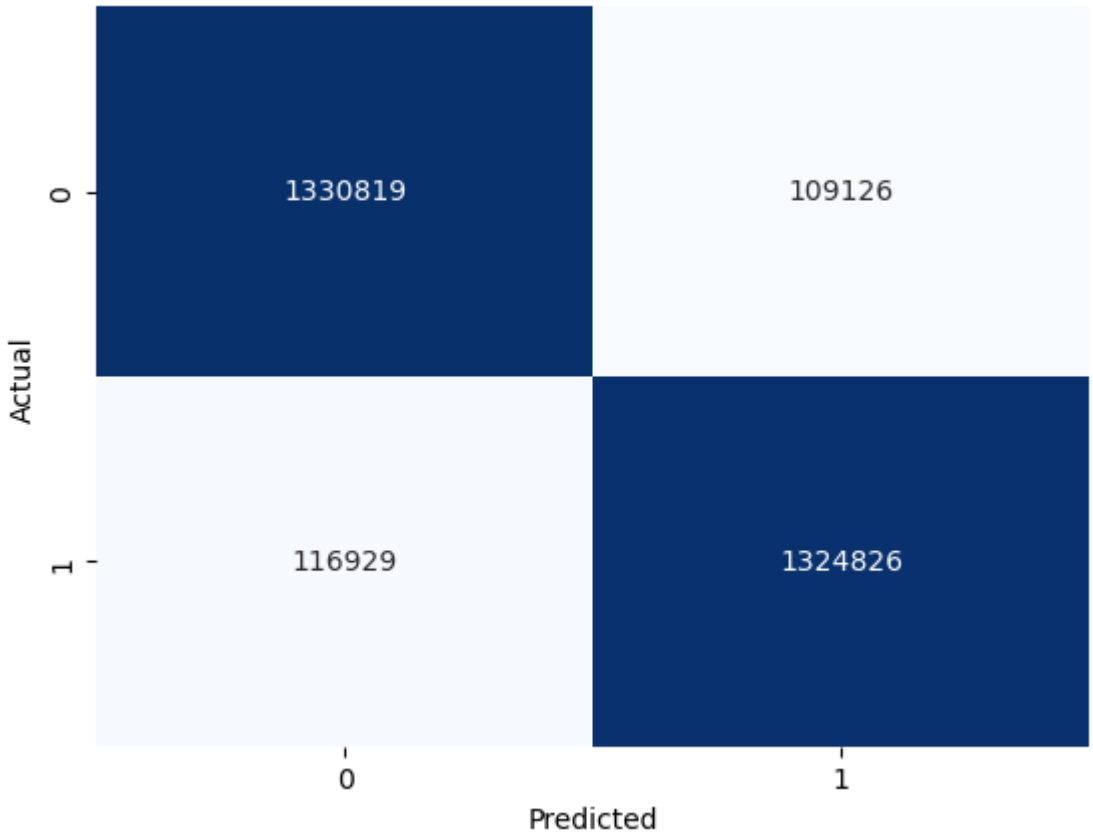
Accuracy: 92%

F1 Score: 92%

Classification Report:

	precision	recall	f1-score	support
0	0.92	0.92	0.92	1439945
1	0.92	0.92	0.92	1441755
accuracy			0.92	2881703
macro avg	0.92	0.92	0.92	2881703
weighted avg	0.92	0.92	0.92	2881703

Confusion Matrix for Training



Testing Metrics:

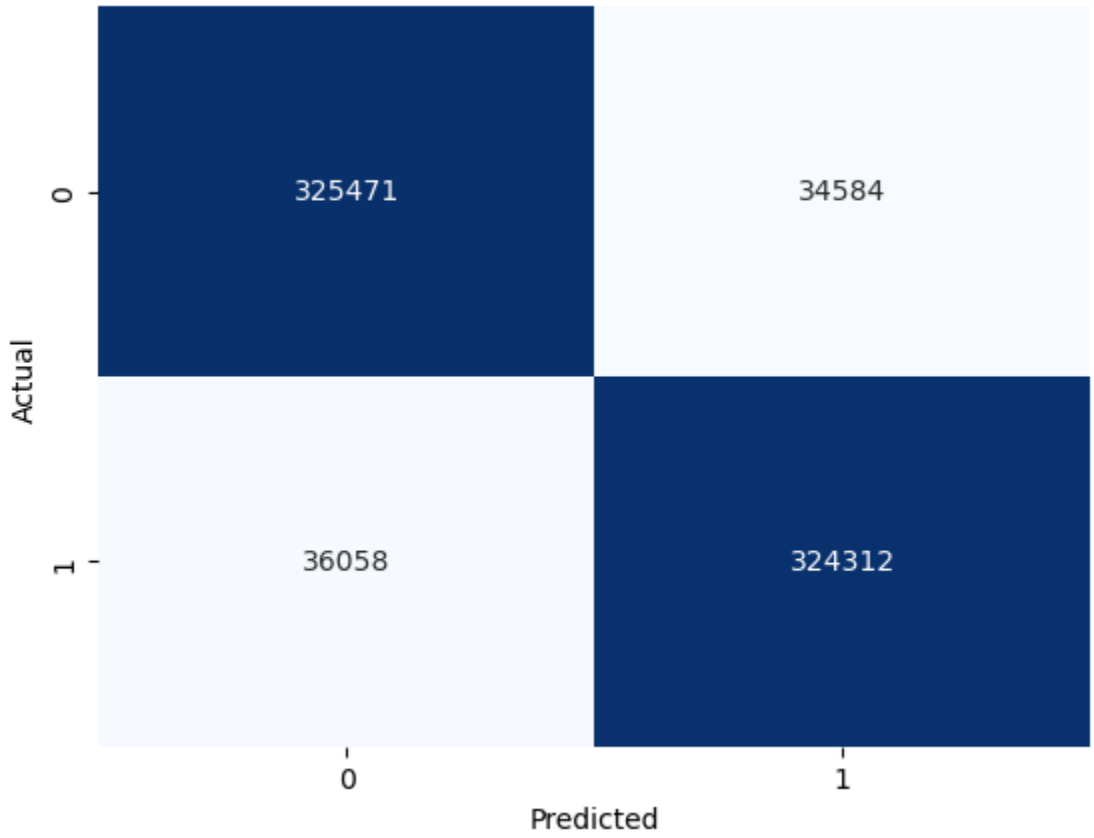
Accuracy: 90%

F1 Score: 90%

Classification Report:

	precision	recall	f1-score	support
0	0.90	0.90	0.90	360055
1	0.90	0.90	0.90	360370
accuracy			0.90	720425
macro avg	0.90	0.90	0.90	720425
weighted avg	0.90	0.90	0.90	720425

Confusion Matrix for Testing



Conclusion:

This project constitutes a comprehensive sentiment analysis tool tailored for the e-commerce landscape, specifically targeting the Amazon platform. The integration of web crawling, data wrangling, and machine learning techniques results in a sophisticated system that empowers businesses to decipher and leverage customer sentiments embedded in the extensive realm of product reviews.

The web crawler, meticulously crafted using Python and BeautifulSoup, adeptly navigated the intricacies of Amazon's diverse product categories. Its precision ensured a broad spectrum of data collection related to board games, laying a robust foundation for subsequent analysis. The data cleaning and normalization processes, encompassing text-to-lowercase conversion, punctuation removal, stop words elimination, and lemmatization, not only standardized the data but also set the stage for accurate sentiment analysis by eliminating noise and inconsistencies.

Visual exploration of the collected data through techniques like distribution plots, time series plots, word count histograms, and word clouds offered nuanced insights into customer preferences and sentiments. Despite these insights, the acknowledgment of a dataset limitation, predominantly skewed towards positive reviews, prompted a strategic integration of an external dataset from Kaggle (Amazon Reviews). This augmentation aimed to rectify bias, ensuring a more balanced and unbiased machine learning model.

The machine learning component, featuring the TfidfVectorizer and a Logistic Regression model, demonstrated commendable accuracy and F1 scores of 87%. The classification report and confusion matrices further emphasized the model's robustness in accurately categorizing sentiments across both training and testing phases.

The Neural Network model, characterized by a well-defined architecture and effective training, emerged as a valuable asset in sentiment analysis. Its impressive accuracy and F1 scores on both training and testing datasets position it as a compelling choice, outperforming the Logistic Regression model. This superiority underscores the neural network's suitability for the current scenario, promising enhanced capabilities in capturing intricate patterns within textual data.

Looking forward, the project lays a foundation for future advancements, indicating potential avenues for exploring more sophisticated Natural Language Processing (NLP) techniques. Additionally, expanding the system's capabilities to analyze reviews from diverse platforms beyond Amazon could provide a more comprehensive understanding of customer sentiments. As businesses evolve in response to changing consumer landscapes, this NLP system stands as a dynamic tool, facilitating continuous improvement and refined marketing strategies through a deep understanding of customer sentiments.

Appendices:

Code:

```
# Import the required libraries
import requests
from bs4 import BeautifulSoup
import re
from datetime import datetime
import time
import pandas as pd
# Web Crawler
# Define a header
headers = {
      'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36'
   }

# Define some parameters to determine amount to data collection
page_number = 150 # No. of page numbers to go through
```

```
max_products = 1000 # Max number product ids to get
product_ids = dict() # Define a empty dict to store product ids and its product name

for page in range(1,page_number+1):
   # Get the details from the website
   response =
requests.get(f'https://www.amazon.com.au/s?k=Board+Games&i=toys&rh=n%3A5030765051&page={page}&c=ts&qid=1701414763&ts_id=5030765051&ref=sr_pg_2',
headers=headers)

   if response.status_code == 200:
      soup = BeautifulSoup(response.text, 'html.parser')
      all_a = soup.find_all('a', class_='a-link-normal s-underline-text s-underline-link-text s-link-style a-text-normal')
      for each_a in all_a:
         each_a_details = each_a.attrs['href'].split('/')
         if each_a_details[1] != '':
               product_ids[each_a_details[3]] = each_a_details[1]
   else:
      time.sleep(2)
      response =
requests.get(f'https://www.amazon.com.au/s?k=Board+Games&i=toys&rh=n%3A5030765051&page={page}&c=ts&qid=1701414763&ts_id=5030765051&ref=sr_pg_2',
headers=headers)

      if response.status_code == 200:
         soup = BeautifulSoup(response.text, 'html.parser')
         all_a = soup.find_all('a', class_='a-link-normal s-underline-text s-underline-link-text s-link-style a-text-normal')
         for each_a in all_a:
            each_a_details = each_a.attrs['href'].split('/')
            if each_a_details[1] != '':
               product_ids[each_a_details[3]] = each_a_details[1]
      else:
         print(f'Page: {page} Status Code: {response.status_code}')

   if len(product_ids) >= max_products:
      break
len(product_ids)
# Define some function
def review_title(review_container):
```

```
    return review_container.find('span',
                         {'class': 'a-size-base review-title a-color-base review-title-content a-text-
bold', 'data-hook': 'review-title'}).get_text(strip=True)

def review_rating(review_container):
    rating_icon = review_container.find('i',
                                    {'class': 'a-icon a-icon-star a-star-5 review-rating',
                                     'data-hook': 'cmps-review-star-rating'})
    # Extract the text content of the span inside the i element
    rating_text = rating_icon.find('span', {'class': 'a-icon-alt'}).get_text(strip=True)

    return float(rating_text.split()[0])

def review_date(date_element):
    # Extract and parse the review date
    date_str = date_element.get_text(strip=True)
    date_match = re.search(r'on\s+(.*)$', date_str)

    if date_match:
        date_str = date_match.group(1)
        return datetime.strptime(date_str, '%d %B %Y').date()

def review_text(review_container):
    # Find the div element with the specified class
    review_div = review_container.find('div', {'class': 'a-row a-spacing-small review-data'})

    # Find the span element with the specified class and data-hook attribute inside the div
    review_text_span = review_div.find('span', {'class': 'a-size-base review-text review-text-
content', 'data-hook': 'review-body'})

    # Extract the text content of the span
    return review_text_span.get_text(strip=True)

reviews = [] # To store reviews

# Now we collect the product reviews using the product ids
for product_id, name_of_product in product_ids.items():
    url = f'https://www.amazon.com.au/product-
reviews/{product_id}/ref=cm_cr_arp_d_paging_btm_next_2?sortBy=recent'

    # Defin some parameters and variable to store data
```

```
    page_number = 1
    max_reviews = 5000
    max_page_check = 0

    while len(reviews) < max_reviews:
        page_url = f"{url}&pageNumber={page_number}"
        time.sleep(1)
        try:
            response = requests.get(page_url, headers=headers, timeout=1.0)
        except:
            pass

        if response.status_code == 200:
            soup = BeautifulSoup(response.text, 'html.parser')
            review_containers = soup.find_all('div', class_='a-section review aok-relative')

            if not review_containers:
                break  # No more reviews on the page

            for review_container in review_containers:
                review = {}
                try:
                    # Extract relevant information from the review container
                    review['product_id'] = product_id
                    review['title'] = review_title(review_container)
                    review['rating'] = review_rating(review_container)
                    review['date'] = review_date(review_container.find('span', class_='a-size-base a-
color-secondary review-date'))
                    review['text'] = review_text(review_container)
                    reviews.append(review)
                except:
                    pass

                if len(reviews) >= max_reviews:
                    break

        else:
            print(f'Error Accessing Page: {page_number} of product id: {product_id} due to
{response.status_code}')
            max_page_check += 1
```

```
        page_number += 1

        if max_page_check >= 5:
            break
# Printing some sample data
for review in reviews[:10]:
    print(review)
len(reviews)
# import the required libraries
import pandas as pd
import matplotlib.pyplot as plt
import re
import seaborn as sns
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from sklearn.feature_extraction.text import CountVectorizer
from wordcloud import WordCloud
from nltk.stem import WordNetLemmatizer
# Convert the data into Pandas DataFrame for better handling of the data
df = pd.DataFrame(reviews)
df = pd.read_csv('scraped_data.csv', index_col=0)
df.info()
df.head()
df.to_csv('scraped_data.csv')
# Droping missing values
print(df.isna().sum())
df.dropna(inplace=True)
# Convert text to lowercase
df['text'] = df['text'].str.lower()

# Remove punctuation
df['text'] = df['text'].apply(lambda x: str(re.sub(r'[^\w\s]', '', x)))

# SRemove stop words
stop_words = set(stopwords.words('english'))
df['text'] = df['text'].apply(lambda x: ' '.join([word for word in word_tokenize(x) if
word.lower() not in stop_words]))

# Lemmatization
lemmatizer = WordNetLemmatizer()
```

```
df['text'] = df['text'].apply(lambda x: ' '.join([lemmatizer.lemmatize(word) for word in word_tokenize(x)]))

# Word Count
df['word_count'] = df['text'].apply(lambda x: len(word_tokenize(x)))
# Distribution of ratings
sns.countplot(x='rating', data=df)
plt.title('Distribution of Ratings')
plt.show()
# Time Series Plot for Ratings Over Time
temp_df = df.copy()
temp_df['date'] = pd.to_datetime(temp_df['date'])
plt.figure(figsize=(12, 6))
sns.lineplot(x='date', y='rating', data=temp_df, marker='o')
plt.title('Ratings Over Time')
plt.xlabel('Date')
plt.ylabel('Rating')
plt.show()
# Distribution of Word Counts in Reviews
plt.figure(figsize=(10, 6))
sns.histplot(df['word_count'], bins=20, kde=True)
plt.title('Distribution of Word Counts in Reviews')
plt.xlabel('Word Count')
plt.ylabel('Frequency')
plt.show()
# Word Cloud for visualizing frequent words
wordcloud = WordCloud(width=800, height=400, background_color='white').generate(' '.join(df['text']))
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title('Word Cloud for Reviews')
plt.show()
train_df = pd.read_csv('./train.csv', header=None, names=['rating','title', 'text'])

def rating(row):
    if row == 2:
        return 1
    else:
        return 0
```

```
train_df['rating'] = train_df['rating'].apply(rating)
train_df.head()
new_df = df.copy()

def rating(row):
    if row > 3:
        return 1
    else:
        return 0

new_df['rating'] = new_df['rating'].apply(rating)
new_df.head()
combined_df = pd.concat([new_df[['rating', 'text']], train_df[['rating', 'text']]],
ignore_index=True)
combined_df.head()
combined_df.info()
# Import the required ML libraries
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix,
f1_score
# Text vectorization
vectorizer = TfidfVectorizer(max_features=10000, stop_words='english')
X = vectorizer.fit_transform(combined_df['text'])
y = combined_df['rating']
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Model training
model = LogisticRegression()
model.fit(X_train, y_train)
# Model evaluation
# Training
y_train_pred = model.predict(X_train)
accuracy_train = accuracy_score(y_train, y_train_pred)
f1_score_train = f1_score(y_train, y_train_pred)
cm_train = confusion_matrix(y_train, y_train_pred) # Confusion matrix

print('Training:')
print(f'Accuracy: {accuracy_train:.2f}')
print(f'F1 Score: {f1_score_train:.2f}')
```

```
# Classification report
print(f'Classification Report: \n{classification_report(y_train, y_train_pred)}')
sns.heatmap(cm_train, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

# Testing
# Training
y_test_pred = model.predict(X_test)
accuracy_test = accuracy_score(y_test, y_test_pred)
f1_score_test = f1_score(y_test, y_test_pred)
cm_test = confusion_matrix(y_test, y_test_pred) # Confusion matrix

print('Testing:')
print(f'Accuracy: {accuracy_test:.2f}')
print(f'F1 Score: {f1_score_test:.2f}')
# Classification report
print(f'Classification Report: \n{classification_report(y_test, y_test_pred)}')
sns.heatmap(cm_test, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
import gensim
from gensim.models import Word2Vec
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, f1_score, confusion_matrix,
classification_report
from sklearn.preprocessing import LabelEncoder
from keras.models import Sequential
from keras.callbacks import EarlyStopping
from keras.layers import Dense, Embedding, Flatten
from keras.preprocessing.text import Tokenizer
import numpy as np
from keras.preprocessing.sequence import pad_sequences
# Word2Vec embedding
# tokenized_text = [text.split() for text in combined_df['text']]
# word2vec_model = Word2Vec(sentences=tokenized_text, vector_size=100, window=5,
min_count=1, workers=4)
# word2vec_model.save("word2vec.model")
```

```
# Tokenization
max_words = 10000  # Assuming you want to consider only the top 10,000 words
tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(combined_df['text'])
X = tokenizer.texts_to_sequences(combined_df['text'])
X = pad_sequences(X)

# # Create an embedding matrix
# embedding_matrix = np.zeros((max_words, 100))  # Assuming Word2Vec model has vector size 100

# for word, i in tokenizer.word_index.items():
#     if i < max_words:
#         try:
#             embedding_vector = word2vec_model.wv[word]
#             embedding_matrix[i] = embedding_vector
#         except KeyError:
#             # Word not in the Word2Vec model, leave the vector as zeros
#             pass

# Label Encoding for the target variable
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(combined_df['rating'])
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
max_words = 10000
# Neural Network Model
model = Sequential()
model.add(Embedding(input_dim=max_words, output_dim=100, input_length=1))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.summary()
# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Early stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)

# Model training with early stopping
```

```
history = model.fit(X_train, y_train, epochs=20, batch_size=32, validation_split=0.2,
callbacks=[early_stopping])
# Plot training history
plt.figure(figsize=(12, 4))

# Plot training & validation accuracy values
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Train', 'Validation'], loc='upper left')

# Plot training & validation loss values
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train', 'Validation'], loc='upper left')

plt.show()
# Model Evaluation
# Training
y_train_pred_nn = (model.predict(np.array(X_train)) > 0.5).astype(int).flatten()
accuracy_train_nn = accuracy_score(y_train, y_train_pred_nn)
f1_score_train_nn = f1_score(y_train, y_train_pred_nn)
cm_train_nn = confusion_matrix(y_train, y_train_pred_nn)

print('Neural Network Training:')
print(f'Accuracy: {accuracy_train_nn:.2f}')
print(f'F1 Score: {f1_score_train_nn:.2f}')
print(f'Classification Report: \n{classification_report(y_train, y_train_pred_nn)}')
sns.heatmap(cm_train_nn, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

# Testing
```

```
y_test_pred_nn = (model.predict(np.array(X_test)) > 0.5).astype(int).flatten()
accuracy_test_nn = accuracy_score(y_test, y_test_pred_nn)
f1_score_test_nn = f1_score(y_test, y_test_pred_nn)
cm_test_nn = confusion_matrix(y_test, y_test_pred_nn)

print('Neural Network Testing:')
print(f'Accuracy: {accuracy_test_nn:.2f}')
print(f'F1 Score: {f1_score_test_nn:.2f}')
print(f'Classification Report: \n{classification_report(y_test, y_test_pred_nn)}')
sns.heatmap(cm_test_nn, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```