Amazon ReviewIQ

Amazon Reviews Analysis & Sentiment Classification Engine

Rishin Tiwari
Saloni Birthare
Premal Doshi

Text Analytics
Fall 2024

Table of Contents

ReviewIQ Sentiment and Insights Engine

1. Executive Summary

This project addresses a key challenge in e-commerce: accurately analyzing customer reviews to provide meaningful insights into product performance and customer sentiment. Online shopping is growing rapidly, with studies indicating that potential buyers heavily rely on existing customer feedback rather than ratings alone. To support this trend, we scraped Amazon reviews across four product categories - laptops, phones, earbuds, and TVs - to understand what customers like or dislike about specific products.

Our analysis employed machine learning models to classify the sentiment of reviews and used generative AI to create a comprehensive summary dashboard. This dashboard highlights sentiment trends, features customers appreciate or find problematic, and can even offers purchase recommendations. These insights not only simplify decision-making for consumers but also provide product managers with valuable feedback on product strengths and areas needing improvement. Ultimately, this solution can be adapted to other e-commerce platforms, enhancing both user experience and product development.

2. Problem Definition & Significance

Amazon ReviewIQ project focuses on two target clients: individual customers who rely on product reviews for purchasing decisions, and product managers or small-to-medium e-commerce businesses that lack advanced tools to analyze customer feedback. Customer reviews play a critical role in online shopping, as they provide insights beyond star ratings, influencing customer trust and purchasing behavior.

In 2021, PowerReviews found that virtually all online shoppers (99%) read reviews, and 96% specifically seek out negative reviews to get a balanced view of a product. Furthermore, BrightLocal's research revealed that half of consumers trust online reviews as much as personal recommendations. These findings highlight the significant impact of customer feedback on purchasing decisions. However, analyzing and extracting meaningful insights from vast volumes of reviews can be challenging, especially for smaller businesses with limited AI resources. This project aims to bridge that gap by providing actionable insights through sentiment analysis and AI-driven dashboards, helping customers make informed choices and enabling product managers to understand customer satisfaction and areas for improvement.

3. Prior Literature

Literature Overview

Several studies have explored AI-driven sentiment analysis and e-commerce applications:

- **LLM-Generated Labels**: Using LLMs to automatically label reviews, improving data preparation.

- **Text Embedding with RAG**: Combining RAG with LLMs for better context-aware sentiment analysis.
- **LLM Comparison in E-commerce**: Fine-tuning models like GPT-3.5 significantly enhances performance for sentiment analysis.
- **LangchainIQ**: Improves efficiency in processing complex queries for faster insights.
- **Sentiment Analysis with Generative AI**: GPT-4 excels in multi-class sentiment analysis with few-shot prompting.
- **RAG-LLMs for Business Insights**: Enhances review insights and customer service through RAG-LLMs.

Novelty of Approach

Our project enhances sentiment analysis by fine-tuning the Roberta, Mistral-7B and LLaMA 3.1 models with LoRA (Low-Rank Adaptation of Large Language Models) for better memory efficiency. This allows us to process large review datasets accurately. We integrate sentiment analysis, trend visualization, and AI-driven summarization into a user-friendly dashboard, offering insights on product sentiment, feature likes, and dislikes. This approach provides comprehensive, actionable insights for both consumers and e-commerce managers, supporting real-time decision-making in e-commerce.

4. Data Source & Preparation

Data Source:

The data was scraped from Amazon's e-commerce platform, focusing on electronics categories: laptops, phones, earbuds, and TVs. Using three web scraping scripts, we extracted product URLs, review details, and product information. The ASIN served as the common key across datasets, enabling a merge. A total of 175,000 reviews were gathered with fields like ASIN, review content, stars, reviewer, and verified purchase status. A manually labeled dataset of 200 reviews was prepared to evaluate ML model accuracy.

Data Cleaning:

Basic text preprocessing included handling missing values, correcting spelling errors, removing special characters, links, and emojis was performed on the reviews data. Additionally, we used SpaCy for advanced cleaning, tokenizing and analysis, including identifying key features and conducting parts-of-speech tagging. This helped in extracting meaningful patterns and improving the quality of data for analysis.

5. Exploratory Data Analysis & Visualizations

Distribution of Product Ratings vs Number of Reviews (Figure 4.1)

- 1 Star: 15,000
- 2 Stars: 5,000
- 3 Stars: 12,000
- 4 Stars: 25,000

- 5 Stars: 63,000

Average Ratings by Category (Figure 4.2)

- Earphones: 4.2
- Laptops: 4.0
- Mobile Phones: 3.8
- TVs: 3.9

Top-Rated Brands (Figure 4.3)

Samsung, Google, TCL, Sony, LG, and HP.

Common Words in Reviews (Figure 4.4)

- **Negative Ratings (1 & 2 Stars):** "phone," “work,"screen," "battery," "issue," "purchase."
- **Positive Ratings (4 & 5 Stars):** "phone," "battery," "quality," "love," "camera," “sound.”

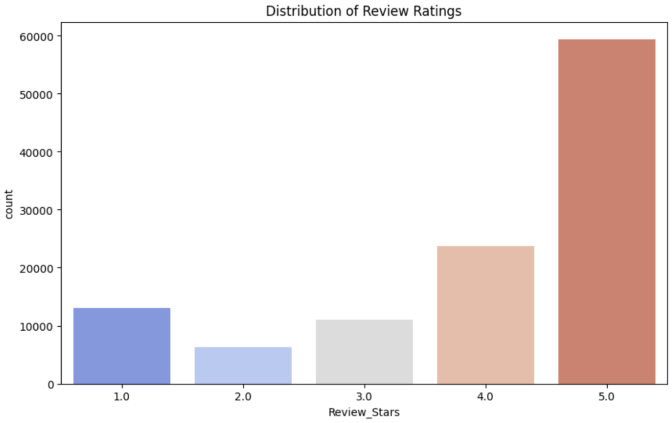


Figure 4.1

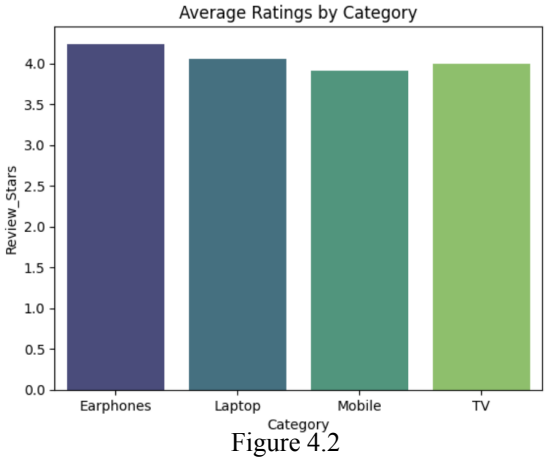


Figure 4.2

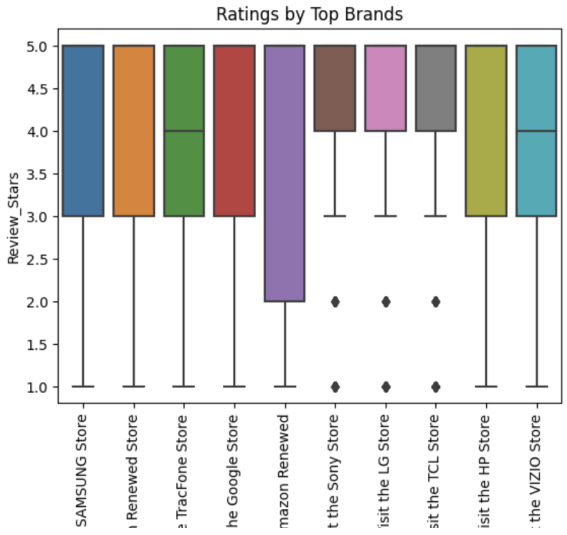


Figure 4.3

Figure 4.4

6. Text Analytics & Results

The following flowchart (figure 5.1) outlines the key stages of the text analytics process, from data collection and preprocessing to sentiment analysis and the creation of actionable insights through machine learning models and interactive dashboards.

Figure 5.1

To evaluate the effectiveness of various models in sentiment analysis, we tested multiple approaches, ranging from lightweight transformers to state-of-the-art large language models (LLMs). Each model was assessed on its ability to classify customer reviews into sentiment categories, with considerations for accuracy, computational efficiency, and suitability for binary or multi-class tasks. Below is a comparison of the performance, advantages, and challenges associated with each model:

Model Comparison Table

Model	Classification Type	Accuracy	Key Advantages	Challenges
DistilBERT	Binary (0, 1)	86%	Efficient, fast, clear sentiment distinction	Limited to binary classification
RoBERTa	Multi-class (0, 1, 2)	76%	Captures neutral sentiments for richer insights	Struggles with neutral sentiment classification

Fine-Tuned RoBERTa	Binary (0, 1)	96.4%	Best binary performance, high precision & recall	Limited to binary tasks
LLMs (Mistral-7B)	Multi-class (0, 1, 2)	41%	Advanced architecture	Lacks sentiment-specific fine-tuning
LLMs (LLaMA 3.1 - 8B)	Multi-class (0, 1, 2)	32%	Useful for Text generation only	Poor performance without fine-tuning
LLMs (GPT-3.5)	Multi-class (0, 1, 2)	68%	Adaptable for various tasks	Needs optimization for better accuracy
Fine-Tuned LLaMA (10k Data)	Multi-class (0, 1, 2)	87.3%	Accurate for nuanced classifications	Resource-intensive, computationally expensive

Summary of Results

- **Fine-Tuned RoBERTa** was the most accurate for binary tasks, achieving 96.4%.
- **Fine-Tuned LLaMA** delivered strong multi-class performance with 87.3% accuracy but was resource-heavy.
- **DistilBERT** balanced speed and accuracy for binary tasks, making it ideal for large-scale datasets.
- **LLMs** like GPT-3.5 showed potential but require extensive fine-tuning for sentiment-specific tasks.

RAG-Based Dashboard Overview

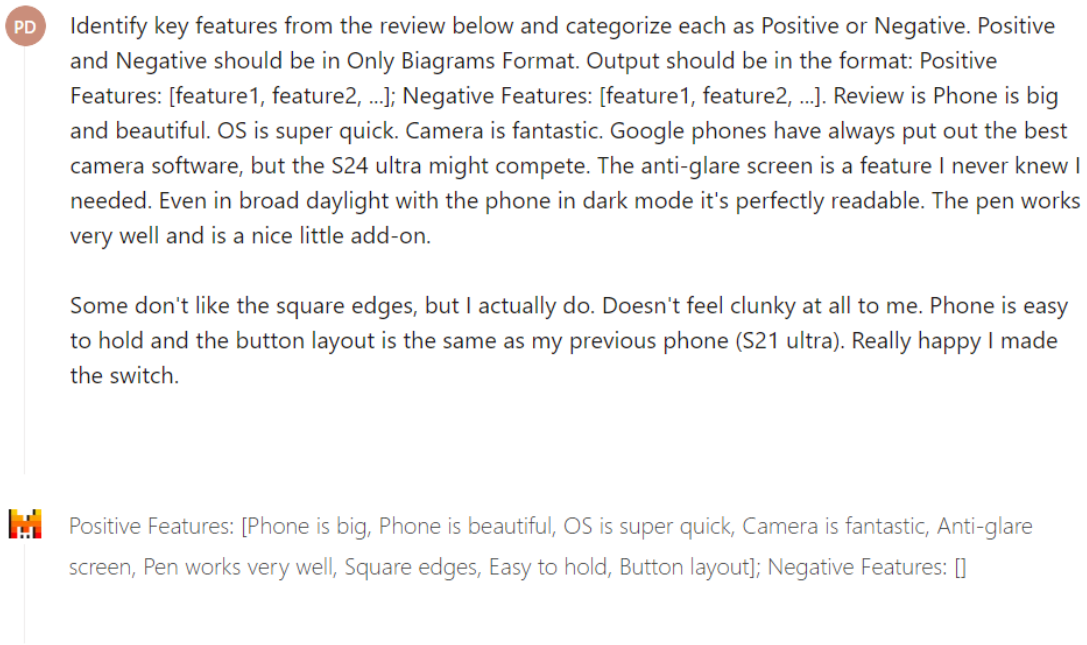
The project focuses on creating dashboards that analyze and visualize customer reviews using a Retrieval-Augmented Generation (RAG) approach. Mistral's API and other models (like RoBERTa) were leveraged for sentiment analysis and feature extraction, supplemented by interactive dashboards for insights.

Challenges Encountered

1. API Rate Limit: Encountered Status 429: Requests rate limit exceeded` when processing high volumes of reviews using Mistral's free API. This was mitigated by using pre-processed data for dashboard creation.
2. Real-Time Processing Issues: While the goal was to create real-time dashboards, limitations in API usage and response times necessitated using batch-processed data.

```
SDKError: API error occurred: Status 429
{"message":"Requests rate limit exceeded"}
```

Feature Extraction Functionality *(Code in Appendix)*
Function: get_mistral_feature_sentiment(review_text)
Purpose: Extracts positive and negative bi-gram features from reviews using the Mistral API.
Output: Lists of bi-grams categorized as Positive or Negative.



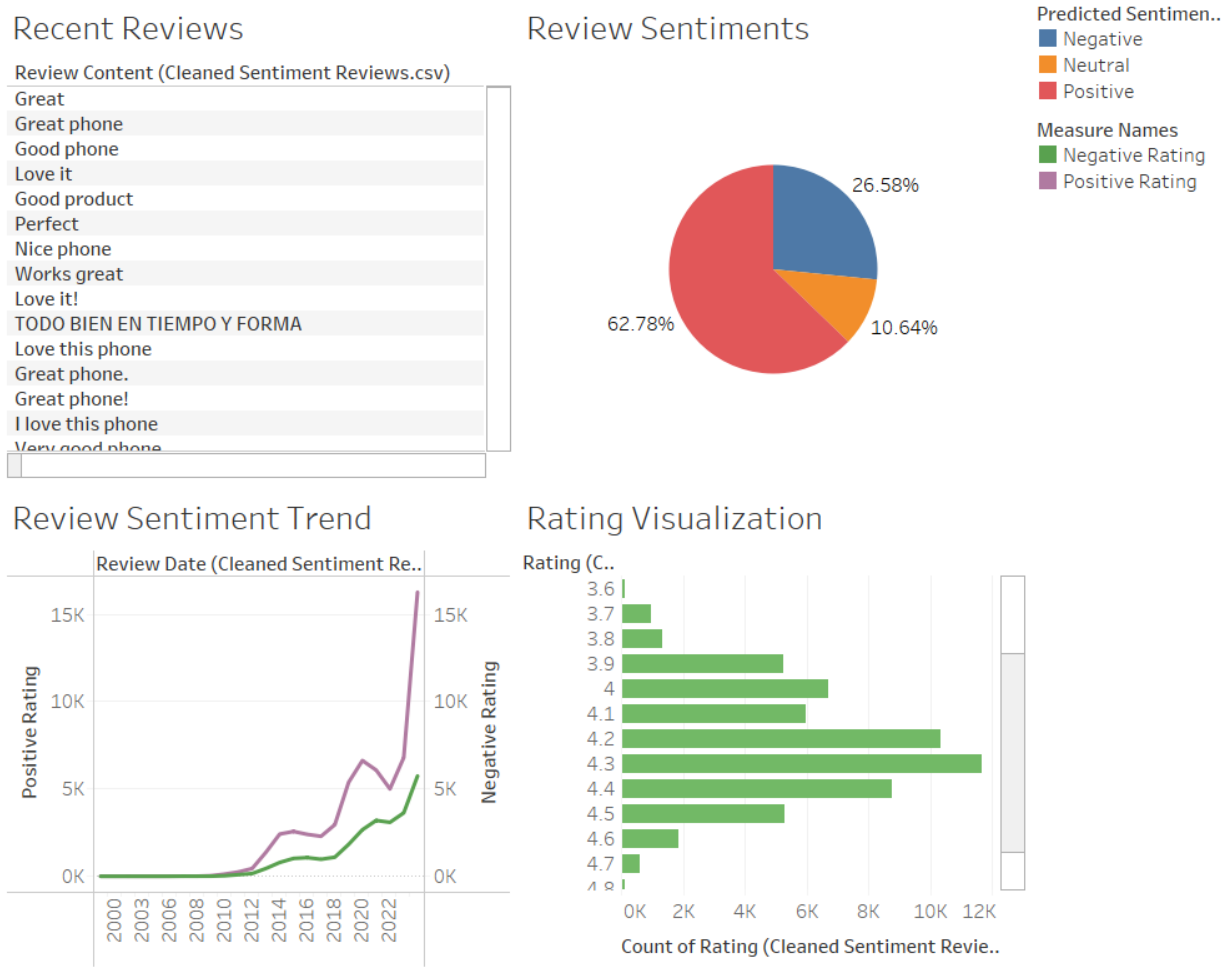
Dashboard Implementation
Three dashboards were developed using **Tableau** and **Dash**, each serving specific analytical purposes:

Dashboard 1: Sentiment and Rating Trends
1. Key Visualizations:
Review Sentiment Trends: Shows sentiment trends over time.

Rating Visualization: Displays distribution and changes in ratings.

Recent Reviews: Highlights the latest customer reviews.

Data Source: Outputs from RoBERTa model and pre-processed sentiment data.



Dashboard 2: Product and Regional Insights

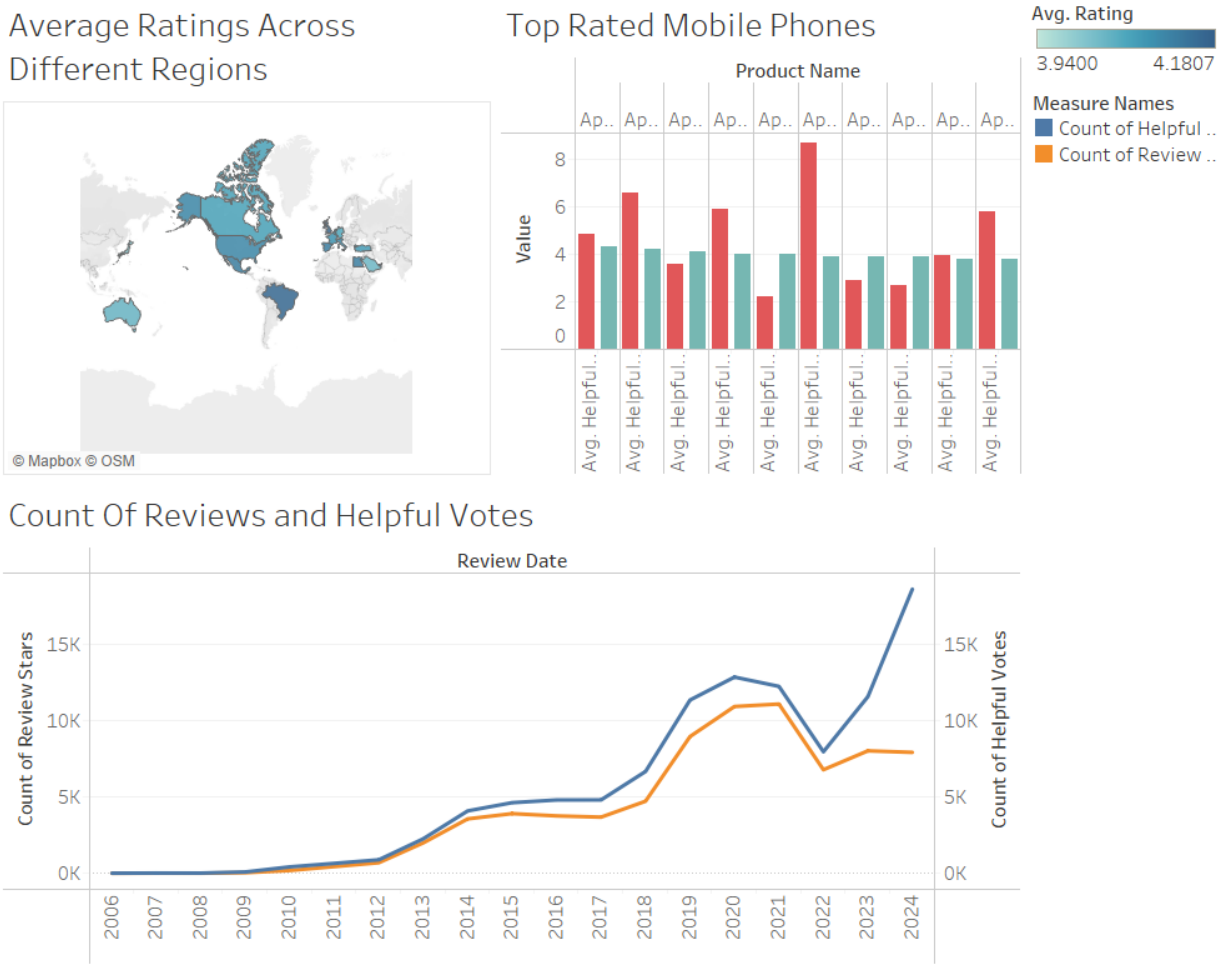
1. Key Visualizations:

Top-Rated Products: Highlights top-rated mobile phones or other product categories.

Average Ratings Across Regions: Showcases regional trends in customer feedback.

Count of Reviews and Helpful Votes: Quantifies customer engagement.

Data Source: Processed outputs (e.g., count metrics, regional ratings).



Dashboard 3: Feature Sentiment Analysis

Built with Python Dash Framework:

1. Functionality:

Input: Accepts product name and sentiment type (Positive/Negative).

Visualization:

Bar chart for Top 10 Features by count.

Pie chart for feature distribution.

Interactive Updates: Users can switch between sentiment types and products dynamically.

2. Code Explanation *(Code in Appendix)*:

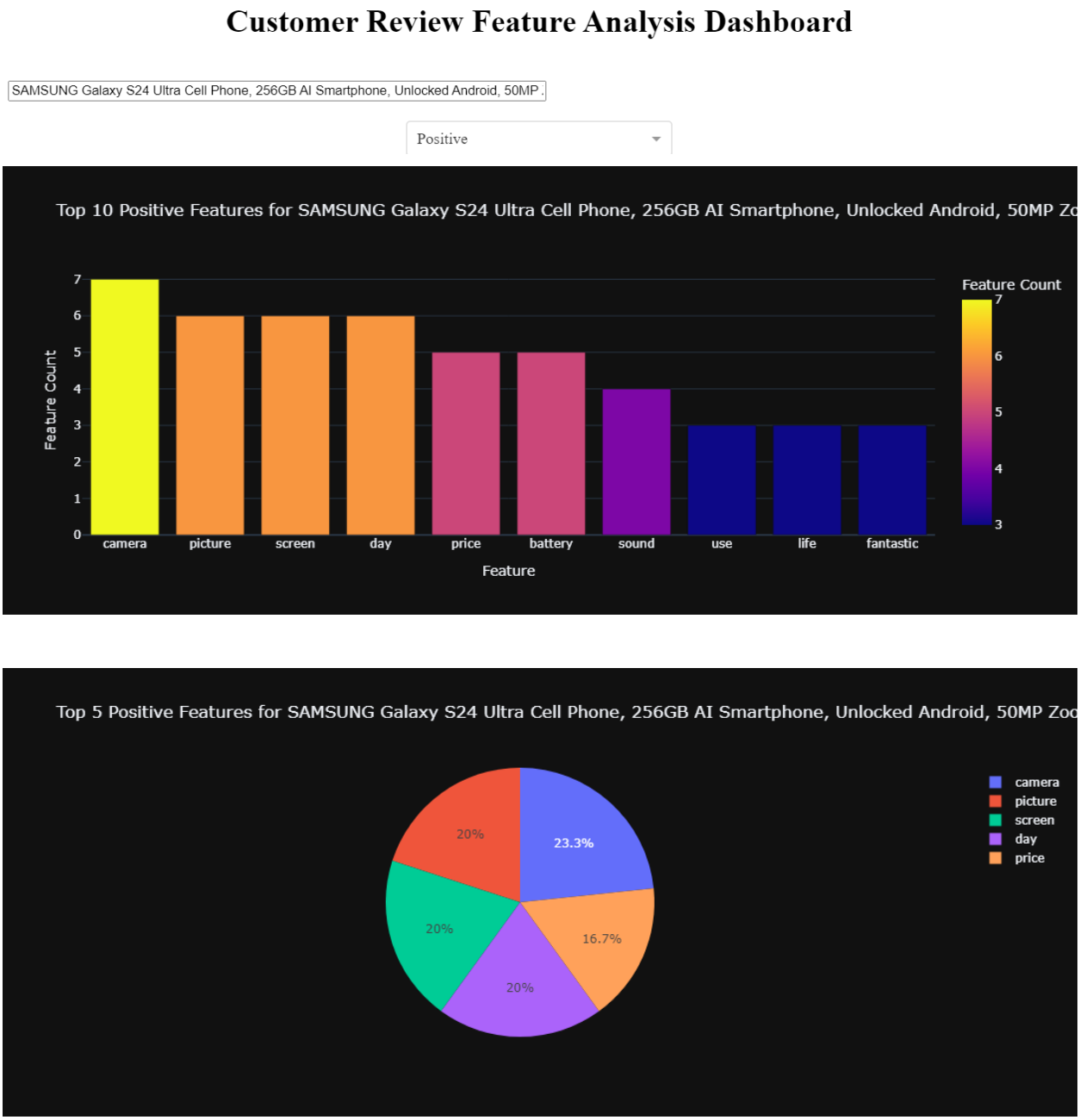
Data Preparation:

The function `get_product_features(product_name)` fetches top positive/negative features for a product by analyzing pre-processed data.

Visualization:

Utilizes `plotly.express` for creating bar and pie charts.

Callbacks: Updates charts dynamically based on user input via Dash's callback mechanism
Users need to select a Product Name and indicate whether the sentiment is Positive or Negative. The dashboard will then display the relevant features based on these selections.



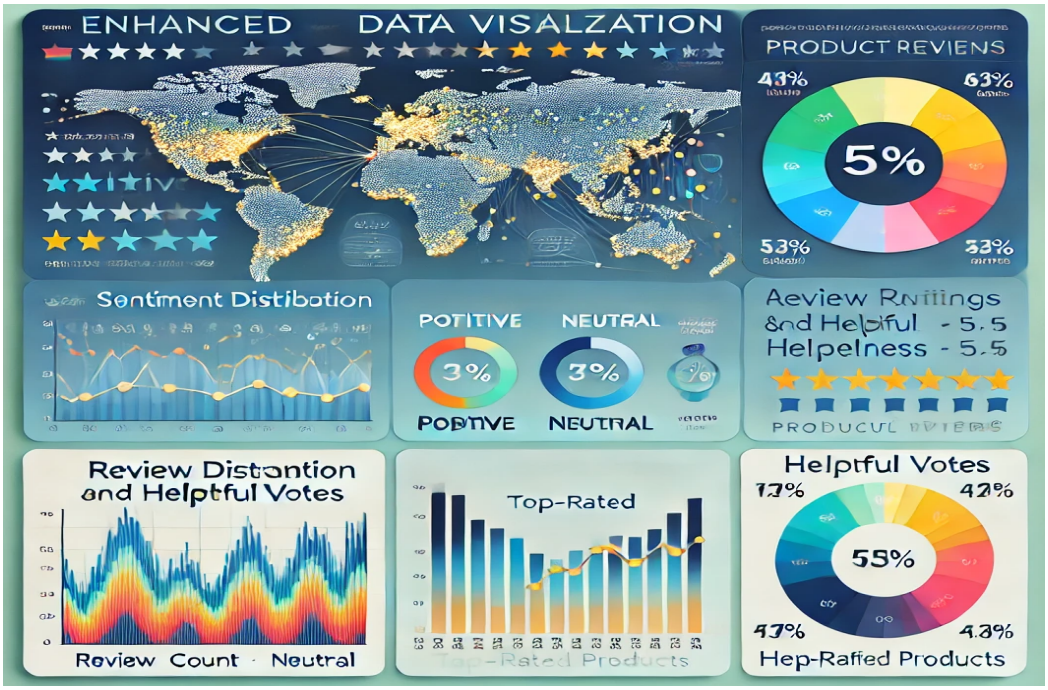
Future Enhancements :

Real-Time Integration: Overcome API limitations to enable live data analysis.

Scalability: Use premium APIs or optimize queries to handle higher volumes.

Advanced Visualizations: Incorporate more granular insights, like temporal trends in feature-specific sentiments.

For future enhancements, a GPT-powered dashboard or a Power BI dashboard integrated with Co-Pilot functionality could also be utilized. Currently, we have combined our dashboard capabilities using AI GPT, which generated a dashboard image similar to this.



Conclusion

Task-specific models like **DistilBERT** and **RoBERTa** outperform general-purpose LLMs in sentiment analysis due to their fine-tuning and efficiency. Fine-tuned **LLaMA** is promising for nuanced classifications but demands significant computational resources.

7. Insights & Recommendations

Based on our analysis of Amazon reviews using LLMs and interactive dashboards, the following actionable recommendations aim to assist both individual customers and product managers or small-to-medium e-commerce businesses.

Insights for Individual Customers

- **Informed Decision-Making**: Customers can leverage sentiment analysis to understand a product's strengths and weaknesses at a glance, saving time and effort. Positive reviews

emphasize product reliability, while negative reviews identify potential issues like quality or durability.

- **Building Trust and Credibility**: Highlight verified purchase reviews and customer engagement metrics to increase confidence in product authenticity and satisfaction.
- **Personalized Dashboards**: Introduce a user-friendly interface allowing customers to filter reviews by sentiment, common keywords, or specific product features.

Recommendations for Product Managers

- **Customer Insights**: Utilize the analysis to identify recurring positive themes (e.g., "battery life") to market key strengths effectively and address recurring issues like "durability" to improve product design.
- **Product Development**: Integrate feedback to prioritize enhancements or new features.
- **Market Trends and Competitor Analysis**: Analyze reviews to identify trends or gaps in competitor products, creating differentiation opportunities.
- **Sentiment Categorization and Dashboard Insights**: Use sentiment dashboards to monitor customer feedback in real-time, adapting strategies as needed.

Actionable Next Steps

- **Implement Enhanced Sentiment Analysis**: Adopt our fine-tuned RoBERTa model for binary sentiment classification (96.4% accuracy) to ensure trustworthiness and efficiency.
- **Engage Actively with Customers**: Regularly respond to reviews—addressing negative feedback builds trust, and acknowledging positive reviews fosters loyalty.
- **Invest in Predictive Analytics**: Incorporate models to predict future customer satisfaction trends based on historical data for proactive decision-making.

By following these recommendations, product managers can refine strategies, improve satisfaction, and build brand trust, while individual customers gain valuable insights for informed decisions. The integration of actionable dashboards ensures all stakeholders can efficiently analyze and act upon customer feedback.

8. References

i. Kohlenberg, L., Horns, L., Sadrieh, F., Kiele, N., Clausen, M., Ketterer, K., Navasardyan, A., Czinczoll, T., de Melo, G., & Herbrich, R. (2024). LLM-generated labels for product reviews. *arXiv*. Retrieved from https://arxiv.org/pdf/2410.12470

ii. Oluwasegun William, I., & Altamimi, M. (2024). Text embedding implementation using retrieval-augmented generation (RAG) model combined with large language model. *ResearchGate*. Retrieved from https://www.researchgate.net/publication/381105654_Text_Embedding_Implementation_Using_Retrieval_Augmented_Generation_RAG_Model_Combined_With_Large_Language_Model

iii. Roumeliotis, K. I., Tselikas, N. D., & Nasiopoulos, D. K. (2024). LLMs in e-commerce: A comparative analysis of GPT and LLaMA models in product review evaluation. *ScienceDirect*. Retrieved from https://www.sciencedirect.com/science/article/pii/S2949719124000049#d1e519

iv. Ghane, S., Sawant, R., Supe, G., & Pichad, C. (2024). LangchainIQ: Intelligent content and query processing. *International Journal of Modern Technology Studies*. Retrieved from https://supublication.com/index.php/ijmts/article/view/1351/972

v. Krugmann, J. O., & Hartmann, J. (2024). Sentiment analysis with generative AI. *Springer*. Retrieved from https://link.springer.com/article/10.1007/s40547-024-00143-4

vi. Arslan, M., Munawar, S., & Cruz, C. (2024). RAG-LLMs for business insights. *Taylor & Francis Online*. Retrieved from https://www.tandfonline.com/doi/full/10.1080/12460125.2024.2410040

9. Appendix

Python Code Snippets & Output

Github Repo : https://github.com/RishinTiwari/Amazon_ReviewIQ

Url Scraping :

```
import time
import pandas as pd
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.chrome.options import Options
from webdriver_manager.chrome import ChromeDriverManager
from selenium.webdriver.chrome.service import Service

# Set up Chrome options
chrome_options = Options()
chrome_options.add_argument('--headless')
chrome_options.add_argument('--disable-gpu')
chrome_options.add_argument("--start-maximized")
chrome_options.add_argument('--no-sandbox')
chrome_options.add_argument('--disable-dev-shm-usage')

# Initialize WebDriver
service = Service(ChromeDriverManager().install())
driver = webdriver.Chrome(service=service, options=chrome_options)
```

```
urls = []

# Function to scrape URLs from a search results page
def scrape_urls_from_page(url):
    driver.get(url)
    time.sleep(2)  # Allow time for the page to load

    try:
        # Locate all search result items on the page
        search_results = driver.find_elements(By.XPATH, "//div[@data-component-type='s-search-result']")

        for result in search_results:
            try:
                # Extract product URL from the result
                link_element = result.find_element(By.XPATH, ".//h2/a")
                link = link_element.get_attribute('href')
                if link:
                    print(link)
                    urls.append(link)
            except Exception as e:
                print(f"Error extracting link: {e}")

    except Exception as e:
        print(f"Error scraping page: {e}")

# Loop through multiple pages
for i in range(1, 201):  # Adjust the range as needed
        url = f"https://www.amazon.com/s?k=earbuds&s=exact-aware-popularity-rank&crid=PMF4Z2U5IFAC&qid=1726865099&sprefix=earbuds%2Caps%2C105&ref=sr_st_exact-aware-popularity-rank&page={i}"
    print(f"Scraping page {i}...")
    scrape_urls_from_page(url)

# Save URLs to a CSV file
df = pd.DataFrame({"Urls": urls})
df.to_csv("amazon_earbuds_urls.csv", index=False)

print(f"Scraped {len(urls)} URLs.")

# Close the WebDriver
driver.quit()
```

Reviews Scraping:

```
import time
import pandas as pd
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.chrome.options import Options
from webdriver_manager.chrome import ChromeDriverManager
from selenium.webdriver.chrome.service import Service
import re

# Set up Chrome options
chrome_options = Options()
chrome_options.add_argument('--headless')
chrome_options.add_argument('--disable-gpu')
chrome_options.add_argument('--start-maximized')
chrome_options.add_argument('--no-sandbox')
chrome_options.add_argument('--disable-dev-shm-usage')

# Initialize WebDriver
service = Service(ChromeDriverManager().install())
driver = webdriver.Chrome(service=service, options=chrome_options)

# Initialize lists to store scraped data
ASINs = []
Product_Names = []
Review_Titles = []
Review_Contents = []
Review_Stars = []
Review_Dates = []
Reviewer_Names = []
Verified_Purchase = []
Helpful_Votes = []
Review_URLs = []

# Load product URLs from CSV file
df = pd.read_csv("amazon_tv_urls.csv")

# Function to extract reviews from a single page
def extract_reviews(asin, product_name):
    try:
        # Scrape the review titles
```

```
            titles = driver.find_elements(By.XPATH, "//a[@data-hook='review-title']")
            Review_Titles.extend([title.text for title in titles])

            # Scrape the review content
            contents = driver.find_elements(By.XPATH, "//span[@data-hook='review-body']")
            Review_Contents.extend([content.text for content in contents])

            # Scrape the star ratings
            stars = driver.find_elements(By.XPATH, "//i[@data-hook='review-star-rating']//span")
            Review_Stars.extend([star.get_attribute('innerHTML').split()[0] for star in stars])

            # Scrape the review date
            dates = driver.find_elements(By.XPATH, "//span[@data-hook='review-date']")
            Review_Dates.extend([date.text for date in dates])

            # Scrape the reviewer names
            names = driver.find_elements(By.XPATH, "//span[@class='a-profile-name']")
            Reviewer_Names.extend([name.text for name in names])

            # Scrape the verified purchase status
            verified = driver.find_elements(By.XPATH, "//span[@data-hook='avp-badge']")
            Verified_Purchase.extend([v.text if v else 'Not Verified' for v in verified])

            # Scrape helpful votes
            helpful = driver.find_elements(By.XPATH, "//span[@data-hook='helpful-vote-statement']")
            Helpful_Votes.extend([help.text if help else '0' for help in helpful])

            # Append the current review URL to Review_URLs list
            Review_URLs.extend([driver.current_url] * len(titles))

            # Append the current ASIN and product name to ASINs and Product_Names list
            ASINs.extend([asin] * len(titles))
            Product_Names.extend([product_name] * len(titles))

        except Exception as e:
            print(f"Error while scraping reviews: {e}")

    # Function to extract product name from URL
    def extract_product_name(url):
        match = re.search(r"/([a-zA-Z0-9\-]+)-dp/[A-Z0-9]+", url)
        if match:
            return match.group(1).replace("-", " ").title()
        else:
```

```
        return "N/A"

# Iterate through all URLs and scrape review data for each product
for index, url in enumerate(df['Urls'].tolist()):
    print(f"Scraping URL {index + 1}/{len(df['Urls'])}: {url}")

    # Extract product name from the URL
    product_name = extract_product_name(url)

    try:
        # Extract ASIN from the URL
        asin = url.split("/dp/")[1].split("/")[0]
                        reviews_url = f"https://www.amazon.com/product-reviews/{asin}/
        ref=cm_cr_arp_d_paging_btm_next_1?
        ie=UTF8&reviewerType=all_reviews&pageNumber=1"
        driver.get(reviews_url)
        time.sleep(2)  # Allow the page to load

        # Extract reviews from the first page and subsequent pages
        while True:
            extract_reviews(asin, product_name)

            # Check if the "Next" button is present for pagination
            try:
                next_button = driver.find_element(By.XPATH, "//li[@class='a-last']/a")
                if "disabled" in next_button.get_attribute("class"):
                    break  # If the "Next" button is disabled, exit the loop
                next_button.click()  # Click the "Next" button
                time.sleep(2)  # Wait for the next page to load
            except Exception:
                # If "Next" button is not found or clickable, break out of the loop
                break

    except IndexError:
        print(f"Error extracting ASIN from URL: {url}")
        continue

# Create a DataFrame ensuring all lists are the same length
max_length = max(len(ASINs), len(Product_Names), len(Review_Titles),
        len(Review_Contents),
                                len(Review_Stars), len(Review_Dates), len(Reviewer_Names),
        len(Verified_Purchase),
            len(Helpful_Votes), len(Review_URLs))
```

```
# Trim lists to max length or fill with None
ASINs += [None] * (max_length - len(ASINs))
Product_Names += [None] * (max_length - len(Product_Names))
Review_Titles += [None] * (max_length - len(Review_Titles))
Review_Contents += [None] * (max_length - len(Review_Contents))
Review_Stars += [None] * (max_length - len(Review_Stars))
Review_Dates += [None] * (max_length - len(Review_Dates))
Reviewer_Names += [None] * (max_length - len(Reviewer_Names))
Verified_Purchase += [None] * (max_length - len(Verified_Purchase))
Helpful_Votes += [None] * (max_length - len(Helpful_Votes))
Review_URLs += [None] * (max_length - len(Review_URLs))

# Compile the scraped data into a dictionary
reviews_dict = {
    "ASIN": ASINs,
    "Product_Name": Product_Names,
    "Review_Title": Review_Titles,
    "Review_Content": Review_Contents,
    "Review_Stars": Review_Stars,
    "Review_Date": Review_Dates,
    "Reviewer_Name": Reviewer_Names,
    "Verified_Purchase": Verified_Purchase,
    "Helpful_Votes": Helpful_Votes,
    "Review_URL": Review_URLs
}

# Convert the dictionary into a DataFrame and save it to a CSV file
df_reviews = pd.DataFrame(reviews_dict)
df_reviews.to_csv("amazon_tv_reviews.csv", index=False)

# Print the first few rows of the scraped data
print(df_reviews.head())

# Close the WebDriver
driver.quit()
```

Product Scraping:

```
import requests
from bs4 import BeautifulSoup
import random
import time
```

```
import pandas as pd

# List of user agents to rotate
USER_AGENTS = [
    'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
        Chrome/91.0.4472.124 Safari/537.36',
    'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
        Chrome/89.0.4389.82 Safari/537.36',
    'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
        Chrome/88.0.4324.104 Safari/537.36',
    'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
        Chrome/90.0.4430.93 Safari/537.36',
]

# Function to get the HTML content of a product page
def get_product_page(asin):
    url = f"https://www.amazon.com/dp/{asin}"
    headers = {
        'User-Agent': random.choice(USER_AGENTS),
        'Accept-Language': 'en-US,en;q=0.9',
        'Accept-Encoding': 'gzip, deflate, br',
        'DNT': '1',  # Do Not Track Request Header
        'Connection': 'keep-alive',
    }

    try:
        response = requests.get(url, headers=headers)
        if response.status_code == 200:
            return response.text
        else:
            print(f"Failed to retrieve page for ASIN {asin}, Status code: {response.status_code}")
            return None
    except Exception as e:
        print(f"Error fetching page for ASIN {asin}: {e}")
        return None

# Function to parse product details from the HTML content
def parse_product_details(asin, html_content):
    soup = BeautifulSoup(html_content, 'html.parser')

    product_details = {
        "ASIN": asin,
        "Product Name": None,
```

```
    "Price": None,
    "Brand": None,
    "Rating": None,
    "Product Description": None,
    "Dimensions": None,
    "Weight": None
}

# Parse product name
product_name_tag = soup.find(id='productTitle')
if product_name_tag:
    product_details['Product Name'] = product_name_tag.get_text(strip=True)

# Parse price
        price_tag = soup.find('span', {'id': 'priceblock_ourprice'}) or soup.find('span', {'id':
       'priceblock_dealprice'})
if price_tag:
    product_details['Price'] = price_tag.get_text(strip=True)

# Parse brand
brand_tag = soup.find('a', {'id': 'bylineInfo'})
if brand_tag:
    product_details['Brand'] = brand_tag.get_text(strip=True)

# Parse rating
rating_tag = soup.find('span', {'class': 'a-icon-alt'})
if rating_tag:
    product_details['Rating'] = rating_tag.get_text(strip=True)

# Parse product description
description_tag = soup.find('div', {'id': 'feature-bullets'})
if description_tag:
    product_details['Product Description'] = description_tag.get_text(strip=True)

# Parse dimensions and weight
technical_details = soup.find('table', {'id': 'productDetails_techSpec_section_1'})
if technical_details:
    for row in technical_details.find_all('tr'):
        th = row.find('th').get_text(strip=True)
        td = row.find('td').get_text(strip=True)
        if 'Dimensions' in th:
            product_details['Dimensions'] = td
        if 'Weight' in th:
```

```
            product_details['Weight'] = td

    return product_details

# Function to scrape product details using ASIN
def scrape_product_details(asin):
    print(f"Scraping ASIN: {asin}")
    html_content = get_product_page(asin)
    if html_content:
        product_details = parse_product_details(asin, html_content)
        return product_details
    else:
        return None

# Read the ASINs from the laptop_reviews.csv file
df_reviews = pd.read_csv("amazon_tv_reviews.csv")

# List of ASINs from the csv file
asins = df_reviews['ASIN'].unique()  # Ensure only unique ASINs are used

# Scrape product details for each ASIN and store in a list
product_data = []
for asin in asins:
    details = scrape_product_details(asin)
    if details:
        product_data.append(details)
    # Sleep to avoid being blocked
    time.sleep(random.uniform(1, 3))

# Convert the list of product details into a DataFrame
df_products = pd.DataFrame(product_data)

# Merge product details with the existing reviews
df_combined = pd.merge(df_reviews, df_products, on='ASIN', how='left')

# Save the combined data to a new CSV file
df_combined.to_csv("combined_TV_reviews_details.csv", index=False)

# Print the combined DataFrame
print(df_combined)
```

Model Training (Roberta) :

```
from transformers import AutoTokenizer, AutoModelForSequenceClassification
import torch

# Load tokenizer and model for RoBERTa sentiment analysis
tokenizer = AutoTokenizer.from_pretrained("cardiffnlp/twitter-roberta-base-sentiment-latest")
model = AutoModelForSequenceClassification.from_pretrained("cardiffnlp/twitter-roberta-base-
        sentiment-latest")

# Set device to GPU if available
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)

# Function to perform sentiment classification on a batch of reviews
def classify_sentiment_roberta(reviews):
    # Tokenize with truncation and padding
            inputs = tokenizer(reviews, padding=True, truncation=True, return_tensors="pt",
        max_length=512).to(device)

    # Pass the inputs through the model
    with torch.no_grad():
        outputs = model(**inputs)

    # Get predictions
    logits = outputs.logits
    predictions = torch.argmax(logits, dim=-1)
    return predictions.cpu().numpy()

# Create a DataLoader to batch the reviews (batch size 32)
batch_size = 32
review_batches = DataLoader(df_english['Cleaned_Review_Content'].tolist(),
        batch_size=batch_size)

# Initialize an empty list to store predictions
predictions = []

# Perform sentiment analysis in batches with progress tracking
for batch in tqdm(review_batches, desc="Classifying sentiments with RoBERTa"):
    batch_predictions = classify_sentiment_roberta(batch)
    predictions.extend(batch_predictions)

# Add predictions to the DataFrame
df_english['Predicted_Sentiment_RoBERTa'] = predictions
```

```
# Convert predicted sentiment (0 for negative, 1 for neutral, 2 for positive)
df_english['Predicted_Sentiment_Label_RoBERTa'] =
        df_english['Predicted_Sentiment_RoBERTa'].apply(
    lambda x: 1 if x == 2 else (0 if x == 0 else 2))  # Map positive to 1, negative to 0, and neutral
        to 2

# Save the DataFrame with predictions (optional)
df_english.to_csv("cleaned_reviews_roberta_mapped.csv", index=False)

# Display the first few rows of the DataFrame with predictions
print(df_english[['Review_Content', 'Predicted_Sentiment_Label_RoBERTa']].head(10))
```

LLM (Llama 3.1 8b) :

```
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM
from sklearn.metrics import accuracy_score, classification_report
from tqdm import tqdm
import pandas as pd


texts = data['Cleaned_Review'].tolist()
true_labels = data['True_Label'].tolist()  # Assuming 1=positive, 0=negative

# Load LLaMA model and tokenizer
model_name = "meta-llama/Llama-3.1-8B"  # Replace with actual model path if needed
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(model_name, torch_dtype=torch.float16 if
torch.cuda.is_available() else torch.float32)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)

# Predict function
def get_sentiment(text):
    inputs = tokenizer(text, return_tensors="pt").to(device)
    outputs = model.generate(**inputs, max_new_tokens=20)
    generated_text = tokenizer.decode(outputs[0], skip_special_tokens=True)
    return 1 if "positive" in generated_text.lower() else 0  # Adjust based on actual output format

# Apply model and calculate accuracy
predictions = [get_sentiment(text) for text in tqdm(texts)]
accuracy = accuracy_score(true_labels, predictions)
report = classification_report(true_labels, predictions)

print(f"Accuracy: {accuracy * 100:.2f}%")
print(report)
```

Llama Prompting (Zero Shot):

```
# Improved prompt template
prompt_template = (
    "You are an expert customer review classifier. Your job is to analyze Amazon product reviews and classify "
    "the sentiment of each review based on the customer's tone and experience. For each review: \n"
    "- Respond with '2' if the review is positive, indicating high satisfaction or praise.\n"
    "- Respond with '1' if the review is neutral, indicating a balanced or mixed sentiment.\n"
    "- Respond with '0' if the review is negative, indicating dissatisfaction or complaints.\n\n"
    "Review: {text}\n\nSentiment:"
)

# Prediction function with enhanced prompt engineering
def get_sentiment(text):
    prompt = prompt_template.format(text=text)
    inputs = tokenizer(prompt, return_tensors="pt").to(device)
    outputs = model.generate(**inputs, max_new_tokens=10)
    generated_text = tokenizer.decode(outputs[0], skip_special_tokens=True).strip()

    # Parse generated output
    if "2" in generated_text:
        return 2
    elif "1" in generated_text:
        return 1
    else:
        return 0

# Apply the model and calculate accuracy
predictions = [get_sentiment(text) for text in tqdm(texts)]
accuracy = accuracy_score(true_labels, predictions)
report = classification_report(true_labels, predictions)

print(f"Accuracy: {accuracy * 100:.2f}%")
print(report)
```

Mistral Fine-Tuning (using Peft) :

```
# Select the desired columns with dictionary comprehension
dataset = df[['Label', 'Text']].rename(columns={'Label': 'labels', 'Text': 'text'})
```

```
# Split the dataset into train and test
from sklearn.model_selection import train_test_split

train_data, test_data = train_test_split(dataset, test_size=0.2, random_state=42)

# Initialize tokenizer and model with quantization
bnb_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_compute_dtype=torch.float16,
    bnb_4bit_use_double_quant=False
)

model = AutoModelForCausalLM.from_pretrained(
    "mistralai/Mistral-7B-Instruct-v0.2",
    quantization_config=bnb_config
)

tokenizer = AutoTokenizer.from_pretrained("mistralai/Mistral-7B-Instruct-v0.2")
model.config.pad_token_id = tokenizer.pad_token_id  # Set padding token

# Disable caching
model.config.use_cache = False

# Prepare the data
def tokenize_function(examples):
    return tokenizer(examples['text'], padding=True, truncation=True, max_length=512)


import pandas as pd
from transformers import AutoModelForCausalLM, AutoTokenizer, BitsAndBytesConfig, Trainer, TrainingArguments, DataCollatorForLanguageModeling
import torch
from sklearn.model_selection import train_test_split

# Load your dataset (replace with your actual data loading)
df = pd.read_csv("your_dataset.csv")

# Prepare the dataset
dataset = df[['Label', 'Text']].rename(columns={'Label': 'labels', 'Text': 'text'})
train_data, test_data = train_test_split(dataset, test_size=0.2, random_state=42)
```

```
data_collator = DataCollatorForLanguageModeling(tokenizer=tokenizer, mlm=False)

def tokenize_function(examples):
    return tokenizer(examples['text'], padding='max_length', truncation=True, max_length=512)

tokenized_train_data = train_data.map(tokenize_function, batched=True)
tokenized_test_data = test_data.map(tokenize_function, batched=True)

# Quantization Configuration
bnb_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_compute_dtype=torch.float16,
    bnb_4bit_use_double_quant=False
)



# PEFT Configuration
lora_config = LoraConfig(
    r=8,
    lora_alpha=16,
    lora_dropout=0.1,
    target_modules=["q_proj", "k_proj", "v_proj"]
)

# Apply PEFT to the model
model = get_peft_model(model, lora_config)

# Training Arguments
training_args = TrainingArguments(
    output_dir="./results",
    num_train_epochs=3,
    per_device_train_batch_size=4,
    per_device_eval_batch_size=8,
    evaluation_strategy="epoch",
    save_strategy="epoch",
    logging_dir="./logs",
    logging_steps=10,
    save_steps=500,
    load_best_model_at_end=True,
    metric_for_best_model="eval_loss",
)
```

```
# Trainer
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_train_data,
    eval_dataset=tokenized_test_data,
    data_collator=data_collator,
    tokenizer=tokenizer
)

# Train the model
trainer.train()

# Evaluate the model
eval_results = trainer.evaluate()
print(eval_results)
```

Outputs:

Scraping:

```
Scraping page 1...
https://www.amazon.com/VIZIO-40-inch-1080p-Virtual-Built/dp/B0CXG3HMX1/ref=sr_1_1?crid=34BZT2YZDL89K&dib=eyJ2IjoiMSJ9.byxHhZj_hF9e7h0GLUC8XZeTmrkaohjn2E_48KkvufqLpBM0iBS-2SbsZKTI2jCHgO5x7M0L-x1Vzd2BENrkWJ37Odj1eoC3GSfjRKrnb9w1LzQNJFFxeMjkE56yfLJx7RJcuPawLyevxhgT24GWIG1L-RLJUa98we9Vgy4DOidHO1kv0P2P4AhjMytU0tX0HQGsyFrzVuwpqFKEs_fmLkeiVJzMw0wBlE1DUPPWxcg.XSq2k2yfLYkXMLXODcf8dSl-eps3cpTiUpKDBf99iK8&dib_tag=se&keywords=tv&qid=1727402869&sprefix=tv%2Caps%2C118&sr=8-1
https://www.amazon.com/INSIGNIA-All-New-50-inch-Class-NS-50F301NA24/dp/B0BTTVRWPR/ref=sr_1_2?crid=34BZT2YZDL89K&dib=eyJ2IjoiMSJ9.byxHhZj_hF9e7h0GLUC8XZeTmrkaohjn2E_48KkvufqLpBM0iBS-2SbsZKTI2jCHgO5x7M0L-x1Vzd2BENrkWJ37Odj1eoC3GSfjRKrnb9w1LzQNJFFxeMjkE56yfLJx7RJcuPawLyevxhgT24GWIG1L-RLJUa98we9Vgy4DOidHO1kv0P2P4AhjMytU0tX0HQGsyFrzVuwpqFKEs_fmLkeiVJzMw0wBlE1DUPPWxcg.XSq2k2yfLYkXMLXODcf8dSl-eps3cpTiUpKDBf99iK8&dib_tag=se&keywords=tv&qid=1727402869&sprefix=tv%2Caps%2C118&sr=8-2
https://www.amazon.com/INSIGNIA-32-inch-Class-Smart-NS-32F202NA23/dp/B0BCMRRKRX/ref=sr_1_3?crid=34BZT2YZDL89K&dib=eyJ2IjoiMSJ9.byxHhZj_hF9e7h0GLUC8XZeTmrkaohjn2E_48KkvufqLpBM0iBS-2SbsZKTI2jCHgO5x7M0L-x1Vzd2BENrkWJ37Odj1eoC3GSfjRKrnb9w1LzQNJFFxeMjkE56yfLJx7RJcuPawLyevxhgT24GWIG1L-RLJUa98we9Vgy4DOidHO1kv0P2P4AhjMytU0tX0HQGsyFrzVuwpqFKEs_fmLkeiVJzMw0wBlE1DUPPWxcg.XSq2k2yfLYkXMLXODcf8dSl-eps3cpTiUpKDBf99iK8&dib_tag=se&keywords=tv&qid=1727402869&sprefix=tv%2Caps%2C118&sr=8-3
https://www.amazon.com/All-New-Insignia-32-inch-Class-NS-32F201NA23/dp/B09ZLTMWWH/ref=sr_1_4?crid=34BZT2YZDL89K&dib=eyJ2IjoiMSJ9.byxHhZj_hF9e7h0GLUC8XZeTmrkaohjn2E_48KkvufqLpBM0iBS-2SbsZKTI2jCHgO5x7M0L-x1Vzd2BENrkWJ37Odj1eoC3GSfjRKrnb9w1LzQNJFFxeMjkE56yfLJx7RJcuPawLyevxhgT24GWIG1L-RLJUa98we9Vgy4DOidHO1kv0P2P4AhjMytU0tX0HQGsyFrzVuwpqFKEs_fmLkeiVJzMw0wBlE1DUPPWxcg.XSq2k2yfLYkXMLXODcf8dSl-eps3cpTiUpKDBf99iK8&dib_tag=se&keywords=tv&qid=1727402869&sprefix=tv%2Caps%2C118&sr=8-4
https://www.amazon.com/amazon-fire-tv-32-inch-2-series-hd-smart-tv/dp/B0CJDSNN4T/ref=sr_1_5?crid=34BZT2YZDL89K&dib=eyJ2IjoiMSJ9.byxHhZj_hF9e7h0GLUC8XZeTmrkaohjn2E_48KkvufqLpBM0iBS-2SbsZKTI2jCHgO5x7M0L-x1Vzd2BENrkWJ37Odj1eoC3GSfjRKrnb9w1LzQNJFFx
```

```
Scraping ASIN: B0CMDJ844V
Scraping ASIN: B0CMZ8ZBVN
Scraping ASIN: B0CF2PV74C
Scraping ASIN: B0C544TBQM
Scraping ASIN: B08PNB9B5Z
Scraping ASIN: B07Z3XZDT5
Scraping ASIN: B077RM9WJB
Scraping ASIN: B0C2W7YYHM
Scraping ASIN: B0BV5SV5HL
Scraping ASIN: B08G5B4PV5
Scraping ASIN: B08CFSZLQ4
Scraping ASIN: B08KRKFHGV
Scraping ASIN: B09G2BN89Q
Scraping ASIN: B0BYHC3ZMS
Scraping ASIN: B08HVXC89J
Scraping ASIN: B083B8HH98
Scraping ASIN: B0BLW47H3M
Scraping ASIN: B00R2K4LCY
Scraping ASIN: B08NWBY8YJ
Scraping ASIN: B0BYLRBG4M
```

```
Scraping URL 1/314: https://www.amazon.com/VIZIO-40-inch-1080p-Virtual-Built/dp/B0CXG3HMX1/ref=sr_1_1?crid=34BZT2YZDL89K&dib=eyJ2IjoiMSJ9.byxHhZj_hF9e7h0GLUC8XZeTmrkaohjn2E_48KkvufqLpBM0iBS-2SbsZKTI2jCHgO5x7M0L-x1Vzd2BENrkWJ37Odj1eoC3GSfjRKrnb9w1LzQNJFFxeMjkE56yfLJx7RJcuPawLyevxhgT24GWIG1L-RLJUa98we9Vgy4DOidHO1kv0P2P4AhjMytU0tX0HQGsyFrzVuwpqFKEs_fmLkeiVJzMw0wBlE1DUPPWxcg.XSq2k2yfLYkXMLXODcf8dSl-eps3cpTiUpKDBf99iK8&dib_tag=se&keywords=tv&qid=1727402869&sprefix=tv%2Caps%2C118&sr=8-1
Scraping URL 2/314: https://www.amazon.com/INSIGNIA-All-New-50-inch-Class-NS-50F301NA24/dp/B0BTTVRWPR/ref=sr_1_2?crid=34BZT2YZDL89K&dib=eyJ2IjoiMSJ9.byxHhZj_hF9e7h0GLUC8XZeTmrkaohjn2E_48KkvufqLpBM0iBS-2SbsZKTI2jCHgO5x7M0L-x1Vzd2BENrkWJ37Odj1eoC3GSfjRKrnb9w1LzQNJFFxeMjkE56yfLJx7RJcuPawLyevxhgT24GWIG1L-RLJUa98we9Vgy4DOidHO1kv0P2P4AhjMytU0tX0HQGsyFrzVuwpqFKEs_fmLkeiVJzMw0wBlE1DUPPWxcg.XSq2k2yfLYkXMLXODcf8dSl-eps3cpTiUpKDBf99iK8&dib_tag=se&keywords=tv&qid=1727402869&sprefix=tv%2Caps%2C118&sr=8-2
Scraping URL 3/314: https://www.amazon.com/INSIGNIA-32-inch-Class-Smart-NS-32F202NA23/dp/B0BCMRRKRX/ref=sr_1_3?crid=34BZT2YZDL89K&dib=eyJ2IjoiMSJ9.byxHhZj_hF9e7h0GLUC8XZeTmrkaohjn2E_48KkvufqLpBM0iBS-2SbsZKTI2jCHgO5x7M0L-x1Vzd2BENrkWJ37Odj1eoC3GSfjRKrnb9w1LzQNJFFxeMjkE56yfLJx7RJcuPawLyevxhgT24GWIG1L-RLJUa98we9Vgy4DOidHO1kv0P2P4AhjMytU0tX0HQGsyFrzVuwpqFKEs_fmLkeiVJzMw0wBlE1DUPPWxcg.XSq2k2yfLYkXMLXODcf8dSl-eps3cpTiUpKDBf99iK8&dib_tag=se&keywords=tv&qid=1727402869&sprefix=tv%2Caps%2C118&sr=8-3
Scraping URL 4/314: https://www.amazon.com/All-New-Insignia-32-inch-Class-NS-32F201NA23/dp/B09ZLTMWWH/ref=sr_1_4?crid=34BZT2YZDL89K&dib=eyJ2IjoiMSJ9.byxHhZj_hF9e7h0GLUC8XZeTmrkaohjn2E_48KkvufqLpBM0iBS-2SbsZKTI2jCHgO5x7M0L-x1Vzd2BENrkWJ37Odj1eoC3GSfjRKrnb9w1LzQNJFFxeMjkE56yfLJx7RJcuPawLyevxhgT24GWIG1L-RLJUa98we9Vgy4DOidHO1kv0P2P4AhjMytU0tX0HQGsyFrzVuwpqFKEs_fmLkeiVJzMw0wBlE1DUPPWxcg.XSq2k2yfLYkXMLXODcf8dSl-eps3cpTiUpKDBf99iK8&dib_tag=se&keywords=tv&qid=1727402869&sprefix=tv%2Caps%2C118&sr=8-4
```

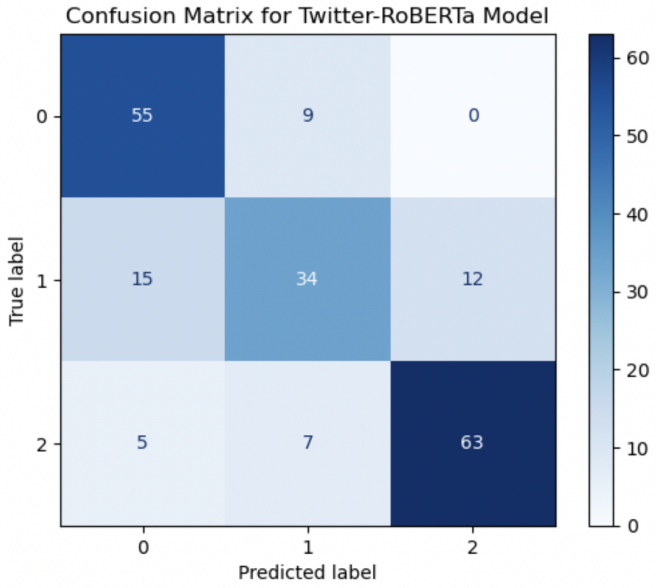
Model Training:

Roberta:

Classification Report for Twitter-RoBERTa Model:

	precision	recall	f1-score	support
0	0.73	0.86	0.79	64
1	0.68	0.56	0.61	61
2	0.84	0.84	0.84	75
accuracy			0.76	200
macro avg	0.75	0.75	0.75	200
weighted avg	0.76	0.76	0.76	200

<Figure size 800x600 with 0 Axes>



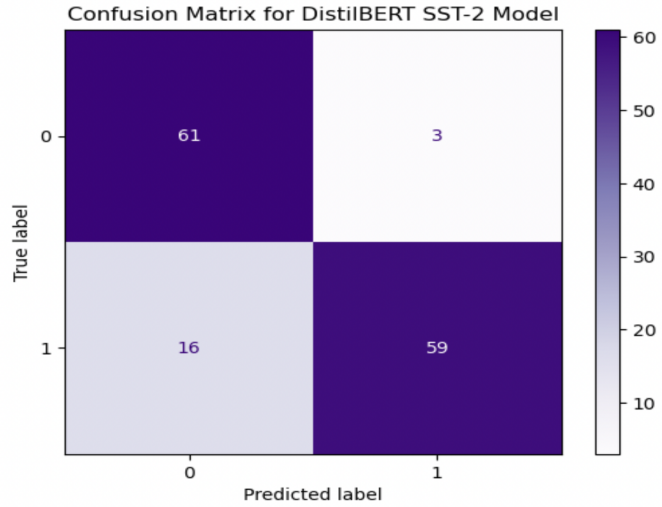
DistillBERT:

```
Classification Report for DistilBERT SST-2 Model:
              precision    recall  f1-score   support

           0       0.79      0.95      0.87        64
           1       0.95      0.79      0.86        75

    accuracy                           0.86       139
   macro avg       0.87      0.87      0.86       139
weighted avg       0.88      0.86      0.86       139

<Figure size 800x600 with 0 Axes>
```



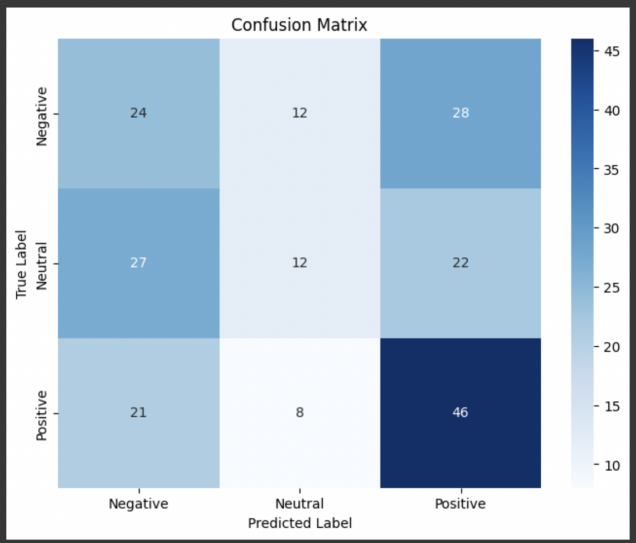
Mistral 7b instruct :

```
Accuracy of Mistral-7B-Instruct-v0.2 for Sentiment Analysis: 41.00%

Classification Report:
              precision    recall  f1-score   support

    Negative       0.33      0.38      0.35        64
     Neutral       0.38      0.20      0.26        61
    Positive       0.48      0.61      0.54        75

    accuracy                           0.41       200
   macro avg       0.40      0.40      0.38       200
weighted avg       0.40      0.41      0.39       200
```



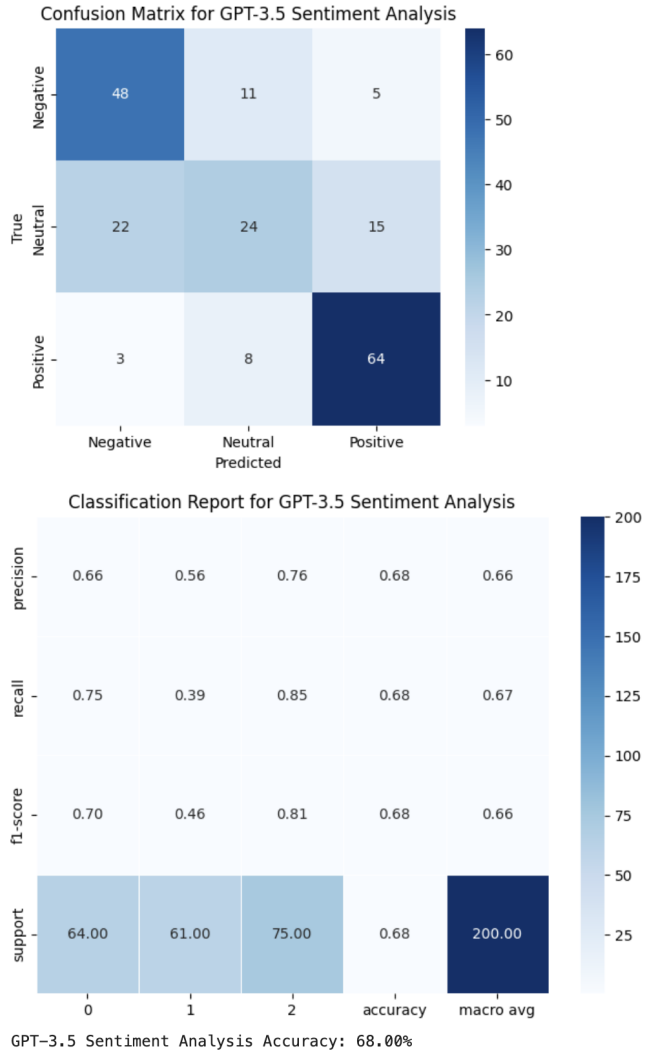
llama-3.1-8b:

```
|          | 200/200 [29:45<00:00,  8.93s/it]Accuracy: 31.50%
              precision    recall  f1-score   support

    Negative       0.32      0.98      0.48        64
     Neutral       0.00      0.00      0.00        61
    Positive       0.00      0.00      0.00        75

    accuracy                           0.32       200
    acro avg       0.11      0.33      0.16       200
    hted avg       0.10      0.32      0.15       200
```

GPT 3.5:



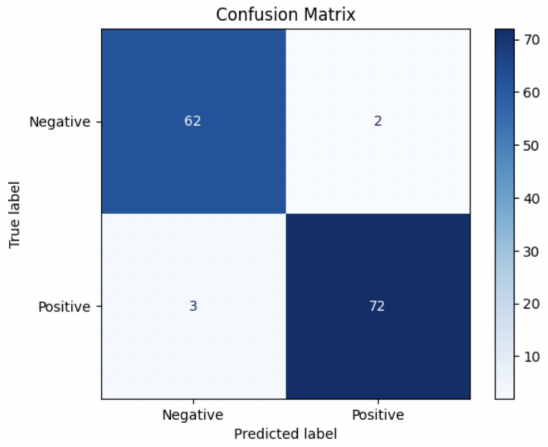
GPT-3.5 Sentiment Analysis Accuracy: 68.00%

Fine-tuned Roberta:

```
Model Accuracy of Fine-Tuned Roberta Model (Binary Classification): 96.40%
Classification Report :
              precision    recall  f1-score   support

    Negative       0.95      0.97      0.96        64
    Positive       0.97      0.96      0.97        75

    accuracy                           0.96       139
   macro avg       0.96      0.96      0.96       139
weighted avg       0.96      0.96      0.96       139
```



Fine-tuned LLama 3:

model_name = "../input/llama-3/transformers/8b-chat-hf/1"

```
In [19]: y_pred = predict(test, model, tokenizer)
         evaluate(y_true, y_pred)

  0%|          | 0/900 [00:00<?, ?it/s]`use_cache=True` is incompatible with gradient checkpoi
nting. Setting `use_cache=False`.
100%|██████████| 900/900 [06:05<00:00,  2.47it/s]

Accuracy: 0.873
Accuracy for label 0: 0.937
Accuracy for label 1: 0.847
Accuracy for label 2: 0.837

Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.94      0.94       300
           1       0.80      0.85      0.82       300
           2       0.87      0.84      0.86       300

    accuracy                           0.87       900
   macro avg       0.88      0.87      0.87       900
weighted avg       0.88      0.87      0.87       900


Confusion Matrix:
[[281  18   1]
 [ 11 254  35]
 [  3  46 251]]
```

Codes for RAG Based Dashboard

1. To get sentiments using Mistral API

```
!pip install transformers torch
from transformers import pipeline
import pandas as pd # Load data
data = pd.read_csv('/content/Mistral_random_200_reviews_with_true_label_PD.csv')
# Display the first few rows
data.head()

!pip install -qU mistralai
api=" (fill with the API key)
import os
from mistralai import Mistral

api_key = api
model = "mistral-large-latest"

client = Mistral(api_key=api_key)


# Function to get sentiment from Mistral
def get_mistral_sentiment(review_text):
   chat_response = client.chat.complete(
      model=model,
      messages=[
         {
            "role": "user",
                "content": f"Simply answer whether the following text as in 1 word Positive, Negative, or
Neutral: '{review_text}'",
         },
      ]
   )
   # Extract and return the response text
   return chat_response.choices[0].message.content.strip()

# Apply sentiment analysis to each review in the DataFrame
data_1['Mistral_Sentiment'] = data_1['Cleaned_Review'].apply(get_mistral_sentiment)

# Display the DataFrame with the new Mistral_Sentiment column
print(data_1.head())
```

2. Function to get feature sentiment from Mistral

```
# Function to get feature sentiment from Mistral
def get_mistral_feature_sentiment(review_text):
   chat_response = client.chat.complete(
      model=model,
      messages=[
         {
            "role": "user",
```

```
                "content": f"Identify key features from the review below and categorize each as Positive or Negative. Positive and Negative should be in Only Biagrams Format. Output should be in the format: Positive Features: [feature1, feature2, ...]; Negative Features: [feature1, feature2, ...]. Review text: '{review_text}'",
            },
        ]
    )
    # Extract and parse the response text
    response_text = chat_response.choices[0].message.content.strip()

    # Parse the response to separate positive and negative features
    positive_features = []
    negative_features = []

    # Split the response by labels
    if "Positive Features:" in response_text:
        positive_part = response_text.split("Positive Features:")[1].split("Negative Features:")[0].strip()
        positive_features = [feature.strip() for feature in positive_part.split(',') if feature]

    if "Negative Features:" in response_text:
        negative_part = response_text.split("Negative Features:")[1].strip()
        negative_features = [feature.strip() for feature in negative_part.split(',') if feature]

    return positive_features, negative_features

# Apply the function to each review and expand columns for positive and negative features
data_1[['Positive_Features', 'Negative_Features']] = data_1['Cleaned_Review'].apply(lambda x: pd.Series(get_mistral_feature_sentiment(x)))

# Display the DataFrame with new columns
print(data_1[['Cleaned_Review', 'Positive_Features', 'Negative_Features']].head())
```

3. To generate Dash dashboard

```
import spacy
import re
import pandas as pd
from textblob import TextBlob
from spacy.lang.en.stop_words import STOP_WORDS

# Load spaCy English model
nlp = spacy.load('en_core_web_sm')

# Define stopwords, adding custom terms
custom_stopwords = {'kind', 'work', 'quality', 'feature', 'thats', 'clear', 'quickly', 'samsung', 'full',
                    'phone', 'fine', 'nice', 'also', 'easy', 'perfect', 'super', 'amazing', 'good', 'best',
                    'exceptional', 'great', 'old', 'new', 'really', 'much', 'better', 'love', 'happy', 'worth',
                    'far', 'awesome', 'excellent'}
```

```
stop_words = STOP_WORDS.union(custom_stopwords)

# Step 1: Clean, preprocess, and lemmatize text
def clean_text(text):
    text = re.sub(r'[^a-zA-Z\s]', '', text.lower())  # Remove punctuation and non-alpha chars
    doc = nlp(text)  # Process text with spaCy
    tokens = [token.lemma_ for token in doc if token.lemma_ not in stop_words and not token.is_stop]  # Lemmatize and remove stopwords
    return tokens

# Step 2: Extract features using bigrams with only nouns
def extract_features_based_on_sentiment(review_text):
    tokens = clean_text(review_text)  # Clean and tokenize review text
    bigrams_list = [' '.join([tokens[i], tokens[i+1]]) for i in range(len(tokens) - 1)]  # Generate bigrams

    # Initialize lists for positive and negative features
    positive_features = []
    negative_features = []

    # Analyze each bigram for sentiment and classify as positive or negative feature
    for bigram in bigrams_list:
        blob = TextBlob(bigram)
        sentiment = blob.sentiment.polarity  # Get sentiment polarity

        # Process bigram with spaCy for POS tagging and filter nouns
        bigram_doc = nlp(bigram)
        noun_words = [token.text for token in bigram_doc if token.pos_ == 'NOUN']  # Extract nouns

        # Only process if we have nouns in the bigram
        if noun_words:
            feature_word = noun_words[-1]  # Use the last noun as the feature

            # Classify the feature based on the sentiment polarity
            if sentiment > 0:
                positive_features.append(feature_word)
            elif sentiment < 0:
                negative_features.append(feature_word)

    # Remove duplicates and join lists into comma-separated strings
    all_positive_features = ', '.join(set([feature for feature in positive_features if feature]))
    all_negative_features = ', '.join(set([feature for feature in negative_features if feature]))

    return pd.Series([all_positive_features, all_negative_features])

# Step 3: Apply feature extraction function to each review
filtered_data[['All_Positive_Features', 'All_Negative_Features']] = filtered_data['Review_Content'].apply(extract_features_based_on_sentiment)

# View the resulting DataFrame
print(filtered_data.head())
import pandas as pd
```

```
import plotly.express as px
from dash import Dash, html, dcc, Input, Output
from difflib import get_close_matches

# Function to get positive and negative features for the closest matching product
def get_product_features(product_name):
    closest_match = get_close_matches(product_name, data['Product Name'], n=1)
    if closest_match:
        product_data = data[data['Product Name'] == closest_match[0]]
                        positive_features = product_data['All_Positive_Features'].str.split(', ').explode().value_counts().reset_index()
        positive_features.columns = ['Feature', 'Count']
          positive_features = positive_features[positive_features['Feature'] != ''] # Remove empty features
           positive_features = positive_features.head(5) # Keep only the top 10 positive features

                        negative_features = product_data['All_Negative_Features'].str.split(', ').explode().value_counts().reset_index()
        negative_features.columns = ['Feature', 'Count']
         negative_features = negative_features[negative_features['Feature'] != ''] # Remove empty features
           negative_features = negative_features.head(5) # Keep only the top 10 negative features

        return positive_features, negative_features
    else:
                          return pd.DataFrame(columns=['Feature', 'Count']), pd.DataFrame(columns=['Feature', 'Count'])

# Initialize the Dash app
app = Dash(__name__)

# Layout for the dashboard
app.layout = html.Div([
           html.H1("Customer Review Feature Analysis Dashboard", style={'text-align': 'center'}),

    # Input for Product Name
    dcc.Input(
        id='product_input',
        type='text',
        placeholder='Enter product name',
              style={'width': '50%', 'margin': 'auto', 'margin-top': '20px', 'margin-bottom': '20px'}
    ),

    # Dropdown for Sentiment Selection
    dcc.Dropdown(
        id='sentiment_dropdown',
        options=[
            {'label': 'Positive', 'value': 'Positive'},
            {'label': 'Negative', 'value': 'Negative'}
        ],
```

```
        value='Positive',
        clearable=False,
        style={'width': '50%', 'margin': 'auto'}
    ),

    # Graph for Feature Counts
    dcc.Graph(id='feature_graph'),

    # Pie chart for Feature Distribution
    dcc.Graph(id='feature_pie')
])

# Callback to update the graph based on product input and sentiment
@app.callback(
    [Output('feature_graph', 'figure'), Output('feature_pie', 'figure')],
    [Input('product_input', 'value'), Input('sentiment_dropdown', 'value')]
)
def update_graph(product_name, selected_sentiment):
    if not product_name:
        # Return empty figures if no product name is provided
        return {}, {}

    # Get features for the closest matching product
    positive_features, negative_features = get_product_features(product_name)

    # Choose data based on selected sentiment
    if selected_sentiment == 'Positive':
        feature_data = positive_features
        title = f'Top 5 Positive Features for {product_name}'
    else:
        feature_data = negative_features
        title = f'Top 5 Negative Features for {product_name}'

    # Bar chart for feature counts
    fig_bar = px.bar(
        feature_data,
        x='Feature',
        y='Count',
        title=title,
        color='Count',
        labels={'Count': 'Feature Count'},
        template='plotly_dark'
    )

    # Pie chart for feature distribution
    fig_pie = px.pie(
        feature_data,
        names='Feature',
        values='Count',
        title=f'{title} Distribution',
        template='plotly_dark'
    )

    return fig_bar, fig_pie
```

```
if __name__ == '__main__':
    app.run_server(debug=True)
```