### **Project Description**

This research project investigates customer sentiment towards iPhone products using a comprehensive dataset collected across multiple continents, including APAC, EMEA, and the United States. The project applies machine learning models, traditional classification methods, and modern large language models (LLMs) to assess and predict customer sentiment. Clustering techniques are also employed to uncover geographic preferences. This research aims to identify key factors that influence customer satisfaction and evaluate the effectiveness of traditional machine learning models versus LLMs in sentiment prediction.

### **Keywords**

iPhone, Sentiment Analysis, Machine Learning, Clustering, NLP, LLM, DistilBERT, Logistic Regression, Random Forest, K-Means, Hypothesis Testing

## **1. Introduction**

### **1.1 Problem Statement**

Customer sentiment analysis is crucial for companies like Apple to understand public perception and improve product offerings. This research aims to analyze sentiment towards iPhone products by studying the impact of various product attributes, such as storage size, color, and region. The primary objectives of this research are:

* **Sentiment Variation Across Geographic Regions**: Identify regional differences in iPhone sentiment using clustering analysis.
* **Key Influencing Factors**: Determine attributes that significantly affect customer satisfaction.
* **Model Efficacy Comparison**: Compare traditional machine learning models and LLMs for sentiment prediction.
* **Temporal Trends Analysis**: Investigate how sentiments evolve over product release cycles.

## **2. Literature Review**

Customer sentiment analysis has been an important topic of research for understanding user perception. Pre-trained language models like **BERT** (Devlin et al., 2018) and **DistilBERT** have shown significant advancements in understanding textual sentiment. Earlier works by **Mikolov et al. (2013)** demonstrated the utility of word embeddings in capturing sentiment nuances. Recent research also includes exploring geographic-specific sentiment differences using clustering (**Lloyd, 1982**), adding a new dimension to understanding market preferences.

## **3. Methodology**

### **3.1 Dataset Overview**

The dataset used in this study comprises **3,062 reviews** with **11 features**, including product variant, country, rating score, and review text. Reviews span multiple geographic regions and are dated from **2021 to Fall 2024**.

|  |  |
| --- | --- |
| **Feature Name** | **Description** |
| productAsin | Product identifier for the iPhone variant |
| country | Country where the review was written |
| date | Date of the review |
| isVerified | Indicates whether the review is verified |
| ratingScore | Rating given by the reviewer (integer values) |
| reviewTitle | Title of the review |
| reviewDescription | Detailed description of the review |
| reviewedIn | Contextual information about the review |
| variant | Information about color, size, and provider |

### **3.2 Data Preprocessing**

* **Data Cleaning**: Missing values were handled, and duplicates were removed. Relevant features such as **size, color**, and **service provider** were extracted from the variant column.
* **Feature Engineering**: Temporal features were extracted from the date and RatingScore column to enable sentiment tracking over time.

### **3.3 Exploratory Data Analysis (EDA)**

EDA was performed using various statistical methods and visualizations:

* **Sentiment Analysis by Country**: **Histograms** and **clustering analysis** were used to identify geographic preferences.
* **Hypothesis Testing**: Statistical testing was used to validate the significance of product attributes influencing sentiment.

### **3.4 Clustering Analysis**

Unsupervised clustering was used to identify geographic regions with similar sentiment patterns. **K-Means clustering** (Lloyd, 1982) was used to group countries into clusters.

* **K-Means Algorithm**: The algorithm is an unsupervised learning technique that partitions n observations into k clusters, initializing centroids and iterating until convergence.
* **Distance Metric**: **Euclidean distance** was used to determine similarity between data points.
* **Cluster Assignments**: Clustering revealed major groups, with insights suggesting differences in satisfaction across regions.

### **3.5 Classification Models**

Three traditional machine learning models were employed for sentiment classification:

* **Logistic Regression**
* **Random Forest**
* **AdaBoost**

A comparison was conducted against the fine-tuned **DistilBERT** model for sentiment classification.

## **4. Algorithms Used**

### **4.1 Clustering**

* **K-Means**: Used to identify geographic differences in customer sentiment. The number of clusters (k) was chosen based on the **elbow method**, which indicated that three clusters were optimal.

### **4.2 Classification**

* **Logistic Regression**: A linear model that predicts sentiment as either positive or negative using probability scores derived from the sigmoid function.
* **Random Forest**: An ensemble learning method combining multiple decision trees, effective in capturing non-linear relationships.
* **AdaBoost**: A boosting algorithm that combines weak learners to iteratively improve accuracy.

### **4.3 Natural Language Processing (NLP)**

* **TF-IDF Vectorization**: Text data was converted into numerical vectors using **TF-IDF**, capturing the significance of each word.
* **DistilBERT Fine-tuning**: The **DistilBERT** model used transformer attention mechanisms to capture complex relationships within sentences.

## **5. Results**

#### **1. Understanding Sentiment Variation Across Countries or Continents**

* **Clustering Analysis**:
  + Countries were grouped into three clusters based on sentiment-related attributes.
    - **Cluster 0**: Japan, UAE, United States
    - **Cluster 1**: India, Japan, UAE
    - **Cluster 2**: UAE, Egypt, US, Mexico, Canada
  + Key findings from clustering:
    - **Cluster 1**: Common focus on product attributes like "iphone," "camera," and "battery."
    - **Cluster 0**: More concerns about product conditions like "scratches" and "battery."
    - **Cluster 2**: Similar issues as Cluster 0 but with more emphasis on "screen" and "condition."
* **Country-Specific Sentiments**:
  + **India & UAE**: Average rating of 3.81 and 3.91, showing balanced feedback with high engagement.
  + **Canada & Egypt**: Sparse data but high average ratings (4.5 and 5.0 respectively).
  + **Japan**: Lower average rating of 3.31, indicating lower satisfaction levels.
* **Insights**:
  + Regional clusters highlight diverse customer expectations.
  + High engagement and satisfaction in India and the US, with quality perception varying across regions.

#### **2. Key Factors Influencing Ratings**

* **Analysis of Product Features**:
  + Storage capacity:
    - 128GB variants dominate usage, indicating economic and accessibility factors.
    - Higher capacities (e.g., 512GB) show limited engagement.
  + Color preferences:
    - "Blue" and "Midnight" are most popular, but data inconsistencies in color naming require cleaning.
  + Product condition:
    - "Scratches" and "screen" were frequent words in reviews, hinting at common customer concerns.
* **Hypothesis Testing**:
  + Statistical tests (e.g., ANOVA or t-tests) can validate the influence of country, product variant, and verification status on sentiment.
  + Observed trends, such as differences in rating averages across countries and storage capacities, suggest significant variance in preferences.

#### **3. Comparison of Traditional Machine Learning Models with LLMs**

* **Performance Metrics**:
  + **Random Forest**:
    - Accuracy: **99.67%**, Precision: **99.67%**, Recall: **99.67%**, F1: **99.67%**
    - Best-performing model, showing strong interpretability and effectiveness for structured data.
  + **DistilBERT**:
    - Accuracy: **90.7%**, Precision: **90.78%**, Recall: **90.7%**, F1: **90.73%**
    - Strong performance for a transformer model but not on par with Random Forest.
  + **Logistic Regression & AdaBoost**:
    - Accuracy: 83.5% (Logistic Regression) and 85.59% (AdaBoost), with lower overall performance compared to Random Forest.
* **Insights**:
  + Random Forest's ability to handle structured data features provides it an edge over the LLM.
  + DistilBERT's contextual understanding makes it suitable for more nuanced NLP tasks but may require fine-tuning for better accuracy.

#### **4. Temporal Trends Analysis**

* **Observation**:
  + Ratings dropped by ~20% between Fall 2022 and Fall 2023, followed by a slight recovery with fluctuations.
* **Seasonal Effects**:
  + Product launches or holiday seasons may influence spikes in ratings, possibly tied to customer excitement or expectations.
* **Insights**:
  + Monitoring temporal trends can reveal actionable insights, such as optimizing product features or marketing campaigns during specific periods.

## 

## **6. Conclusion**

This research successfully analyzed customer sentiment towards iPhone products, revealing significant insights regarding regional preferences, popular product features, and factors contributing to satisfaction or dissatisfaction. The **DistilBert model** offers advanced NLP capabilities, the **Random Forest model demonstrates superior overall performance** in the context provided, particularly in terms of handling varied and potentially nuanced sentiment expressions within the data. **This showcases the importance of model selection based on specific task requirements and data characteristics.**

### **6.1 Future Work**

* **Sentiment Analysis Expansion**: Incorporate more demographic data, such as age and gender.
* **Model Improvement**: Utilize newer LLMs, such as **GPT-4**, for even higher accuracy.
* **Actionable Insights**: Apply clustering and temporal analysis insights to develop targeted marketing campaigns and product features.

## **References**

* **Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018)**. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.
* **Le, Q., & Mikolov, T. (2014)**. Distributed Representations of Sentences and Documents. ICML.
* **Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013)**. Efficient Estimation of Word Representations in Vector Space.
* **Lloyd, S. (1982)**. Least Squares Quantization in PCM. IEEE Transactions on Information Theory.
* **Pennington, J., Socher, R., & Manning, C**. (2014). GloVe: Global Vectors for Word Representation.

## **Appendix**

### **A.1 Code Implementation**

### **Extracted Information from Code Cells**

#### **Initial Data Preparation**

1. **Library Installation and Imports**:
   * Libraries such as lime, PyDrive, pandas, and several authentication modules from Google were used. This suggests that the dataset was obtained from Google Drive, which might explain the use of **PyDrive** and **Google OAuth**.
2. **Data Retrieval**:
   * The dataset was downloaded from Google Drive using a shared link with file\_id = '1xzoEQPIZP1tkTPbmRtyEJ29tFLIQHIj9'.

**DataFrame Creation**: The dataset was loaded into a pandas DataFrame named df:  
python  
Copy code  
df = pd.read\_csv("iphone.csv")

1. **Dataset Overview**:

**Shape of the Dataset**: The dataset was examined for its dimensions:  
python  
Copy code  
print(df.shape)

The initial rows and data types were displayed using:  
python  
Copy code  
df.head()

df.info()

* + **Missing Values Check**:

A check was performed to determine if there are any missing values:  
python  
Copy code  
if df.isnull().sum().sum() > 0:

print("There are missing values in the dataframe.")

null\_df = df.isnull().sum()

print(null\_df[null\_df > 0])

else:

print("There are no missing values in the dataframe.")

* + **Duplicate Values Check**:

A check was also performed for duplicates:  
python  
Copy code  
if df.duplicated().sum() > 0:

print("There are duplicate values in the dataframe.")

dupl\_df = df.duplicated().sum()

print(dupl\_df[dupl\_df > 0])

else:

print("There are no duplicate values in the dataframe.")

1. **Unique Feature Analysis**:

The **unique values** of each feature were also computed to get a sense of the variability in the data:  
python  
Copy code  
df.apply(lambda x: [x.nunique(), x.unique()]).T

### 

1. **Summarize Dataset**:
   * Create a summary of the dataset based on the extracted content: number of rows, columns, presence of missing or duplicate values, and an overview of features.
2. **Proceed with Analyzing Visualizations and Models**:
   * Extract subsequent code cells where visualization functions and model implementations may be present.
   * Identify key metrics used to evaluate model performance, such as accuracy, precision, recall, or F1 score.
3. **Develop Insights and Recommendations**:
   * Based on initial data processing and the markdown descriptions, build towards a final recommendation and next steps.

### 

#### **Data Visualization and Analysis**

1. **Rating Score Distribution**:
   * A histogram was plotted to analyze the distribution of the **ratingScore** feature.

Code used:  
python  
Copy code  
plt.figure(figsize=(10, 6))

sns.histplot(df['ratingScore'], color='pink')

plt.title('Distribution of Rating Scores')

plt.xlabel('Rating Score')

plt.ylabel('Frequency')

plt.show()

* + **Purpose**: This visualization provides an overview of the rating scores given by users, highlighting the general distribution and the frequency of each score.

1. **Descriptive Statistics for ratingScore**:
   * Key statistical metrics such as **mean**, **median**, and **mode** were calculated to understand the central tendencies of the ratingScore.

Code used:  
python  
Copy code  
mean\_rating = df['ratingScore'].mean()

median\_rating = df['ratingScore'].median()

mode\_rating = df['ratingScore'].mode()[0]

print(f"Mean Rating: {mean\_rating:.2f}")

print(f"Median Rating: {median\_rating}")

print(f"Mode Rating: {mode\_rating}")

print(df['ratingScore'].describe())

* + **Purpose**: Understanding central tendencies helps identify user satisfaction trends, and analyzing the distribution can reveal potential biases or patterns in the ratings.

1. **Country-wise Rating Score Distribution**:

A histogram was plotted to visualize how **rating scores differ by country**:  
python  
Copy code  
plt.figure(figsize=(12, 6))

sns.histplot(x='ratingScore', hue='country', data=df, multiple="dodge")

plt.title('Rating Score Distribution per Country')

plt.xlabel('Rating Score')

plt.ylabel('Frequency')

plt.show()

* + **Purpose**: The aim is to identify how customer satisfaction varies geographically. This visualization can provide insights into which countries have higher or lower satisfaction rates.

1. **Grouped Analysis of Ratings by Country**:

The data was grouped by country and ratingScore, and a summary table was created:  
python  
Copy code  
rating\_counts = df.groupby(['country', 'ratingScore']).size().unstack(fill\_value=0)

* + **Purpose**: To easily compare the number of ratings across different countries and observe differences in how customers perceive iPhone products.

1. **Variant Feature Analysis (CPU, SSD, Color)**:

**Variant Extraction Function**:  
python  
Copy code  
def extract\_variant\_features(variant\_string):

features = {}

# Assuming the variant string format: "Size:128GB,Color:Space Gray,Service Provider:Verizon"

parts = variant\_string.split(',')

for part in parts:

key\_value = part.split(':')

if len(key\_value) == 2:

features[key\_value[0]] = key\_value[1].strip()

return features

* + **Purpose**: This function aims to extract features such as size, color, and service provider from the dataset for analysis. By comparing these features across different countries, the goal is to understand if certain product configurations are linked to higher satisfaction.

### **Summary of Findings So Far**

* **Rating Distribution**:
  + The visualizations and statistics help us understand how customers perceive iPhone products across different regions.
  + The ratingScore distribution histogram provides a general overview of customer ratings, while the country-wise histogram highlights geographic differences.
* **Key Metrics**:
  + **Mean, Median, and Mode**: Provide a summary of customer sentiment.
  + **Country-Level Analysis**: Differences in customer satisfaction among countries can help uncover region-specific issues or successes.

### **Next Steps for Report Creation**

* **Extract Model Building Cells**:
  + Move on to extract cells that involve building machine learning models (both traditional models and LLMs).
  + Summarize the performance metrics and evaluation methods used for these models.
* **Develop a Detailed Report**:
  + Combine insights from data exploration, visualizations, and modeling.
  + Draft future recommendations for model improvement and product strategies.

### 

#### **Clustering Analysis and Visualization**

1. **Cluster Analysis - Country Distribution**:
   * **Pie Chart of Cluster 0**:
     + A pie chart was created to visualize the **distribution of countries** within **Cluster 0**.

Code used:  
python  
Copy code  
cluster\_0\_df = df[df['cluster'] == 0]

country\_counts = cluster\_0\_df['country'].value\_counts()

plt.figure(figsize=(8, 8))

plt.pie(country\_counts, labels=country\_counts.index, autopct='%1.1f%%', startangle=90)

plt.title('Cluster 0 - Country Distribution')

plt.axis('equal')

plt.show()

* + - **Purpose**: This analysis provides insight into the regional composition of a specific cluster, potentially showing which countries are predominant in this cluster of users with similar sentiment or behavior.

1. **Country Listing and Frequent Words for Cluster 0**:
   * **List of Countries in Cluster 0**:

A list of countries in **Cluster 0** was printed to understand the geographic makeup:  
python  
Copy code  
countries\_in\_cluster = df[df['cluster'] == cluster\_num]['country'].unique()

print(f"Countries in Cluster {cluster\_num}: {', '.join(countries\_in\_cluster)}")

* + **Frequent Words in Cluster 0**:

The **frequent words** in Cluster 0 were identified using an existing function frequent\_words\_in\_cluster(). The frequent words were then plotted as a bar chart:  
python  
Copy code  
if frequent\_words:

words, counts = zip(\*frequent\_words)

plt.figure(figsize=(10, 6))

plt.bar(words, counts)

plt.xlabel('Words')

plt.ylabel('Frequency')

plt.title(f'Most Frequent Words in Cluster {cluster\_num}')

plt.xticks(rotation=45, ha='right')

plt.tight\_layout()

plt.show()

else:

print(f"No frequent words found for Cluster {cluster\_num}")

* + - **Purpose**: The bar chart helps illustrate which terms are most commonly used in reviews within the cluster, providing insights into shared sentiment topics.

1. **Cluster 1 Analysis**:
   * Similar to Cluster 0, the **country distribution** and **frequent words** for **Cluster 1** were analyzed.
   * This kind of analysis highlights differences between clusters, potentially revealing contrasting attitudes or experiences across different user segments.

### **Summary of Findings So Far**

* **Clustering Analysis**:
  + The use of clustering was aimed at segmenting customers based on their reviews and geographic location. This segmentation can help identify specific patterns and preferences.
  + **Country Distribution** for Clusters: Understanding which countries dominate particular clusters helps infer whether certain regions have notably positive or negative sentiments.
  + **Frequent Words Analysis**: This is useful for understanding the topics or features users in different clusters are discussing, which could help the company focus on specific aspects that matter most to different groups.

### **Next Steps for Report Creation**

* **Extract Machine Learning Models**:
  + Focus on extracting more code cells to analyze the machine learning models used for sentiment analysis.
  + Compare **traditional machine learning models** (e.g., Random Forest, Logistic Regression) with **LLMs** (e.g., DistilBERT).
  + Extract information about model performance metrics, such as accuracy, recall, F1 score, and more.
* **Complete the Report**:
  + Compile a complete summary of all findings, visualizations, and models.
  + Develop **future steps** for model improvement and make **recommendations**

### 

#### **Feature Engineering**

1. **Color Normalization**:

The **Color** attribute in the variant\_df was normalized to lowercase for consistency:  
python  
Copy code  
variant\_df['Color'] = variant\_df['Color'].apply(lambda x: x.lower())

1. **Size Extraction**:

Extracted numerical values (e.g., sizes like **64GB** or **128GB**) from the **Size** column using a regular expression:  
python  
Copy code  
def extract\_numbers(s):

pattern\_size = r'\b(\d+)\s?GB?\b'

match = re.search(pattern\_size, s)

if match:

return match.group(1)

else:

return None

sizes = variant\_df['Size'].map(lambda x: extract\_numbers(x))

variant\_df['Size'] = sizes

* + **Purpose**: Standardizing size attributes and making them numeric helps facilitate numerical comparisons and feature engineering for the machine learning models.

1. **Dropping Unnecessary Columns**:

Columns such as **Color**, **Size**, and **Service\_Provider** were dropped from the main DataFrame df:  
python  
Copy code  
df = df.drop(['Color', 'Size', 'Service\_Provider'], axis=1, errors='ignore')

* + **Reason**: These features were likely redundant or transformed into a more usable form in variant\_df.

1. **Concatenating the Transformed Variant Data**:

The variant\_df was concatenated back to the main df to create an enriched dataset:  
python  
Copy code  
df\_2 = pd.concat([df, variant\_df], axis=1)

* + **Purpose**: This ensures that all the cleaned and normalized features are available for model training.

#### **Data Visualizations and Insights**

1. **Review Share of iPhone by Country**:

A **count plot** was created to visualize the distribution of reviews by country:  
python  
Copy code  
plt.figure(figsize=(10, 4))

sns.countplot(y='country', data=df)

plt.grid(axis='x', linestyle="--", color="white")

plt.title("Review Share of iPhone")

plt.show()

* + **Purpose**: This plot helps identify which countries have the highest or lowest number of reviews, potentially indicating key markets or underserved areas.

1. **Most Sold Colors Globally and by Country**:

A **pie chart** and **stacked bar chart** were created to visualize the **most sold colors** of iPhones both globally and country-wise:  
python  
Copy code  
fig, ax = plt.subplots(1, 2, figsize=(10, 6))

g = df\_2.groupby(by=['Color'])['productAsin'].agg(['count'])

g['percentage'] = round(100 \* g['count'] / df\_2.shape[0], 2)

g = g.sort\_values(by='percentage')

ax[0].pie(x=g.percentage, labels=g.index, autopct='%.2f%%')

ax[0].set\_title("Global")

# Group by country

g = df\_2.pivot\_table(values='productAsin', index=['country'], aggfunc='count', columns='Color', fill\_value=0)

g.plot(kind='bar', stacked=True, ax=ax[1], title="Color Country-wise")

plt.suptitle("Most Selling Colors")

plt.tight\_layout()

plt.show()

* + **Purpose**: The **color preferences** by region could be an important factor for marketing and inventory strategies. Knowing which colors are more popular helps Apple understand the needs of specific markets better.

### **Summary of Feature Engineering and Visual Analysis**

* **Feature Normalization**:
  + Columns like Color and Size were transformed to numeric or standardized text values for easier processing in machine learning models.
* **Color and Size Information**:
  + The detailed visualization of **color popularity** shows valuable marketing insights into consumer preferences globally and regionally.
* **Review Distribution**:
  + The count plot for reviews helps identify key geographical regions where more or fewer reviews are being gathered.

### **Next Steps for Model Analysis**

* **Focus on Machine Learning Models**:
  + Extract more cells to understand how machine learning models (traditional models like Random Forest, LLMs like DistilBERT) were trained and evaluated.
  + Summarize the models used, performance metrics, and compare their strengths and weaknesses.

### 

#### **Feature Engineering - Size Distribution Analysis**

1. **Global and Country-wise Size Distribution**:
   * **Pie Chart for Size Distribution**:

A **pie chart** was plotted to represent the **global distribution of iPhone sizes**:  
python  
Copy code  
g = df\_2.groupby(by=['Size'])['productAsin'].agg(['count'])

g['percentage'] = round(100 \* g['count'] / df\_2.shape[0], 2)

g = g.sort\_values(by='percentage')

ax[0].pie(x=g.percentage, labels=g.index, autopct='%.2f%%')

ax[0].set\_title("Global")

* + - **Purpose**: The goal here is to visualize which iPhone sizes are most popular globally.
  + **Stacked Bar Chart by Country**:

A **stacked bar chart** was plotted to visualize the **distribution of sizes by country**:  
python  
Copy code  
g = df\_2.pivot\_table(values='productAsin', index=['country'], aggfunc='count', columns='Size', fill\_value=0)

g.plot(kind='bar', stacked=True, ax=ax[1], title="Size Country wise")

* + - **Purpose**: This analysis helps understand if specific countries have preferences for certain sizes, which could be useful for product planning and inventory distribution.

#### **Clustering - Frequent Words Extraction from Reviews**

1. **Frequent Words Extraction**:
   * **Frequent Words in Reviews by Cluster**:

The function frequent\_words\_in\_cluster() was implemented to identify the **most frequent words in each cluster**:  
python  
Copy code  
def frequent\_words\_in\_cluster(cluster\_df):

stop\_words = set(stopwords.words('english'))

all\_words = []

for review in cluster\_df['reviewDescription']:

words = nltk.word\_tokenize(str(review).lower())

words = [word for word in words if word.isalnum() and word not in stop\_words]

all\_words.extend(words)

return Counter(all\_words).most\_common(10)

* + - **Purpose**: Extracting the most common words allows the identification of themes or common topics that resonate with customers within specific clusters. This can help in refining customer sentiment understanding.

### **Summary of Feature Analysis and Clustering**

* **Size Analysis**:
  + The global and country-wise analysis of **iPhone sizes** provides valuable insights into product variants that are more popular in different regions.
* **Frequent Word Extraction**:
  + Extracting frequent words from review descriptions is useful for understanding key factors driving customer sentiment.

1. **Machine Learning Model Implementation**:
   * Extract code related to traditional ML models like Random Forest or Logistic Regression.
   * Analyze large language models (LLMs) like DistilBERT, which might have been fine-tuned for sentiment analysis.
2. **Evaluate Model Performance**:
   * Summarize key metrics, such as accuracy, precision, recall, and F1 score, used to evaluate model performance.

### 

#### **Text Preprocessing Steps for Machine Learning Models**

1. **Text Preprocessing Function**:
   * A function text\_preprocessing() was defined for cleaning textual data in the columns reviewTitle, reviewDescription, and reviewedIn.
   * **Preprocessing Steps**:
     + **Lowercasing**: Convert all text to lowercase for uniformity.
     + **HTML Tag Removal**: Remove HTML tags and non-ASCII characters using regular expressions.
     + **Punctuation Removal**: Remove punctuation marks.
     + **Stop Words Removal**: Use the **NLTK stop words** set to remove commonly used words that do not contribute to the sentiment (e.g., "the", "and", etc.).

**Code Example**:  
python  
Copy code  
def text\_preprocessing(text):

if isinstance(text, str):

text = text.lower()

text = re.sub(r'<.\*?>|[^\x00-\x7f]', '', text)

text = re.sub(f"[{re.escape(string.punctuation)}]", " ", text)

stop\_words = set(stopwords.words("english"))

words = [word for word in text.split() if word not in stop\_words]

cleaned\_text = ' '.join(words)

return cleaned\_text

else:

return ''

1. **Application of Preprocessing**:

The preprocessing function was applied to the text columns (reviewTitle, reviewDescription, reviewedIn) of the DataFrame:  
python  
Copy code  
df["reviewTitle"] = df["reviewTitle"].apply(text\_preprocessing)

df["reviewDescription"] = df["reviewDescription"].apply(text\_preprocessing)

df["reviewedIn"] = df["reviewedIn"].apply(text\_preprocessing)

* + **Purpose**: This preprocessing ensures that the data is clean, consistent, and ready for machine learning algorithms by reducing the vocabulary size and removing noise.

1. **Download NLTK Stopwords**:

The **NLTK stop words** were downloaded to facilitate the removal of common English words:  
python  
Copy code  
nltk.download('stopwords')

1. **WordCloud Generation** (as inferred from imports):

The **WordCloud** package was imported, indicating that visualizations to represent word frequencies in reviews were likely generated:  
python  
Copy code  
from wordcloud import WordCloud

* + **Purpose**: The WordCloud is often used to visualize the most common words in text data, helping to understand key themes in customer reviews.

### **Summary of Text Preprocessing**

* **Uniform Text Representation**:
  + Textual data was transformed to lowercase, with punctuation and stop words removed, ensuring that models could focus on meaningful features.
* **Text Cleaning**:
  + HTML tags and non-ASCII characters were removed, making the data suitable for traditional ML models and modern LLMs.

### **Next Steps for Report Completion**

The text preprocessing code extracted above is a crucial step in preparing the data for machine learning models, ensuring the quality of features used for prediction.

Next steps:

1. **Extract Model Training Code**:
   * Identify traditional ML models (e.g., Random Forest, Logistic Regression) and fine-tune LLMs (e.g., DistilBERT).
   * Focus on extracting performance metrics and understanding the effectiveness of each model in the context of sentiment analysis.
2. **Compare Performance**:
   * Compare the **accuracy, precision, recall, and F1 score** of different models to understand their strengths and weaknesses.

### 

#### **Text Vectorization Using TF-IDF and Lemmatization**

1. **TF-IDF Vectorization**:

The **TF-IDF (Term Frequency-Inverse Document Frequency) Vectorizer** was initialized to convert the text data into numerical vectors that machine learning models can process:  
python  
Copy code  
from sklearn.feature\_extraction.text import TfidfVectorizer

tfidf\_vectorizer = TfidfVectorizer(tokenizer=Lemmatizer().lemmatize, ngram\_range=(1, 2))

* + **N-grams**: The vectorizer uses **1-gram and 2-gram** ranges to capture individual words as well as two-word combinations, allowing the model to recognize both individual terms and short phrases that might carry meaning in customer reviews.

1. **Lemmatization**:
   * A custom **Lemmatizer** class was used for preprocessing:
     + The Lemmatizer class leverages **WordNetLemmatizer** from NLTK to reduce words to their base forms (lemmas).

The method lemmatize() processes tokens to identify their **POS tags** (using pos\_tag()) and applies appropriate lemmatization for each token:  
python  
Copy code  
class Lemmatizer:

def \_\_init\_\_(self):

self.lemmatizer = WordNetLemmatizer()

def get\_wordnet\_pos(self, tag):

if tag.startswith('J'):

return wordnet.ADJ

elif tag.startswith('V'):

return wordnet.VERB

elif tag.startswith('N'):

return wordnet.NOUN

elif tag.startswith('R'):

return wordnet.ADV

else:

return None

def lemmatize(self, text):

tokens = word\_tokenize(text)

tagged\_tokens = pos\_tag(tokens)

lemmatized\_tokens = []

for token, tag in tagged\_tokens:

wordnet\_pos = self.get\_wordnet\_pos(tag) or wordnet.NOUN

lemma = self.lemmatizer.lemmatize(token, pos=wordnet\_pos)

lemmatized\_tokens.append(lemma)

return ' '.join(lemmatized\_tokens)

* + **Purpose**: Lemmatization helps in reducing inflected words to their root form, minimizing vocabulary size while retaining semantic meaning.

1. **Transforming Text Data Using TF-IDF**:

After text preprocessing, the **TF-IDF vectorizer** was applied to both the training and testing datasets:  
python  
Copy code  
X\_train\_tfidf = tfidf\_vectorizer.fit\_transform(X\_train) # Fit and transform on training data

X\_test\_tfidf = tfidf\_vectorizer.transform(X\_test) # Only transform on test data

* + **Handling Missing Values**:

Missing values in the text columns were handled by replacing NaNs with empty strings before applying TF-IDF:  
python  
Copy code  
X\_train = X\_train.fillna('')

X\_test = X\_test.fillna('')

* + **Purpose**: The TF-IDF representation provides numerical values representing the importance of each term, enabling machine learning models to learn relationships between words and sentiments.

### **Summary of Vectorization and Text Preprocessing**

* **TF-IDF Vectorization**:
  + Converts textual data into numerical features, allowing machine learning algorithms to learn from the dataset.
  + **N-grams (1, 2)** allow capturing phrases and individual words, which is especially important in the sentiment analysis context.
* **Lemmatization**:
  + Reducing words to their base forms ensures that words with similar meanings are treated the same, minimizing noise and improving model performance.

1. **Training the Machine Learning Models**:
   * Extract details on which machine learning models were used (e.g., **Random Forest, Logistic Regression, LLMs**).
   * Analyze model training procedures and identify hyperparameters used.
2. **Evaluate Model Performance**:
   * Extract evaluation metrics to compare traditional models against LLMs.
   * Develop insights and final recommendations based on model performance.

​​

### 

#### **1. Testing with Random Samples**

* **Sentiment Testing**:
  + A set of **random reviews** were tested using a trained sentiment analysis pipeline named sentiment\_pipe to predict sentiment as either **"NEGATIVE"** or **"POSITIVE"**.

Example Reviews Tested:  
python  
Copy code  
sample = [

'iphone battery sucks, whenever I connect to the charger it takes 2 hours to charge fully',

'I love iPhone but this time they have not provided a good battery life',

'This iPhone camera is amazing, and so is the color purple.',

'iPhone is amazing',

'Charger is so bad, I returned the order'

]

labels = [0, 0, 1, 1, 0] # Ground truth labels for the reviews

* + **Mapping Sentiment**:
    - The output label 0 corresponds to **"NEGATIVE"** and 1 to **"POSITIVE"**.

A lambda function was used to map the labels:  
python  
Copy code  
mapper = lambda x: 'NEGATIVE' if x == 0 else 'POSITIVE'

sample\_y = list(map(mapper, labels))

* + **Model Prediction**:
    - The sentiment\_pipe was used to predict the sentiment of the sample reviews.

#### **2. Fine-tuning a DistilBERT Model for Sentiment Analysis**

* **Dataset Preparation**:
  + The dataset used includes combined **review titles and review descriptions** (X) and a label indicating whether the sentiment is positive (y).

The dataset was restructured to work with the DistilBERT tokenizer and training workflow:  
python  
Copy code  
X = df['reviewTitle'] + ' ' + df['reviewDescription']

y = df['IsPositiveRating']

dataset = pd.concat([X, y], axis=1)

dataset = dataset.rename(columns={0: 'text', 'IsPositiveRating': 'label'})

* **Tokenizer Loading and Dataset Tokenization**:
  + **DistilBERT Tokenizer**:

The tokenizer from **distilbert-base-uncased** was loaded to tokenize the dataset:  
python  
Copy code  
tokenizer = DistilBertTokenizerFast.from\_pretrained('distilbert-base-uncased')

* + **Tokenization**:

A function was created to **tokenize the dataset**:  
python  
Copy code  
def tokenize\_func(examples):

return tokenizer(examples['text'], padding='max\_length', truncation=True)

* + - The dataset was tokenized for the DistilBERT model.
* **Dataset Splitting**:

The tokenized dataset was split into training and evaluation sets using an 80-20 split:  
python  
Copy code  
train\_test\_split = tokenized\_datasets.train\_test\_split(test\_size=0.2)

train\_dataset = train\_test\_split['train']

eval\_dataset = train\_test\_split['test']

* **Model Loading and Training Arguments**:
  + **DistilBERT Model**:

The **DistilBERT for Sequence Classification** was loaded for the sentiment analysis task:  
python  
Copy code  
model = DistilBertForSequenceClassification.from\_pretrained('distilbert-base-uncased')

* + **Training Arguments**:

Default training arguments were set using TrainingArguments():  
python  
Copy code  
training\_args = TrainingArguments(

output\_dir='./results',

evaluation\_strategy="steps",

per\_device\_train\_batch\_size=16,

per\_device\_eval\_batch\_size=16,

num\_train\_epochs=3,

save\_steps=10,

save\_total\_limit=2,

logging\_dir='./logs',

)

### **Summary of Modeling Approach**

* **Traditional Testing Approach**:
  + Initial testing was performed on random customer review samples to validate sentiment prediction.
  + The sentiment model (sentiment\_pipe) was used to predict each review's sentiment and compare it with the actual sentiment.
* **DistilBERT Implementation**:
  + A **fine-tuning approach** was applied using **DistilBERT** for sentiment classification.
  + The review titles and descriptions were tokenized and used for training a sequence classification model.
  + The dataset was split into **training** and **evaluation** sets for training purposes.
  + **TrainingArguments** were configured to train the model effectively and save the progress at different steps.

### 