Introduction

In response to the growing influx of customer feedback received monthly by a retail company, this project focuses on leveraging Natural Language Processing (NLP) techniques to automate the analysis and categorization of customer reviews. The aim is to enhance operational efficiency by classifying reviews into positive, negative, or neutral sentiments, and subsequently providing detailed summaries based on review scores across various product categories. Additionally, the project will culminate in the creation of an interactive visualization dashboard to facilitate real-time insights and decision-making.

This business case outlines the development of an NLP model tailored for automated customer feedback processing. By implementing advanced machine learning techniques, the project seeks to alleviate the challenges posed by manual review categorization and analysis. Key objectives include developing robust classification models for sentiment analysis, generating comprehensive summaries categorized by review ratings, and visualizing insights through a dynamic dashboard.

Traditional NLP & ML approaches

Preparation steps

Importing the libraries

```
import pandas as pd
from google.colab import drive
import numpy as np
from sklearn.utils.class_weight import compute_class_weight
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Bidirectional, LSTM, Dense, Dropout, GRU, Conv1D, MaxPooling1D
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.regularizers import l2
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, ConfusionMatrixDi
import matplotlib.image  as mpimg
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler
from collections import Counter
from wordcloud import WordCloud
import re
import nltk
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score, GridSearchCV
from sklearn.pipeline import Pipeline
from nltk.translate.bleu_score import corpus_bleu
from rouge_score import rouge_scorer
import random



nltk.download('stopwords')
nltk.download('punkt')
nltk.download('wordnet')


from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
from nltk.translate.bleu_score import sentence_bleu, corpus_bleu

```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
```

```
[nltk_data]   Package wordnet is already up-to-date!
```

Loading the dataset

When loading the contents from the original dataset, we considered only the rows 'asins','id', 'categories', 'name','reviews.date', 'reviews.rating', 'reviews.text', and 'reviews.title'.

```
# The dataset is the publicly available and downsized dataset of Amazon customer reviews from their online marketplace,
# We are loading our dataset from a Google Drive account

drive.mount('/content/drive')

# Load the dataset
nlp_df = pd.read_csv('/content/drive/MyDrive/Project1/Consumer Reviews of Amazon Products/1429_1.csv', usecols =[ 'asins


```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remour
<ipython-input-3-6a7fdae275f3>:7: DtypeWarning: Columns (1) have mixed types. Specify dtype option on import or set low_
  nlp_df = pd.read_csv('/content/drive/MyDrive/Project1/Consumer Reviews of Amazon Products/1429_1.csv', usecols =[ 'as:
```

```
# Visualize the first 5 rows of the dataframe
nlp_df.head()
```

	id	name	asins	categories	reviews.date	reviews.
0	AVqkIhwDv8e3D1O-lebb	All-New Fire HD 8 Tablet, 8 HD Display, Wi-Fi,...	B01AHB9CN2	Electronics,iPad & Tablets,All Tablets,Fire Ta...	2017-01-13T00:00:00.000Z	
1	AVqkIhwDv8e3D1O-lebb	All-New Fire HD 8 Tablet, 8 HD Display,	B01AHB9CN2	Electronics,iPad & Tablets,All Tablets,Fire Ta...	2017-01-13T00:00:00.000Z	

Data Preprocessing

Preparing the dataset

In the following cells, we perform data cleaning by removing rows where either the 'reviews.rating' or 'reviews.text' columns contain null or NaN values from the DataFrame df.

After cleaning, we initialize two lists: texts containing all cleaned review texts and labels containing corresponding ratings.

Finally, we print the lengths of these lists to verify the number of non-null entries remaining after data cleaning and extraction

```
# Dropping rows with null/NaN values
nlp_df = nlp_df.dropna(subset=['reviews.rating', 'reviews.text'])

# Initializing the texts and labels

texts = nlp_df['reviews.text'].tolist()
labels = nlp_df['reviews.rating'].tolist()

# Check lengths after cleaning
print(len(texts))
print(len(labels))

```

```
34626
34626
```

Split into training, validation, and test sets

In this section of code, we use train_test_split from sklearn.model_selection to divide our dataset into training, validation, and test sets in a 60-20-20 ratio.

First, we split the original texts and labels into training and test sets, ensuring balanced distribution using stratify=labels and setting random_state=42 for reproducibility.

Next, we further split the test set into validation and final test sets using the same stratify=test_labels and random_state=42 parameters.

Finally, we print the sizes of each set to verify the distribution to ensure we have properly partitioned our data for training, validation, and testing purposes, maintaining consistency and effectiveness in model evaluation.

```
# Split the dataset into train and validation with a 60-40 ratio
training_texts, test_texts, training_labels, test_labels = train_test_split(texts, labels, test_size=.4, random_state=42

# Split the validation dataset into validation and test sets with a 50-50 ratio
validation_texts, test_texts, validation_labels, test_labels = train_test_split(test_texts, test_labels, test_size=.5, r

# Print the sizes of each set
print(f'Training set size: {len(training_texts)} and {len(training_labels)}')
print(f'Validation set size: {len(validation_texts)}')
print(f'Test set size: {len(test_texts)}')
```

```
Training set size: 20775 and 20775
Validation set size: 6925
Test set size: 6926
```

Map ratings into sentiments

In this segment of code, we utilize the Counter class from Python's collections module to tally occurrences of each category following the mapping of ratings to three distinct labels: negative, neutral, and positive. This mapping is achieved through the map_ratings_to_labels function, which categorizes ratings based on predefined thresholds.

Subsequently, we apply this mapping to the training, validation, and test sets' labels, generating mapped_training_labels, mapped_validation_labels, and mapped_test_labels respectively.

We then employ Counter to compute the frequency of each category within these mapped label sets, resulting in training_label_counts, validation_label_counts, and test_label_counts.

Lastly, we print the counts to observe the distribution across categories in each dataset.

```
# Function to map ratings to three categories
def map_ratings_to_labels(rating):
     """
    This function takes an integer rating as input and returns a corresponding sentiment label:
    - Ratings of 1 or 2 are mapped to 0, indicating a negative sentiment.
    - A rating of 3 is mapped to 1, indicating a neutral sentiment.
    - Ratings of 4 or higher are mapped to 2, indicating a positive sentiment.

    Parameters:
    rating (int): The numerical rating to be mapped. Expected values are integers typically in the range of 1 to 5.

    Returns:
    int: The sentiment label corresponding to the input rating. The labels are:
         0 - Negative
         1 - Neutral
         2 - Positive
    """
     if rating in [1, 2]:
         return 0  # Negative
     elif rating == 3:
         return 1  # Neutral
     else:
         return 2  # Positive

```

```
# Map training, validation, and test labels to categories
mapped_training_labels = [map_ratings_to_labels(rating) for rating in training_labels]
mapped_validation_labels = [map_ratings_to_labels(rating) for rating in validation_labels]
mapped_test_labels = [map_ratings_to_labels(rating) for rating in test_labels]

# Count occurrences in each mapped labels set
training_label_counts = Counter(mapped_training_labels)
validation_label_counts = Counter(mapped_validation_labels)
test_label_counts = Counter(mapped_test_labels)

# Display the counts
print("Training label counts:", training_label_counts)
print("Validation label counts:", validation_label_counts)
print("Test label counts:", test_label_counts)

```

```
Training label counts: Counter({2: 19389, 1: 899, 0: 487})
Validation label counts: Counter({2: 6463, 1: 300, 0: 162})
Test label counts: Counter({2: 6463, 1: 300, 0: 163})
```

Clean and preprocess the reviews text

In this code block, we import the re module for handling regular expressions and nltk along with its stopwords corpus for text preprocessing. The clean_text function is designed to process input text by stripping non-alphanumeric characters, reducing whitespace, converting text to lowercase, and removing common English stopwords. This ensures that the text is appropriately cleaned and prepared for further analysis or modeling tasks.

This part of the code imports NLTK's word_tokenize for word tokenization and WordNetLemmatizer for lemmatizing words. It ensures the required NLTK resources (punkt for tokenization and wordnet for lemmatization) are downloaded. The lemmatize_text function would subsequently tokenize input text into words and apply lemmatization to reduce words to their base forms, facilitating semantic analysis and improving text processing accuracy.

```
def preprocess_text(text):
    """
    This function performs several preprocessing steps on the input text:
    1. Handles non-string types by returning an empty string.
    2. Removes special characters and punctuation, keeping only alphanumeric characters and spaces.
    3. Removes unnecessary whitespace and trims the text.
    4. Converts the text to lowercase.
    5. Tokenizes the text into individual words.
    6. Removes stopwords and numeric tokens.
    7. Lemmatizes the tokens to their base forms.
    8. Returns the cleaned and processed text as a single string.

    Parameters:
    text (str): The input text to be preprocessed.

    Returns:
    str: The preprocessed and cleaned text.
    """

# Handle non-string types
    if not isinstance(text, str):
      return ""  # Or handle it differently based on your needs

    # Remove special characters and punctuation (excluding numbers and spaces)
    text = re.sub(r'[^\w\s]', '', text)
    # Remove unnecessary whitespace
    text = re.sub(r'\s+', ' ', text).strip()
    # Convert text to lowercase
    text = text.lower()

    # Tokenize the text
    tokens = word_tokenize(text)

    # Remove stopwords and numeric tokens
    stop_words = set(stopwords.words('english'))
    tokens = [token for token in tokens if token not in stop_words and not token.isdigit()]

    # Lemmatize tokens
    lemmatizer = WordNetLemmatizer()
    tokens = [lemmatizer.lemmatize(token) for token in tokens]

    # Return cleaned text as a single string
    return ' '.join(tokens)
```

```
# Apply preprocessing to training, validation, and test sets
preprocessed_training_texts = [preprocess_text(text) for text in training_texts]
preprocessed_validation_texts = [preprocess_text(text) for text in validation_texts]
preprocessed_test_texts = [preprocess_text(text) for text in test_texts]
```

Transform the data with TfidfVectorizer

This section of code employs TfidfVectorizer to transform text data into TF-IDF matrices, and create the features for the models.

When we transform the text data into a numerical format, each word or n-gram in the vocabulary of the dataset is assigned a unique feature. The TF-IDF score represents how important a word is to a document in a collection of documents.

First, the vectorizer is initialized and trained on preprocessed training texts (preprocessed_training_texts). Then, it applies the trained vectorizer to transform validation and test datasets (preprocessed_validation_texts and preprocessed_test_texts).

```
# Initialize TfidfVectorizer
# max_features=5000 means taht the vectorizer will choose only the top max_features ordered by term frequency across the
# ngram_range=(1, 2) means the vectorizer will generate features that are single words and pairs of consecutive words
# min_df=5 means that a word must appear in at least 5 different documents (texts) to be included as a feature
# max_df=0.8 means that words appearing in more than 80% of the documents (texts) will be ignored as they are too common

tfidf_vectorizer = TfidfVectorizer(max_features=5000, ngram_range=(1, 2), min_df=5, max_df=0.8)

# Fit-transform on training data
vectorized_training_texts = tfidf_vectorizer.fit_transform(preprocessed_training_texts)

# Transform validation and test data using the fitted vectorizer
vectorized_validation_texts = tfidf_vectorizer.transform(preprocessed_validation_texts)
vectorized_test_texts = tfidf_vectorizer.transform(preprocessed_test_texts)
```

```
# Convert TF-IDF matrices to pandas DataFrame for easier inspection (optional)
def tfidf_to_dataframe(matrix, feature_names):
    """
    This function takes a TF-IDF matrix and a list of feature names, and returns a pandas DataFrame
    where each row corresponds to a document and each column corresponds to a term's TF-IDF score.

    Parameters:
    matrix (scipy.sparse.csr.csr_matrix): The TF-IDF matrix to be converted. This is typically the output of a TF-IDF ve
    feature_names (list of str): The list of feature names (terms) corresponding to the columns of the matrix.

    Returns:
    pd.DataFrame: A pandas DataFrame where each row represents a document and each column represents a term's TF-IDF sco

    """
    return pd.DataFrame(matrix.toarray(), columns=feature_names)

# Get feature names (vocabulary)
feature_names = tfidf_vectorizer.get_feature_names_out()

# Convert TF-IDF matrices to DataFrames for inspection
df_training = tfidf_to_dataframe(vectorized_training_texts, feature_names)

# Display the TF-IDF DataFrames
print("Training TF-IDF Matrix:")
df_training
```

Training TF-IDF Matrix:

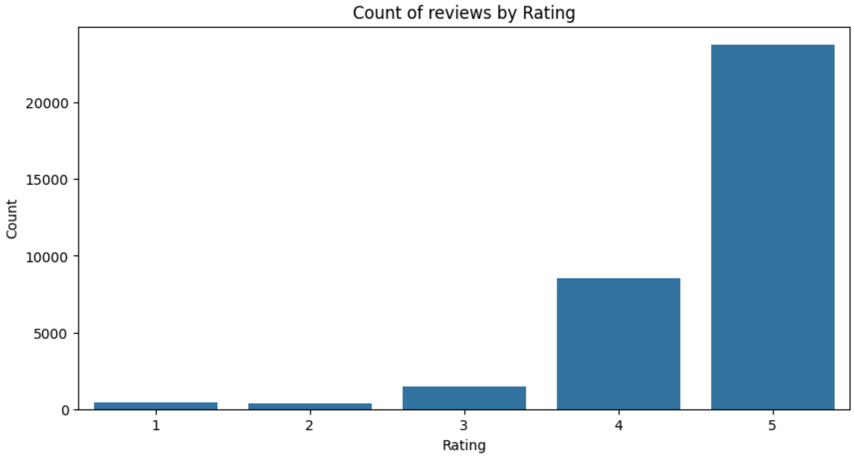
	1080p	128gb	16gb	1st	1st gen	2nd	2nd generation	2nd one	32gb	3g	...	youre
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...
20770	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0
20771	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0
20772	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0
20773	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0
20774	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0

20775 rows × 5000 columns

Data Insights

```
# Ensure 'categories' and 'reviews.rating' are strings
nlp_df['categories'] = nlp_df['categories'].astype(str)
nlp_df['reviews.rating'] = nlp_df['reviews.rating'].astype(int)

# Group by 'categories' and 'reviews.rating' and count the occurrences
category_rating_counts = nlp_df.groupby(['reviews.rating']).size().reset_index(name='count')
```

```
# Plot using barplot
plt.figure(figsize=(10, 5))
sns.barplot(x='reviews.rating', y='count', data=category_rating_counts)

plt.title('Count of reviews by Rating')
plt.xlabel('Rating')
plt.ylabel('Count')
plt.show()
```



Word Cloud

In this section, the code employs the WordCloud library and matplotlib for visualizing TF-IDF scores of terms. It starts by extracting feature names (vocabulary) using tfidf_vectorizer.get_feature_names_out() and aggregating TF-IDF scores across documents. These scores are mapped to terms in term_tfidf_scores. Using the WordCloud object configured with dimensions and background color, it generates a visual representation of term importance based on TF-IDF scores. Finally, it displays the word cloud using matplotlib, offering a visual insight into the most significant terms in the dataset.

```
# Get feature names (vocabulary)
feature_names = tfidf_vectorizer.get_feature_names_out()

# Aggregate TF-IDF scores across all documents
tfidf_scores = vectorized_training_texts.sum(axis=0).A1

# Create a dictionary mapping terms to their TF-IDF scores
term_tfidf_scores = {term: score for term, score in zip(feature_names, tfidf_scores)}

# Initialize WordCloud object
wordcloud = WordCloud(width=800, height=400, background_color='white')

# Generate word cloud based on TF-IDF scores
wordcloud.generate_from_frequencies(term_tfidf_scores)

# Display the word cloud using matplotlib
plt.figure(figsize=(16, 6))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()

```


```
Start coding or generate with AI.
```

Model Building

Naive Bayes

```
# Create Pipeline for Naive Bayes
pipeline_nb = Pipeline([
    ('nb', MultinomialNB())
])

# Cross-validation
cv_scores_nb = cross_val_score(pipeline_nb, vectorized_validation_texts, mapped_validation_labels, cv=5)

# Hyperparameter tuning with GridSearchCV
param_grid_nb = {'nb__alpha': [0.1, 0.5, 1.0]}
grid_search_nb = GridSearchCV(pipeline_nb, param_grid_nb, cv=5)
grid_search_nb.fit(vectorized_validation_texts, mapped_validation_labels)

```

```
▸ GridSearchCV
▸ estimator: Pipeline
  ▸ MultinomialNB
```

```
# Calculate mean cross-validation score as percentage
mean_cv_score_nb = cv_scores_nb.mean() * 100

# Print the scores
print(f"Mean CV Score: {mean_cv_score:.2f}%")
print(f"Best Params: {grid_search_nb.best_params_}")
print(f"Best CV Score: {grid_search_nb.best_score_:.2%}")
```

```
Mean CV Score: 93.33%
Best Params: {'nb__alpha': 0.1}
Best CV Score: 93.36%
```

```
# Fit the Naive Bayes pipeline on vectorized training data
pipeline_nb.fit(vectorized_training_texts, mapped_training_labels)

# Predict on validation data
y_pred_nb = pipeline_nb.predict(vectorized_validation_texts)

# Generate classification report
classification_rep_nb = classification_report(mapped_validation_labels, y_pred_nb, zero_division=1)

```

```
# Print classification report
print("Classification Report for Support Vector Machine:")
print(classification_rep_nb)
```

```
Classification Report for Support Vector Machine:
              precision    recall  f1-score   support

           0       1.00      0.00      0.00       162
           1       0.00      0.00      0.00       300
           2       0.93      1.00      0.97      6463

    accuracy                           0.93      6925
   macro avg       0.64      0.33      0.32      6925
weighted avg       0.89      0.93      0.90      6925
```

Logistic Regression

```
# Pipeline for Logistic Regression
pipeline_lr = Pipeline([
    ('lr', LogisticRegression(solver='liblinear', max_iter=200))  # set max_iter as 200 so the solver will execute to co
])

# Define the parameter grid
param_grid_lr = {
    'C': [0.1, 0.5, 1.0],
    'max_iter': [100, 200, 300],
    'penalty': ['l1', 'l2']
}

# Cross-validation
cv_scores_lr = cross_val_score(pipeline_lr, vectorized_validation_texts, mapped_validation_labels, cv=5)

# Hyperparameter tuning with GridSearchCV
param_grid_lr = {'lr__C': [0.1, 1.0, 10.0]}
grid_search_lr = GridSearchCV(pipeline_lr, param_grid_lr, cv=5)
grid_search_lr.fit(vectorized_validation_texts, mapped_validation_labels)

```

```
▸ GridSearchCV
▸ estimator: Pipeline
▸ LogisticRegression
```

```
# Calculate mean cross-validation score as percentage
mean_cv_score_lr = cv_scores_lr.mean() * 100

# Print the scores
print(f"Mean CV Score: {mean_cv_score_lr:.2f}%")
print(f"Best Params: {grid_search_lr.best_params_}")
print(f"Best CV Score: {grid_search_lr.best_score_:.2%}")
```

```
Mean CV Score: 93.33%
Best Params: {'lr__C': 10.0}
Best CV Score: 93.43%
```

```
# Fit the Logistic Regression pipeline on vectorized training data
pipeline_lr.fit(vectorized_training_texts, mapped_training_labels)

# Predict on validation data
y_pred_lr = pipeline_lr.predict(vectorized_validation_texts)

# Generate classification report
classification_rep_lr = classification_report(mapped_validation_labels, y_pred_lr, zero_division=1)

```

```
# Print classification report
print("Classification Report for Logistic Regression:")
print(classification_rep_lr)
```

```
Classification Report for Logistic Regression:
              precision    recall  f1-score   support

           0       0.56      0.03      0.06       162
           1       0.52      0.04      0.07       300
           2       0.94      1.00      0.97      6463

    accuracy                           0.93      6925
   macro avg       0.67      0.36      0.37      6925
weighted avg       0.91      0.93      0.91      6925
```

Support Vector Machines (SVM)

```
# Pipeline for SVM
pipeline_svm = Pipeline([
    ('svm', SVC())
])

# Cross-validation
cv_scores_svm = cross_val_score(pipeline_svm, vectorized_validation_texts, mapped_validation_labels, cv=5)

# Hyperparameter tuning with GridSearchCV
param_grid_svm = {
    'svm__C': [0.1, 1, 10, 100],
    'svm__gamma': [1, 0.1, 0.01, 0.001],
    'svm__kernel': ['linear', 'rbf']
}



grid_search_svm = GridSearchCV(pipeline_svm, param_grid_svm, cv=5)
grid_search_svm.fit(vectorized_validation_texts, mapped_validation_labels)

```

```
▸ GridSearchCV
▸ estimator: Pipeline
    ▸ SVC
```

```
# Calculate mean cross-validation score as percentage
mean_cv_score_svm = cv_scores_svm.mean() * 100

# Print the scores
print(f"Mean CV Score: {mean_cv_score_svm:.2f}%")
print(f"Best Params: {grid_search_svm.best_params_}")
print(f"Best CV Score: {grid_search_svm.best_score_:.2%}")
```

```
Mean CV Score: 93.33%
Best Params: {'svm__C': 10, 'svm__gamma': 0.1, 'svm__kernel': 'rbf'}
Best CV Score: 93.46%
```

```
# Fit the SVM pipeline on vectorized training data
pipeline_svm.fit(vectorized_training_texts, mapped_training_labels)

# Predict on validation data
y_pred_svm = pipeline_svm.predict(vectorized_validation_texts)

# Generate classification report
classification_rep_svm = classification_report(mapped_validation_labels, y_pred_svm, zero_division=1)
```

```
# Print classification report
print("Classification Report for Support Vector Machine:")
print(classification_rep_svm)
```

```
Classification Report for Support Vector Machine:
              precision    recall  f1-score   support

           0       0.67      0.01      0.02       162
           1       0.67      0.01      0.01       300
           2       0.93      1.00      0.97      6463

    accuracy                           0.93      6925
   macro avg       0.76      0.34      0.33      6925
weighted avg       0.92      0.93      0.90      6925
```

Random Forest

```
# Pipeline for Random Forest
pipeline_rf = Pipeline([
    ('rf', RandomForestClassifier())
])

# Cross-validation
cv_scores_rf = cross_val_score(pipeline_rf, vectorized_validation_texts, mapped_validation_labels, cv=5)

# Hyperparameter tuning with GridSearchCV
param_grid_rf = {'rf__n_estimators': [50, 100, 200], 'rf__max_depth': [None, 10, 20]}
grid_search_rf = GridSearchCV(pipeline_rf, param_grid_rf, cv=5)
grid_search_rf.fit(vectorized_validation_texts, mapped_validation_labels)
```

```
▸ GridSearchCV
▸ estimator: Pipeline
▸ RandomForestClassifier
```

```
# Calculate mean cross-validation score as percentage
mean_cv_score_rf = cv_scores_rf.mean() * 100

# Print the scores
print(f"Mean CV Score: {mean_cv_score_rf:.2f}%")
print(f"Best Params: {grid_search_rf.best_params_}")
print(f"Best CV Score: {grid_search_rf.best_score_:.2%}")
```

```
Mean CV Score: 93.37%
Best Params: {'rf__max_depth': None, 'rf__n_estimators': 200}
Best CV Score: 93.40%
```

```

# Fit the Random Forest pipeline on vectorized training data
pipeline_rf.fit(vectorized_training_texts, mapped_training_labels)

# Predict on validation data
y_pred_rf = pipeline_rf.predict(vectorized_validation_texts)

# Generate classification report
classification_rep_rf = classification_report(mapped_validation_labels, y_pred_rf, zero_division=1)
```

```
# Print classification report
print("Classification Report for Random Forest:")
print(classification_rep_rf)
```

```
Classification Report for Random Forest:
              precision    recall  f1-score   support

           0       0.78      0.04      0.08       162
           1       0.43      0.01      0.02       300
           2       0.93      1.00      0.97      6463

    accuracy                           0.93      6925
   macro avg       0.71      0.35      0.36      6925
weighted avg       0.91      0.93      0.90      6925
```

Evaluating the models performances for selection

On a first assessment, we used the validation_labels before mapping them, which resulted in lower performance for all models, as can be seen on the table below.

After mapping the ratings into sentiment, the models performed a lot better.

```
#DO NOT RUN THIS CELL

# This dataframe contains the results from the validation_labels before mapping

results_df
```

	Model	Accuracy	Precision	Recall	F1 Score
0	Naive Bayes	0.690542	0.667260	0.690542	0.570919
1	Logistic Regression	0.714513	0.675614	0.714513	0.654548
2	Support Vector Machine	0.714079	0.683915	0.714079	0.637614
3	Random Forest	0.702816	0.652398	0.702816	0.613475

```
# Initialize dictionary with pipelines
pipelines = {
    'Naive Bayes': pipeline_nb,
    'Logistic Regression': pipeline_lr,
    'Support Vector Machine': pipeline_svm,
    'Random Forest': pipeline_rf
}

# List of pipeline names
pipeline_names = ['Naive Bayes', 'Logistic Regression', 'Support Vector Machine', 'Random Forest']

# List of sentiment classes
class_labels = ['Negative', 'Neutral', 'Positive']

# Initialize lists to store overall metrics
model_names = []
accuracies = []
precisions = []
recalls = []
f1_scores = []

# Initialize dictionaries to store detailed metrics for each sentiment class
detailed_metrics = {label: {'Precision': [], 'Recall': [], 'F1 Score': []} for label in class_labels}

# Evaluate each pipeline
for name in pipeline_names:
    pipeline = pipelines[name]

    # Fit the pipeline on vectorized training data
    pipeline.fit(vectorized_training_texts, mapped_training_labels)

    # Predict on validation data
    y_pred = pipeline.predict(vectorized_validation_texts)

    # Calculate overall metrics
    accuracy = accuracy_score(mapped_validation_labels, y_pred)
    precision = precision_score(mapped_validation_labels, y_pred, average='weighted', zero_division=1)
    recall = recall_score(mapped_validation_labels, y_pred, average='weighted', zero_division=1)
    f1 = f1_score(mapped_validation_labels, y_pred, average='weighted', zero_division=1)

    # Append overall metrics to lists
    model_names.append(name)
    accuracies.append(accuracy)
    precisions.append(precision)
    recalls.append(recall)
    f1_scores.append(f1)

    # Calculate detailed metrics for each sentiment class
    precision_per_class = precision_score(mapped_validation_labels, y_pred, average=None, zero_division=1)
    recall_per_class = recall_score(mapped_validation_labels, y_pred, average=None, zero_division=1)
    f1_per_class = f1_score(mapped_validation_labels, y_pred, average=None, zero_division=1)

    # Append detailed metrics to dictionaries
    for i, label in enumerate(class_labels):
        detailed_metrics[label]['Precision'].append(precision_per_class[i])
        detailed_metrics[label]['Recall'].append(recall_per_class[i])
        detailed_metrics[label]['F1 Score'].append(f1_per_class[i])

# Create a DataFrame to store overall results
results_df_2 = pd.DataFrame({
    'Model': model_names,
    'Accuracy': accuracies,
    'Precision': precisions,
    'Recall': recalls,
    'F1 Score': f1_scores
})

# Create a DataFrame to store detailed metrics
detailed_metrics_df = pd.DataFrame({
    'Model': model_names
})

for label in class_labels:
    detailed_metrics_df[f'{label} Precision'] = detailed_metrics[label]['Precision']
    detailed_metrics_df[f'{label} Recall'] = detailed_metrics[label]['Recall']
    detailed_metrics_df[f'{label} F1 Score'] = detailed_metrics[label]['F1 Score']
```

```
#Print the dataframe with the results for all models for easy comparison

results_df_2
```

	Model	Accuracy	Precision	Recall	F1 Score
0	Naive Bayes	0.933285	0.894541	0.933285	0.901146
1	Logistic Regression	0.934874	0.909851	0.934874	0.906986
2	Support Vector Machine	0.933718	0.916118	0.933718	0.902482
3	Random Forest	0.933574	0.900759	0.933574	0.904673

```
detailed_metrics_df
```

	Model	Negative Precision	Negative Recall	Negative F1 Score	Neutral Precision	Neutral Recall	Neutral F1 Score	Posi Precis
0	Naive Bayes	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.93
1	Logistic Regression	0.555556	0.030864	0.058480	0.521739	0.040000	0.074303	0.93
2	Support Vector	0.666667	0.012346	0.024242	0.666667	0.006667	0.013201	0.93

Taking into consideration the evaluation results for each model, Support Vector Machine (SVM) has distinguished itself due to the following reasons:

- **Accuracy**: SVM has demonstrated competitive accuracy among the models tested, achieving an accuracy of 93.37%. This indicates that it makes correct predictions frequently and is reliable for this classification task.
- **Precision**: Precision measures how many of the predicted positive sentiments are actually correct. SVM shows the highest precision (91.61%), which suggests it effectively identifies positive sentiments without overly predicting false positives. This high precision is crucial for applications where the cost of false positives is high.
- **Recall**: Recall measures how many of the actual positive sentiments were correctly predicted by the model. SVM performs well in recall (93.37%), indicating it captures a high proportion of actual positive sentiments. This is essential for applications where it is critical to identify as many positive cases as possible.
- **F1 Score**: The F1 Score, which balances precision and recall, is also high for SVM (90.25%). This indicates a good overall balance between correctly predicting positive sentiments and avoiding false positives. A high F1 score is beneficial for maintaining a robust performance across varying class distributions.

On the detailed metrics, SVM also distinguishes itself with the best precision scores for all classes: 'Negative', 'Neutral', and 'Positive'.

We will now fit the model and train it with our data, using random parameters and the suggested best parameters:

- 'svm__C': 10
- 'svm__gamma': 0.1
- 'svm__kernel': 'rbf'

Random parameters

```
# Initialize SVM model
svm_model_1 = SVC(kernel='linear', C=1, class_weight='balanced', probability=True)

# Train the model on the vectorized training data
svm_model_1.fit(vectorized_training_texts, mapped_training_labels)

# Predict on the validation data
y_val_pred_1 = svm_model_1.predict(vectorized_validation_texts)

# Calculate evaluation metrics on validation data
accuracy_val_svm_model_1 = accuracy_score(mapped_validation_labels, y_val_pred_1)
precision_val_svm_model_1 = precision_score(mapped_validation_labels, y_val_pred_1, average='weighted')
recall_val_svm_model_1 = recall_score(mapped_validation_labels, y_val_pred_1, average='weighted')
f1_val_svm_model_1 = f1_score(mapped_validation_labels, y_val_pred_1, average='weighted')

print("Support Vector Machine Metrics on Validation Data:")
print(f"  Accuracy: {accuracy_val_svm_model_1 * 100:.2f}%")
print(f"  Precision: {precision_val_svm_model_1 * 100:.2f}%")
print(f"  Recall: {recall_val_svm_model_1 * 100:.2f}%")
print(f"  F1 Score: {f1_val_svm_model_1 * 100:.2f}%")

```

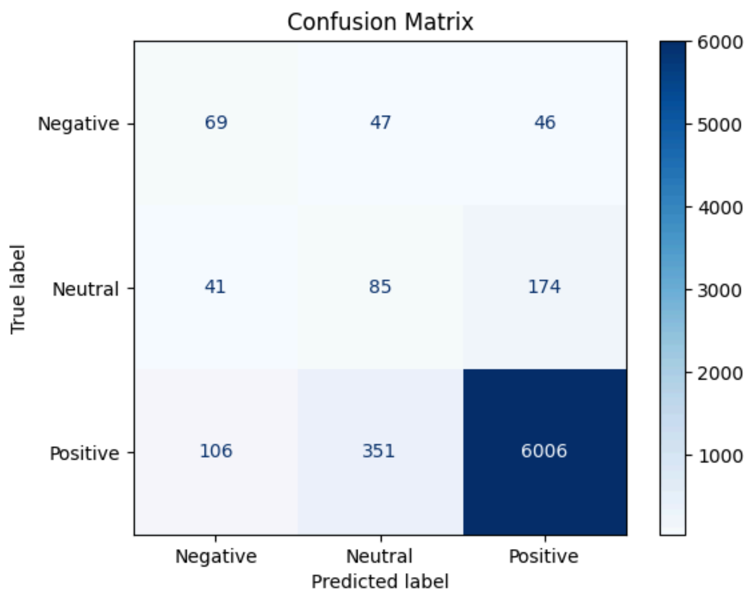
```
Support Vector Machine Metrics on Validation Data:
  Accuracy: 85.13%
  Precision: 91.89%
  Recall: 85.13%
```

```
F1 Score: 88.02%
```

```
# Detailed class-wise metrics
precision_per_class = precision_score(mapped_validation_labels, y_val_pred_1, average=None)
recall_per_class = recall_score(mapped_validation_labels, y_val_pred_1, average=None)
f1_per_class = f1_score(mapped_validation_labels, y_val_pred_1, average=None)

for i, class_label in enumerate(['Negative', 'Neutral', 'Positive']):
    print(f"{class_label}: Precision={precision_per_class[i] * 100:.2f}%, Recall={recall_per_class[i] * 100:.2f}%, F1-sc

```

```
Negative: Precision=24.83%, Recall=46.30%, F1-score=32.33%
Neutral: Precision=15.35%, Recall=38.33%, F1-score=21.93%
Positive: Precision=97.12%, Recall=88.27%, F1-score=92.49%
```

```

# Calculate and display the confusion matrix for validation data
cm = confusion_matrix(mapped_validation_labels, y_val_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Negative', 'Neutral', 'Positive'])
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix")
plt.show()

```



Best Parameters {

'svm__C': 10,

'svm__gamma': 0.1,

'svm__kernel': 'rbf'}

```
# Initialize SVM model with parameters : {'svm__C': , 'svm__gamma': 0.1, 'svm__kernel': 'rbf'}
svm_model_2 = SVC(kernel='rbf', C=10, class_weight='balanced', gamma = 0.1, probability=True)


# Train the model on the vectorized training data
svm_model_2.fit(vectorized_training_texts, mapped_training_labels)

```

Model Evaluation

```
# Predict on the validation data
y_val_pred_2 = svm_model_2.predict(vectorized_validation_texts)

# Calculate evaluation metrics on validation data
accuracy_val_svm_model_2 = accuracy_score(mapped_validation_labels, y_val_pred_2)
precision_val_svm_model_2 = precision_score(mapped_validation_labels, y_val_pred_2, average='weighted')
recall_val_svm_model_2 = recall_score(mapped_validation_labels, y_val_pred_2, average='weighted')
f1_val_svm_model_2 = f1_score(mapped_validation_labels, y_val_pred_2, average='weighted')

print("Support Vector Machine Metrics on Validation Data:")
print(f"  Accuracy: {accuracy_val_svm_model_2 * 100:.2f}%")
print(f"  Precision: {precision_val_svm_model_2 * 100:.2f}%")
print(f"  Recall: {recall_val_svm_model_2 * 100:.2f}%")
print(f"  F1 Score: {f1_val_svm_model_2 * 100:.2f}%")
```

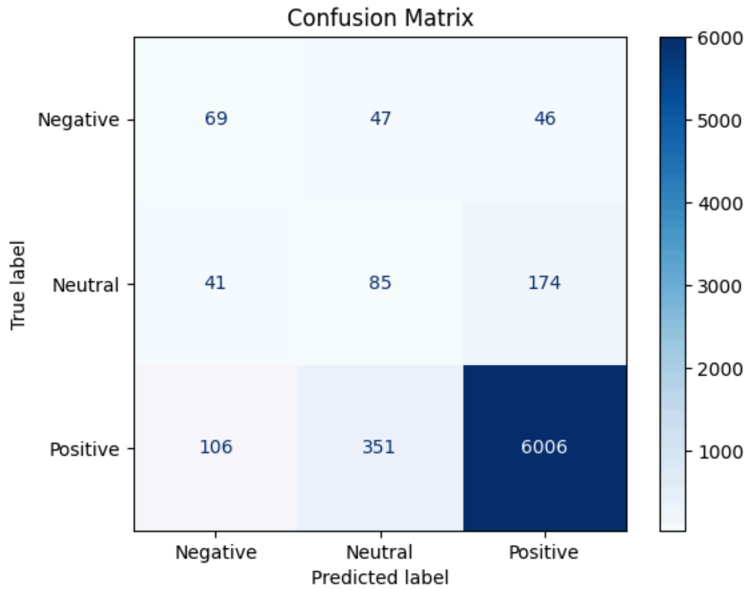
```
Support Vector Machine Metrics on Validation Data:
  Accuracy: 88.95%
  Precision: 91.54%
  Recall: 88.95%
  F1 Score: 90.14%
```

```
# Detailed class-wise metrics
precision_per_class = precision_score(mapped_validation_labels, y_val_pred_2, average=None)
recall_per_class = recall_score(mapped_validation_labels, y_val_pred_2, average=None)
f1_per_class = f1_score(mapped_validation_labels, y_val_pred_2, average=None)

for i, class_label in enumerate(['Negative', 'Neutral', 'Positive']):
    print(f"{class_label}: Precision={precision_per_class[i] * 100:.2f}%, Recall={recall_per_class[i] * 100:.2f}%, F1-sc
```

```
Negative: Precision=31.94%, Recall=42.59%, F1-score=36.51%
Neutral: Precision=17.60%, Recall=28.33%, F1-score=21.71%
Positive: Precision=96.47%, Recall=92.93%, F1-score=94.66%
```

When the model uses the best parameters, as suggested by the GridSearchCV function, we get better results even for the detailed metrics.

```

# Calculate and display the confusion matrix for validation data
cm = confusion_matrix(mapped_validation_labels, y_val_pred_2)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Negative', 'Neutral', 'Positive'])
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix")
plt.show()
```



Assessing the model on the test set

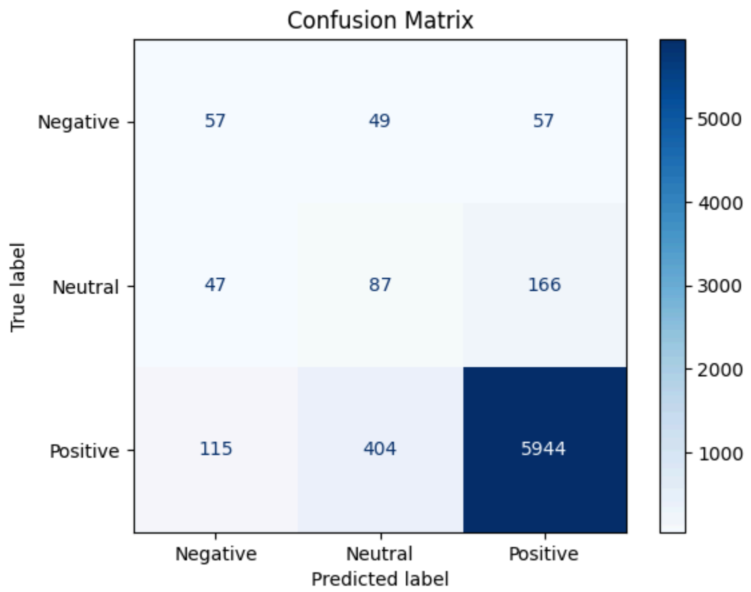
```
# Predict on the test data
y_test_pred_2 = svm_model_2.predict(vectorized_test_texts)

# Calculate evaluation metrics on test data
accuracy_test_2 = accuracy_score(mapped_test_labels, y_test_pred_2)
precision_test_2 = precision_score(mapped_test_labels, y_test_pred_2, average='weighted')
recall_test_2 = recall_score(mapped_test_labels, y_test_pred_2, average='weighted')
f1_test_2 = f1_score(mapped_test_labels, y_test_pred_2, average='weighted')

print("SVM Metrics on Test Data:")
print(f"  Accuracy: {accuracy_test_2}")
print(f"  Precision: {precision_test_2}")
print(f"  Recall: {recall_test_2}")
print(f"  F1 Score: {f1_test_2}")

```

```
SVM Metrics on Test Data:
  Accuracy: 0.8790066416401964
  Precision: 0.9125114885115363
  Recall: 0.8790066416401964
  F1 Score: 0.8943245715675847
```

```
# Detailed class-wise metrics
precision_per_class_2 = precision_score(mapped_validation_labels, y_val_pred_2, average=None)
recall_per_class_2 = recall_score(mapped_validation_labels, y_val_pred_2, average=None)
f1_per_class_2 = f1_score(mapped_validation_labels, y_val_pred_2, average=None)

for i, class_label in enumerate(['Negative', 'Neutral', 'Positive']):
    print(f"{class_label}: Precision={precision_per_class_2[i] * 100:.2f}%, Recall={recall_per_class_2[i] * 100:.2f}%, F

```

```
Negative: Precision=31.94%, Recall=42.59%, F1-score=36.51%
Neutral: Precision=17.60%, Recall=28.33%, F1-score=21.71%
Positive: Precision=96.47%, Recall=92.93%, F1-score=94.66%
```

```
# Calculate and display the confusion matrix for test data
cm_2 = confusion_matrix(mapped_test_labels, y_test_pred_2)
disp = ConfusionMatrixDisplay(confusion_matrix=cm_2, display_labels=['Negative', 'Neutral', 'Positive'])
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix")
plt.show()
```



Testing the model using random reviews

```python
new_reviews = [
    {'review': 'I absolutely love this product! It works wonders.', 'expected_sentiment': 'Positive'},
    {'review': 'The product is okay, not great but not terrible either.', 'expected_sentiment': 'Neutral'},
    {'review': 'I hate this product. It broke after one use.', 'expected_sentiment': 'Negative'},
    {'review': 'This product is quite good. I am satisfied with its performance.', 'expected_sentiment': 'Positive'},
    {'review': 'Terrible experience. The product did not meet my expectations at all.', 'expected_sentiment': 'Negative'
    {'review': 'The product is decent. It gets the job done but there are better options.', 'expected_sentiment': 'Neutr
]

# Iterate over each review
for review_dict in new_reviews:
    review = review_dict['review']
    expected_sentiment = review_dict['expected_sentiment']

    processed_review = preprocess_text(review)
    vectorized_review = tfidf_vectorizer.transform([processed_review])
    predicted_sentiment = svm_model_2.predict(vectorized_review)

    if predicted_sentiment == 0:
        predicted_sentiment = 'Negative'
    elif predicted_sentiment == 1:
        predicted_sentiment = 'Neutral'
    else:
        predicted_sentiment = 'Positive'

    # Print the predicted sentiment and compare with expected
    print(f"Review: '{review}'")
    print(f"Processed Review: '{processed_review}'")
    print(f"Expected Sentiment: {expected_sentiment}")
    print(f"Predicted Sentiment: {predicted_sentiment}")
    print()
```

```
Review: 'I absolutely love this product! It works wonders.'
Processed Review: 'absolutely love product work wonder'
Expected Sentiment: Positive
Predicted Sentiment: Positive

Review: 'The product is okay, not great but not terrible either.'
Processed Review: 'product okay great terrible either'
Expected Sentiment: Neutral
Predicted Sentiment: Negative

Review: 'I hate this product. It broke after one use.'
Processed Review: 'hate product broke one use'
Expected Sentiment: Negative
Predicted Sentiment: Positive

Review: 'This product is quite good. I am satisfied with its performance.'
Processed Review: 'product quite good satisfied performance'
Expected Sentiment: Positive
Predicted Sentiment: Positive

Review: 'Terrible experience. The product did not meet my expectations at all.'
Processed Review: 'terrible experience product meet expectation'
Expected Sentiment: Negative
Predicted Sentiment: Negative

Review: 'The product is decent. It gets the job done but there are better options.'
Processed Review: 'product decent get job done better option'
Expected Sentiment: Neutral
Predicted Sentiment: Neutral
```

Sequence-to-Sequence modeling with LSTM

When deciding between using a Bidirectional LSTM model and a Support Vector Machine (SVM) for sentiment analysis, several factors favor the LSTM approach.

Bidirectional LSTMs excel in understanding the sequential nature of text data, making them highly effective in capturing relationships between words and nuances in language. Unlike SVMs, LSTMs automatically extract relevant features from the data during training, which simplifies the modeling process and eliminates the need for manual feature engineering.

Moreover, LSTMs are flexible in handling inputs of varying lengths, which is crucial for analyzing text where sentence lengths can vary significantly. They perform well on tasks requiring an understanding of complex patterns and non-linear relationships within data, attributes that are particularly valuable in sentiment analysis.

However, it's important to note that LSTMs can be computationally intensive during training and may require careful regularization to prevent overfitting. Additionally, their output may be less interpretable compared to SVMs, which provide clear decision boundaries.

Preprocessing

Reloading data

The initial step involves loading and preprocessing the dataset to ensure data integrity and eliminate redundancy.

```
original_1 = pd.read_csv('/content/drive/MyDrive/Project1/Consumer Reviews of Amazon Products/1429_1.csv',low_memory=Fal
original_2 = pd.read_csv('/content/drive/MyDrive/Project1/Consumer Reviews of Amazon Products/Datafiniti_Amazon_Consumer
```

```
print("file 1 info:")
original_1.info()
print("\nfile 2 info:")
original_2.info()
```

```
file 1 info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 34660 entries, 0 to 34659
Data columns (total 21 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   id                    34660 non-null  object
 1   name                  27900 non-null  object
 2   asins                 34658 non-null  object
 3   brand                 34660 non-null  object
 4   categories            34660 non-null  object
 5   keys                  34660 non-null  object
 6   manufacturer          34660 non-null  object
 7   reviews.date          34621 non-null  object
 8   reviews.dateAdded     24039 non-null  object
 9   reviews.dateSeen      34660 non-null  object
 10  reviews.didPurchase   1 non-null      object
 11  reviews.doRecommend   34066 non-null  object
 12  reviews.id            1 non-null      float64
 13  reviews.numHelpful    34131 non-null  float64
 14  reviews.rating        34627 non-null  float64
 15  reviews.sourceURLs    34660 non-null  object
 16  reviews.text          34659 non-null  object
 17  reviews.title         34654 non-null  object
 18  reviews.userCity      0 non-null      float64
 19  reviews.userProvince  0 non-null      float64
 20  reviews.username      34653 non-null  object
dtypes: float64(5), object(16)
memory usage: 5.6+ MB

file 2 info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 28332 entries, 0 to 28331
Data columns (total 24 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   id                   28332 non-null  object
 1   dateAdded            28332 non-null  object
 2   dateUpdated          28332 non-null  object
 3   name                 28332 non-null  object
 4   asins                28332 non-null  object
 5   brand                28332 non-null  object
 6   categories           28332 non-null  object
 7   primaryCategories    28332 non-null  object
 8   imageURLs            28332 non-null  object
 9   keys                 28332 non-null  object
 10  manufacturer         28332 non-null  object
 11  manufacturerNumber   28332 non-null  object
 12  reviews.date         28332 non-null  object
 13  reviews.dateSeen     28332 non-null  object
 14  reviews.didPurchase  9 non-null      object
 15  reviews.doRecommend  16086 non-null  object
 16  reviews.id           41 non-null     float64
 17  reviews.numHelpful   16115 non-null  float64
 18  reviews.rating       28332 non-null  int64
 19  reviews.sourceURLs   28332 non-null  object
 20  reviews.text         28332 non-null  object
 21  reviews.title        28332 non-null  object
```

We ensure the dataset cleanliness by removing duplicate reviews, guaranteing that each review is unique and contributes distinct information to the analysis.

```
# Find duplicates based on review.text
duplicate_texts = original_1['reviews.text'].unique()
original_2 = original_2[~original_2['reviews.text'].isin(duplicate_texts)]

# Check the result
original_2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 14027 entries, 0 to 28249
Data columns (total 24 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   id                   14027 non-null  object
 1   dateAdded            14027 non-null  object
 2   dateUpdated          14027 non-null  object
 3   name                 14027 non-null  object
 4   asins                14027 non-null  object
 5   brand                14027 non-null  object
 6   categories           14027 non-null  object
 7   primaryCategories    14027 non-null  object
 8   imageURLs            14027 non-null  object
 9   keys                 14027 non-null  object
 10  manufacturer         14027 non-null  object
 11  manufacturerNumber   14027 non-null  object
 12  reviews.date         14027 non-null  object
 13  reviews.dateSeen     14027 non-null  object
 14  reviews.didPurchase  9 non-null      object
 15  reviews.doRecommend  2040 non-null   object
 16  reviews.id           41 non-null     float64
 17  reviews.numHelpful   2063 non-null   float64
 18  reviews.rating       14027 non-null  int64
 19  reviews.sourceURLs   14027 non-null  object
 20  reviews.text         14027 non-null  object
 21  reviews.title        14027 non-null  object
 22  reviews.username     14027 non-null  object
 23  sourceURLs           14027 non-null  object
dtypes: float64(2), int64(1), object(21)
memory usage: 2.7+ MB
```

To address potential biases introduced by imbalanced review ratings, specifically an excess of 5-star ratings, a stratified approach is adopted to maintain dataset integrity.

It ensures a more balanced representation of reviews across different rating categories, mitigating potential biases towards higher ratings.

```
# reducing size of 5 star reviews by half
original_2 = original_2.drop(original_2[original_2['reviews.rating'] == 5].sample(frac=0.5, random_state=1).index)
original_2['reviews.rating'].value_counts()
```

```
reviews.rating
5    5161
4    1875
1     781
3     623
2     425
Name: count, dtype: int64
```

```
original_1[['id', 'name', 'categories', 'reviews.date', 'reviews.rating', 'reviews.text', 'reviews.title']].copy()
```

	id	name	categories	reviews.date	reviews.rating	reviews.text	reviews.title
0	AVqkIhwDv8e3D1O-lebb	All-New Fire HD 8 Tablet, 8 HD Display, Wi-Fi,...	Electronics,iPad & Tablets,All Tablets,Fire Ta...	2017-01-13T00:00:00.000Z	5.0	This product so far has not disappointed. My c...	Kindle
1	AVqkIhwDv8e3D1O-lebb	All-New Fire HD 8 Tablet, 8 HD Display, Wi-Fi,...	Electronics,iPad & Tablets,All Tablets,Fire Ta...	2017-01-13T00:00:00.000Z	5.0	great for beginner or experienced person. Boug...	very fast
2	AVqkIhwDv8e3D1O-lebb	All-New Fire HD 8 Tablet, 8 HD Display, Wi-Fi,...	Electronics,iPad & Tablets,All Tablets,Fire Ta...	2017-01-13T00:00:00.000Z	5.0	Inexpensive tablet for him to use and learn on...	Beginner tablet for our 9 year old son.
3	AVqkIhwDv8e3D1O-lebb	All-New Fire HD 8 Tablet, 8 HD Display, Wi-Fi,...	Electronics,iPad & Tablets,All Tablets,Fire Ta...	2017-01-13T00:00:00.000Z	4.0	I've had my Fire HD 8 two weeks now and I love...	Good!!!
4	AVqkIhwDv8e3D1O-lebb	All-New Fire HD 8 Tablet, 8 HD Display, Wi-Fi,...	Electronics,iPad & Tablets,All Tablets,Fire Ta...	2017-01-12T00:00:00.000Z	5.0	I bought this for my grand daughter when she c...	Fantastic Tablet for kids
...	...	...	...	...	...	...	...
	[illegible]		Computers/Tablets &	[illegible]		This is not	Not appreciably

Processing

We proceed with mapping and categorizing the dataset to enhance clarity and facilitate analysis based on product categories. It helps on a focused analysis within defined product categories while maintaining dataset integrity.

```
file_1 = original_1[['id', 'name', 'categories', 'reviews.date', 'reviews.rating', 'reviews.text', 'reviews.title']].cop
file_1 = original_1.dropna(subset=['reviews.rating', 'reviews.text', 'reviews.date'])   # dropping rows with no review t
file_1['reviews.title'] = file_1['reviews.title'].fillna(" ")
file_1['name'] = file_1['name'].fillna("NA")    # replace missing product names with "NA" > used in mapping below
```

```
<ipython-input-34-c5676d665c6f>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
  file_1['reviews.title'] = file_1['reviews.title'].fillna(" ")
<ipython-input-34-c5676d665c6f>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
  file_1['name'] = file_1['name'].fillna("NA")    # replace missing product names with "NA" > used in mapping below
```

Defining the mapping dicts

```python
## Defining the mapping dicts

name_categories = {
    'All-New Fire HD 8 Tablet, 8 HD Display, Wi-Fi, 16 GB - Includes Special Offers, Magenta': 'Tablets',
    'Kindle Oasis E-reader with Leather Charging Cover - Merlot, 6 High-Resolution Display (300 ppi), Wi-Fi - Includes S
    'Amazon Kindle Lighted Leather Cover,,,\r\nAmazon Kindle Lighted Leather Cover,,,': 'Accessories',
    'Amazon Kindle Lighted Leather Cover,,,\r\nKindle Keyboard,,,': 'Accessories',
    'Kindle Keyboard,,,\r\nKindle Keyboard,,,': 'Accessories',
    'All-New Fire HD 8 Tablet, 8 HD Display, Wi-Fi, 32 GB - Includes Special Offers, Magenta': 'Tablets',
    'Fire HD 8 Tablet with Alexa, 8 HD Display, 32 GB, Tangerine - with Special Offers,': 'Tablets',
    'Amazon 5W USB Official OEM Charger and Power Adapter for Fire Tablets and Kindle eReaders,,,\r\nAmazon 5W USB Offic
    'All-New Kindle E-reader - Black, 6 Glare-Free Touchscreen Display, Wi-Fi - Includes Special Offers,,': 'E-readers',
    'Amazon Kindle Fire Hd (3rd Generation) 8gb,,,\r\nAmazon Kindle Fire Hd (3rd Generation) 8gb,,,': 'E-readers',
    'Fire Tablet, 7 Display, Wi-Fi, 8 GB - Includes Special Offers, Magenta': 'Tablets',
    'Kindle Oasis E-reader with Leather Charging Cover - Black, 6 High-Resolution Display (300 ppi), Wi-Fi - Includes Sp
    'Amazon - Kindle Voyage - 4GB - Wi-Fi + 3G - Black,,,\r\nAmazon - Kindle Voyage - 4GB - Wi-Fi + 3G - Black,,,': 'E-r
    'Amazon - Kindle Voyage - 4GB - Wi-Fi + 3G - Black,,,\r\nFire HD 8 Tablet with Alexa, 8 HD Display, 16 GB, Tangerine
    'Fire HD 8 Tablet with Alexa, 8 HD Display, 16 GB, Tangerine - with Special Offers,': 'Tablets',
    'Amazon Standing Protective Case for Fire HD 6 (4th Generation) - Black,,,\r\nAmazon Standing Protective Case for Fi
    'Certified Refurbished Amazon Fire TV (Previous Generation - 1st),,,\r\nCertified Refurbished Amazon Fire TV (Previo
    'Brand New Amazon Kindle Fire 16gb 7 Ips Display Tablet Wifi 16 Gb Blue,,,': 'E-readers',
    'Amazon Kindle Touch Leather Case (4th Generation - 2011 Release), Olive Green,,,\r\nAmazon Kindle Touch Leather Cas
    'Fire Kids Edition Tablet, 7 Display, Wi-Fi, 16 GB, Green Kid-Proof Case': 'Tablets',
    'Amazon Kindle Paperwhite - eBook reader - 4 GB - 6 monochrome Paperwhite - touchscreen - Wi-Fi - black,,,': 'E-read
    'Kindle Voyage E-reader, 6 High-Resolution Display (300 ppi) with Adaptive Built-in Light, PagePress Sensors, Wi-Fi
    'Certified Refurbished Amazon Fire TV Stick (Previous Generation - 1st),,,\r\nCertified Refurbished Amazon Fire TV S
    'Certified Refurbished Amazon Fire TV Stick (Previous Generation - 1st),,,\r\nKindle Paperwhite,,,': 'Streaming Devi
    'Kindle Paperwhite,,,\r\nKindle Paperwhite,,,': 'E-readers',
    'Amazon Fire Kids Edition Tablet, 7 Display, Wi-Fi, 16 GB, Blue Kid-Proof Case - Blue': 'Tablets',
    'Kindle Paperwhite E-reader - White, 6 High-Resolution Display (300 ppi) with Built-in Light, Wi-Fi - Includes Speci
    'Amazon Echo and Fire TV Power Adapter,,,\r\nAmazon Echo and Fire TV Power Adapter,,,': 'Accessories',
    'Amazon Fire Hd 8 8in Tablet 16gb Black B018szt3bk 6th Gen (2016) Android,,,\r\nAmazon Fire Hd 8 8in Tablet 16gb Bla
    'Certified Refurbished Amazon Fire TV with Alexa Voice Remote,,,\r\nCertified Refurbished Amazon Fire TV with Alexa
    'Amazon - Fire 16GB (5th Gen, 2015 Release) - Black,,,\r\nAmazon - Fire 16GB (5th Gen, 2015 Release) - Black,,,': 'T
    'Fire Tablet, 7 Display, Wi-Fi, 8 GB - Includes Special Offers, Black': 'Tablets',
    'Echo (White),,,\r\nEcho (White),,,': 'Streaming Devices & Smart Speakers',
    'Echo (White),,,\r\nFire Tablet, 7 Display, Wi-Fi, 8 GB - Includes Special Offers, Tangerine"': 'Streaming Devices &
    'Echo (Black),,,\r\nEcho (Black),,,': 'Streaming Devices & Smart Speakers',
    'Echo (Black),,,\r\nAmazon 9W PowerFast Official OEM USB Charger and Power Adapter for Fire Tablets and Kindle eRead
    'Amazon 9W PowerFast Official OEM USB Charger and Power Adapter for Fire Tablets and Kindle eReaders,,,\r\nAmazon 9W
    'Amazon Fire Hd 6 Standing Protective Case(4th Generation - 2014 Release), Cayenne Red,,,\r\nAmazon Fire Hd 6 Standi
    'Amazon Fire Hd 6 Standing Protective Case(4th Generation - 2014 Release), Cayenne Red,,,\r\nAmazon 5W USB Official
    'Amazon Fire Hd 10 Tablet, Wi-Fi, 16 Gb, Special Offers - Silver Aluminum,,,\r\nAmazon Fire Hd 10 Tablet, Wi-Fi, 16
    'Amazon - Amazon Tap Portable Bluetooth and Wi-Fi Speaker - Black,,,\r\nAmazon - Amazon Tap Portable Bluetooth and W
    'Coconut Water Red Tea 16.5 Oz (pack of 12),,,\r\nAmazon Fire Tv,,,': 'Streaming Devices & Smart Speakers',
    'Amazon Fire Tv,,,\r\nAmazon Fire Tv,,,': 'Streaming Devices & Smart Speakers',
    'Amazon Fire Tv,,,\r\nKindle Dx Leather Cover, Black (fits 9.7 Display, Latest and 2nd Generation Kindle Dxs)",,': '
    'Kindle Dx Leather Cover, Black (fits 9.7 Display, Latest and 2nd Generation Kindle Dxs),,': 'Accessories',
    'Amazon Kindle Fire 5ft USB to Micro-USB Cable (works with most Micro-USB Tablets),,,\r\nAmazon Kindle Fire 5ft USB
    'New Amazon Kindle Fire Hd 9w Powerfast Adapter Charger + Micro Usb Angle Cable,,,\r\nNew Amazon Kindle Fire Hd 9w P
    'New Amazon Kindle Fire Hd 9w Powerfast Adapter Charger + Micro Usb Angle Cable,,,\r\n': 'Accessories',
    'NA': 'Uncategorized'   # for products with no names
}



na_categories = {
    'Stereos,Remote Controls,Amazon Echo,Audio Docks & Mini Speakers,Amazon Echo Accessories,Kitchen & Dining Features,S
    'Fire Tablets,Tablets,Computers & Tablets,All Tablets,Frys': 'Tablets',
    'TVs Entertainment,Wireless Speakers,Virtual Assistant Speakers,Featured Brands,Electronics,Amazon Devices,Home,Home
    'Chargers & Adapters,Computers & Accessories,Tablet & E-Reader Accessories,Amazon Devices & Accessories,Fire Tablet
    'Cases,Kindle Store,Amazon Device Accessories,Accessories,Tablet Accessories': 'Accessories',
    'Electronics,eBook Readers & Accessories,Power Adapters,Computers/Tablets & Networking,Tablet & eBook Reader Accs,Ch
    'Electronics,Tablets & E-Readers,Tablets,Back To College,College Electronics,College Ipads & Tablets,Featured Brands
    'Featured Brands,Electronics,Amazon Devices,Home,Home Improvement,Home Safety & Security,Home Security,Alarms & Sens
    'Rice Dishes,Ready Meals,Beauty,Moisturizers,Lotions': 'Food & Beverages',
    'Back To College,College Electronics,College Tvs & Home Theater,Electronics,Tvs & Home Theater,Streaming Devices,Fea
    'Electronics,Amazon Device Accessories,Kindle Store,Covers,Kindle E-Reader Accessories,Kindle DX (2nd Generation, Gl
    'Power Adapters & Cables,Electronics,USB Cables': 'Accessories',
    'Computers/Tablets & Networking,Tablet & eBook Reader Accs,Chargers & Sync Cables,Power Adapters & Cables,Kindle Sto
    'Amazon Devices & Accessories,Amazon Device Accessories,Power Adapters & Cables,Kindle Store,Kindle E-Reader Accesso
}

```

Mapping the categories

```python
## Mapping
file_1['primaryCategories'] = file_1['name'].map(name_categories).fillna('Uncategorized')   # 1st mapping: setting prima
uncategorized_mask = file_1['primaryCategories'] == 'Uncategorized'   # defining mask for second mapping
file_1.loc[uncategorized_mask, 'primaryCategories'] = file_1.loc[uncategorized_mask, 'categories'].map(na_categories) #
```

```
<ipython-input-36-4acd269842ff>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
  file_1['primaryCategories'] = file_1['name'].map(name_categories).fillna('Uncategorized')   # 1st mapping: setting pr:
```

```
1 file_1 = file_1[file_1['primaryCategories'] != 'Food & Beverages'] # dropping this single review on a drink
2 file_1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 34597 entries, 0 to 34659
Data columns (total 22 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   id                    34597 non-null  object
 1   name                  34597 non-null  object
 2   asins                 34595 non-null  object
 3   brand                 34597 non-null  object
 4   categories            34597 non-null  object
 5   keys                  34597 non-null  object
 6   manufacturer          34597 non-null  object
 7   reviews.date          34597 non-null  object
 8   reviews.dateAdded     24037 non-null  object
 9   reviews.dateSeen      34597 non-null  object
 10  reviews.didPurchase   1 non-null      object
 11  reviews.doRecommend   34066 non-null  object
 12  reviews.id            1 non-null      float64
 13  reviews.numHelpful    34118 non-null  float64
 14  reviews.rating        34597 non-null  float64
 15  reviews.sourceURLs    34597 non-null  object
 16  reviews.text          34597 non-null  object
 17  reviews.title         34597 non-null  object
 18  reviews.userCity      0 non-null      float64
 19  reviews.userProvince  0 non-null      float64
 20  reviews.username      34590 non-null  object
 21  primaryCategories     34597 non-null  object
dtypes: float64(5), object(17)
memory usage: 6.1+ MB
```

```
1 file_1['primaryCategories'].value_counts()
```

```
primaryCategories
Tablets                             16510
Streaming Devices & Smart Speakers  12533
E-readers                            4928
Accessories                           626
Name: count, dtype: int64
```

```
1 df = file_1[['id', 'name', 'categories', 'reviews.date', 'reviews.rating', 'reviews.text', 'reviews.title', 'primaryCate
2 df.columns
```

```
Index(['id', 'name', 'categories', 'reviews.date', 'reviews.rating',
       'reviews.text', 'reviews.title', 'primaryCategories'],
      dtype='object')
```

Text Cleaning

The dataset undergoes rigorous text preprocessing to enhance the quality and relevance of textual data for sentiment analysis.

These functions clean the text by removing special characters, stopwords, and performing lemmatization to reduce words to their base form, ensuring standardized text inputs for the sentiment analysis model

```
1 df.dropna(subset=['reviews.text'],inplace=True)
2 print(len(df))
```

```
34597
```

```
def clean_text(text):
    # Handle non-string types
    if not isinstance(text, str):
        return ""  # Or handle it differently based on your needs

    # Remove special characters and punctuation
    text = re.sub(r'[^A-Za-z0-9\s]', '', text)
    # Remove unnecessary whitespace
    text = re.sub(r'\s+', ' ', text).strip()
    # Convert text to lowercase
    text = text.lower()
    # Remove stop words
    stop_words = set(stopwords.words('english'))
    text = ' '.join(word for word in text.split() if word not in stop_words)
    return text

def lemmatize_text(text):
    lemmatizer = WordNetLemmatizer()
    tokens = word_tokenize(text)
    lemmatized_tokens = [lemmatizer.lemmatize(token) for token in tokens]
    return ' '.join(lemmatized_tokens)

def preprocess_text(text):
    cleaned_text = clean_text(text)
    lemmatized_text = lemmatize_text(cleaned_text)
    return lemmatized_text
```

Labeling Sentiments Based on Ratings

This step categorizes reviews into negative, neutral, and positive sentiments based on predefined rating thresholds, aiding in the interpretation and evaluation of model predictions.

```
# Creating sentiment labels based on rating
def get_sentiment(rating):
    if rating <= 2:
        return 0  # Negative
    elif rating == 3:
        return 1  # Neutral
    else:
        return 2  # Positive

df['sentiment'] = df['reviews.rating'].apply(get_sentiment)
df['processed_text'] = df['reviews.text'].apply(preprocess_text)
df.head()
```

	id	name	categories	reviews.date	reviews.rating	reviews.text	reviews.title	primaryCategories
0	AVqkIhwDv8e3D1O-lebb	All-New Fire HD 8 Tablet, 8 HD Display, Wi-Fi,...	Electronics,iPad & Tablets,All Tablets,Fire Ta...	2017-01-13T00:00:00.000Z	5.0	This product so far has not disappointed. My c...	Kindle	Tablets
1	AVqkIhwDv8e3D1O-lebb	All-New Fire HD 8 Tablet, 8 HD Display, Wi-Fi,...	Electronics,iPad & Tablets,All Tablets,Fire Ta...	2017-01-13T00:00:00.000Z	5.0	great for beginner or experienced person. Boug...	very fast	Tablets
2	AVqkIhwDv8e3D1O-lebb	All-New Fire HD 8 Tablet, 8 HD Display, Wi-Fi,...	Electronics,iPad & Tablets,All Tablets,Fire Ta...	2017-01-13T00:00:00.000Z	5.0	Inexpensive tablet for him to use and learn on...	Beginner tablet for our 9 year old son.	Tablets
3	AVqkIhwDv8e3D1O-lebb	All-New Fire HD 8 Tablet, 8 HD Display,	Electronics,iPad & Tablets,All Tablets,Fire Ta...	2017-01-13T00:00:00.000Z	4.0	I've had my Fire HD 8 two weeks now and I love...	Good!!!	Tablets

```
sentiment_counts = df['sentiment'].value_counts()
print(sentiment_counts)
```

```
sentiment
2    32286
1     1499
0      812
Name: count, dtype: int64
```

Split into training and validation set

Finally, the dataset is prepared for model training and evaluation by selecting relevant columns and dropping entries with missing review texts.

This ensures that only complete and necessary data points are retained for subsequent analysis, ensuring a more robust model training and evaluation.

To address class imbalance and prepare for model training, oversampling using SMOTE (Synthetic Minority Over-sampling Technique) is applied. SMOTE is used to balance the distribution of sentiment labels in the dataset, enhancing the model's ability to generalize across different sentiment categories.

```
# Separate features and target
X = df['reviews.text']
y = df['sentiment']

# Tokenizing and padding the sequences
tokenizer = Tokenizer(oov_token='<OOV>')
tokenizer.fit_on_texts(X)
vocab_size = len(tokenizer.word_index) + 1
sequences = tokenizer.texts_to_sequences(X)
padded_sequences = pad_sequences(sequences, maxlen=200, padding='post', truncating='post')

# Apply SMOTE (oversampling)
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(padded_sequences, y)

# # Apply RandomUnderSampler (optional, if you want to combine both)
# rus = RandomUnderSampler(random_state=42)
# X_resampled, y_resampled = rus.fit_resample(X_resampled, y_resampled)

# Convert sentiment labels to categorical format
sentiment_labels = to_categorical(y_resampled)

# Split data into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X_resampled, sentiment_labels, test_size=0.2, random_state=42)

# Calculate class weights
y_train_integers = np.argmax(y_train, axis=1)
class_weights = compute_class_weight(class_weight='balanced', classes=np.unique(y_train_integers), y=y_train_integers)
class_weights_dict = dict(enumerate(class_weights))
```

Test Set

The test set also undergoes similar preprocessing steps as the training data to ensure consistency and compatibility for sentiment analysis. The dataset is cleaned by removing rows with missing values to maintain data integrity and ensure accurate sentiment analysis results.

```
test_df = original_2[['id', 'name', 'categories', 'reviews.date', 'reviews.rating', 'reviews.text', 'reviews.title', 'pr
test_df.head()
```

Show hidden output

Labeling Sentiments Based on Ratings

Sentiment labels are assigned to each review in test_df based on its rating using predefined thresholds.

```
test_df.dropna(inplace=True)
print(len(test_df))

# Creating sentiment labels based on rating
def get_sentiment(rating):
    if rating <= 2:
        return 0  # Negative
    elif rating == 3:
        return 1  # Neutral
    else:
        return 2  # Positive

test_df['sentiment'] = test_df['reviews.rating'].apply(get_sentiment)
test_df['processed_text'] = test_df['reviews.text'].apply(preprocess_text)
```

```
8865
```

The text data in test_df is preprocessed using the same functions used during training to maintain consistency and ensure the input format required by the sentiment analysis model.

The text sequences in test_df are then tokenized and padded to a fixed length to match the input requirements of the LSTM model trained on the training data.

```
# Tokenizing and padding the sequences
test_sequences = tokenizer.texts_to_sequences(test_df['reviews.text'])
X_test = pad_sequences(test_sequences, maxlen=200, padding='post', truncating='post')
y_test = to_categorical(test_df['sentiment'])


```

Count of total Sentiments per type

```
sentiment_counts = test_df['sentiment'].value_counts()
print(sentiment_counts)
```

```
sentiment
2    7036
0    1206
1     623
Name: count, dtype: int64
```

LSTM Model

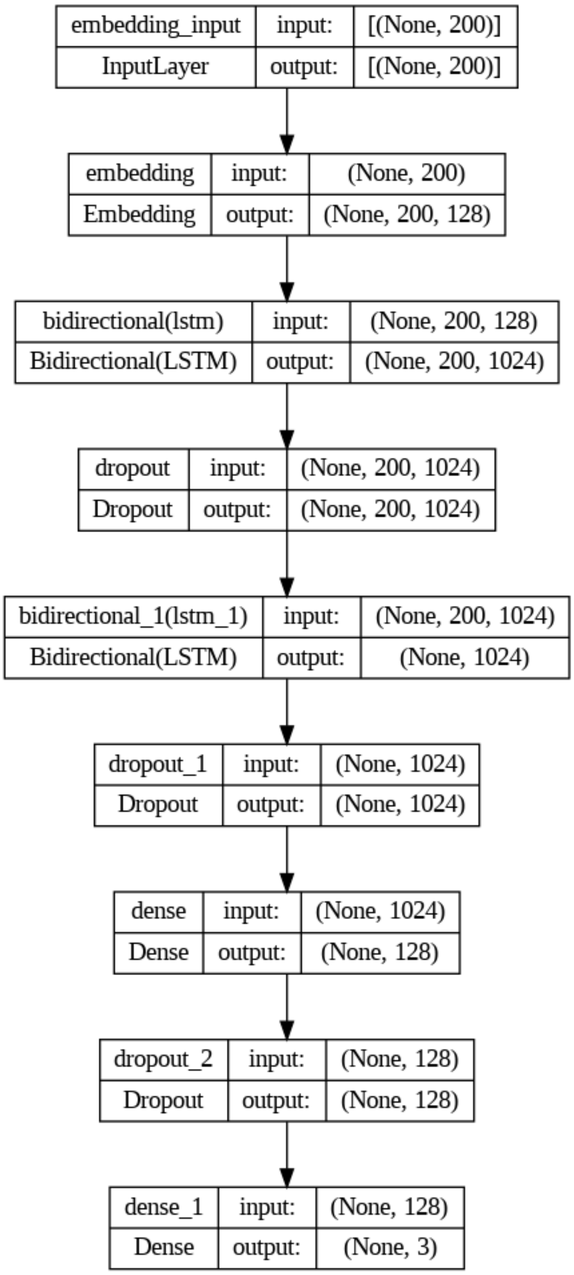
Model Creation

The LSTM model (model_4) is designed with multiple layers to capture sequential dependencies in text data efficiently.

```
model_4 = Sequential([
    Embedding(input_dim=vocab_size, output_dim=128, input_length=200),
    Bidirectional(LSTM(512, return_sequences=True)),
    Dropout(0.5),
    Bidirectional(LSTM(512)),
    Dropout(0.5),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(3, activation='softmax')
])


model_4.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy',
                       tf.keras.metrics.Precision(name='precision'),
                       tf.keras.metrics.Recall(name='recall')])
model_4.summary()

```

Show hidden output

```
from tensorflow.keras.utils import plot_model

plot_model(model_4, to_file='LSTM_model.png', show_shapes=True, show_layer_names=True)
```



The model consists of multiple layers including an Embedding layer, two Bidirectional LSTM layers, Dropout layers to prevent overfitting, and Dense layers for classification. The total trainable parameters are substantial, since it is a complex model.

- Embedding Layer: Maps each word to a vector space of specified dimensions (vocab_size and 128 in this case).
- Bidirectional LSTM Layers: Utilizes two LSTM layers to capture both past and future context of sequences (512 units each), enhancing model performance in understanding text sequences.
- Dropout Layers: Applied to mitigate overfitting by randomly setting a fraction of input units to zero during training (0.5 dropout rate).
- Dense Layers: Two fully connected layers (128 units, 3 output units for sentiment classes) with ReLU and softmax activations respectively for classification.

Training

The model was trained for 23 epochs with a batch size of 64. Early stopping was employed with a patience of 3 to monitor validation loss and restore the best weights.

```
early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True, mode='min')
history = model_4.fit(X_train, y_train, epochs=23, batch_size=64, validation_data=(X_val, y_val), callbacks=[early_stopp
```

```
Epoch 1/23
1211/1211 [==============================] - 94s 73ms/step - loss: 0.7758 - accuracy: 0.5844 - precision: 0.6576 - reca
Epoch 2/23
1211/1211 [==============================] - 62s 51ms/step - loss: 0.9101 - accuracy: 0.5267 - precision: 0.6387 - reca
Epoch 3/23
1211/1211 [==============================] - 61s 50ms/step - loss: 0.7212 - accuracy: 0.6009 - precision: 0.6621 - reca
Epoch 4/23
1211/1211 [==============================] - 60s 50ms/step - loss: 0.6680 - accuracy: 0.6271 - precision: 0.6676 - reca
```

```
Epoch 5/23
1211/1211 [==============================] - 60s 49ms/step - loss: 0.6232 - accuracy: 0.6428 - precision: 0.6704 - reca
Epoch 6/23
1211/1211 [==============================] - 59s 49ms/step - loss: 0.6027 - accuracy: 0.6499 - precision: 0.6696 - reca
Epoch 7/23
1211/1211 [==============================] - 59s 49ms/step - loss: 0.5924 - accuracy: 0.6570 - precision: 0.6747 - reca
Epoch 8/23
1211/1211 [==============================] - 59s 49ms/step - loss: 0.5631 - accuracy: 0.6708 - precision: 0.6828 - reca
Epoch 9/23
1211/1211 [==============================] - 59s 49ms/step - loss: 0.5388 - accuracy: 0.6891 - precision: 0.6969 - reca
Epoch 10/23
1211/1211 [==============================] - 59s 49ms/step - loss: 0.5172 - accuracy: 0.7094 - precision: 0.7147 - reca
Epoch 11/23
1211/1211 [==============================] - 59s 49ms/step - loss: 0.5282 - accuracy: 0.7133 - precision: 0.7249 - reca
Epoch 12/23
1211/1211 [==============================] - 59s 49ms/step - loss: 0.5417 - accuracy: 0.7085 - precision: 0.7205 - reca
Epoch 13/23
1211/1211 [==============================] - 59s 49ms/step - loss: 0.4972 - accuracy: 0.7351 - precision: 0.7421 - reca
Epoch 14/23
1211/1211 [==============================] - 59s 49ms/step - loss: 0.4707 - accuracy: 0.7573 - precision: 0.7618 - reca
Epoch 15/23
1211/1211 [==============================] - 59s 49ms/step - loss: 0.4499 - accuracy: 0.7726 - precision: 0.7760 - reca
Epoch 16/23
1211/1211 [==============================] - 59s 49ms/step - loss: 0.4323 - accuracy: 0.7849 - precision: 0.7875 - reca
```

Evaluation and Results

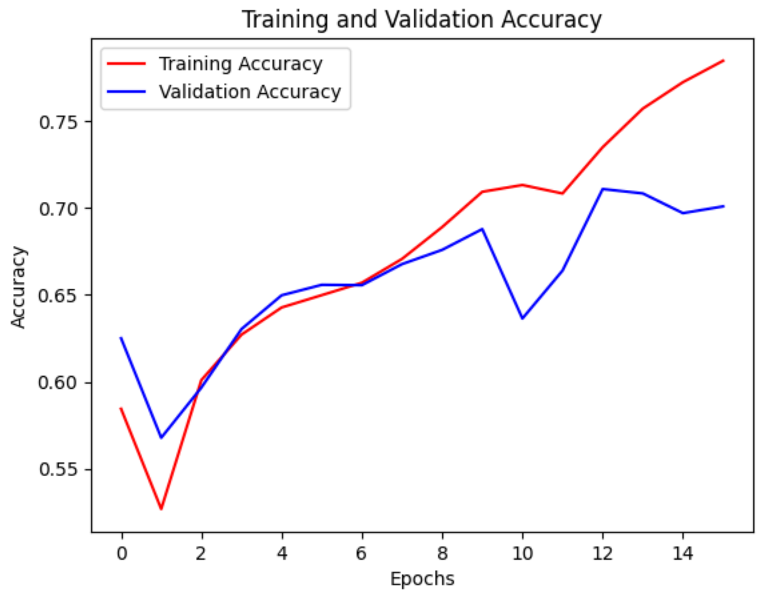
```
evaluation = model_4.evaluate(X_val, y_val)

# Print the results
print(f'Validation Loss: {evaluation[0]}')
print(f'Validation Accuracy: {evaluation[1]}')
print(f'Validation Precision: {evaluation[2]}')
print(f'Validation Recall: {evaluation[3]}')

```

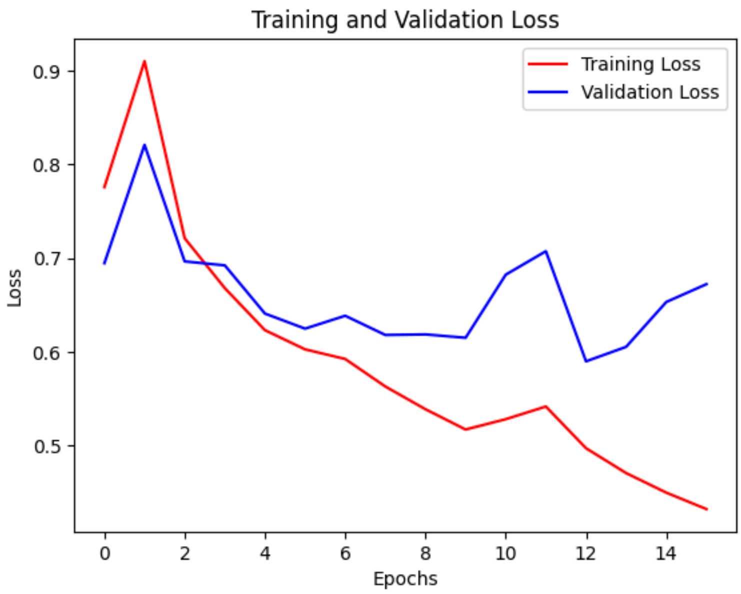
```
606/606 [==============================] - 12s 19ms/step - loss: 0.5899 - accuracy: 0.7110 - precision: 0.7235 - recall:
Validation Loss: 0.5899065136909485
Validation Accuracy: 0.7110262513160706
Validation Precision: 0.7235174179077148
Validation Recall: 0.6952818632125854
```

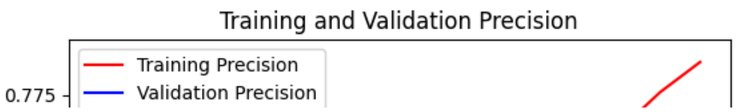
Accuracy and Loss: The model achieves around 71.1% validation accuracy, with fluctuations indicating potential overfitting beyond certain epochs.

Precision and Recall: Precision (72.4%) and recall (69.5%) metrics indicate decent performance across all classes, suggesting balanced performance in identifying both positive and negative sentiments.

```
# Retrieve history
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
precision = history.history['precision']
val_precision = history.history['val_precision']
recall = history.history['recall']
val_recall = history.history['val_recall']

epochs = range(len(acc))

# Plot training and validation accuracy
plt.plot(epochs, acc, 'r', label='Training Accuracy')
plt.plot(epochs, val_acc, 'b', label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Plot training and validation loss
plt.figure()
plt.plot(epochs, loss, 'r', label='Training Loss')
plt.plot(epochs, val_loss, 'b', label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

# Plot training and validation precision
plt.figure()
plt.plot(epochs, precision, 'r', label='Training Precision')
plt.plot(epochs, val_precision, 'b', label='Validation Precision')
plt.title('Training and Validation Precision')
plt.xlabel('Epochs')
plt.ylabel('Precision')
plt.legend()
plt.show()

# Plot training and validation recall
plt.figure()
plt.plot(epochs, recall, 'r', label='Training Recall')
plt.plot(epochs, val_recall, 'b', label='Validation Recall')
plt.title('Training and Validation Recall')
plt.xlabel('Epochs')
plt.ylabel('Recall')
plt.legend()
plt.show()
```







Predicting on Test data

```
# Make predictions on the test data
y_pred_prob = model_4.predict(X_test)
y_pred = np.argmax(y_pred_prob, axis=1)
y_true = np.argmax(y_test, axis=1)

# Calculate accuracy, precision, recall, and F1-score
accuracy = accuracy_score(y_true, y_pred)
precision = precision_score(y_true, y_pred, average='weighted')
recall = recall_score(y_true, y_pred, average='weighted')
f1 = f1_score(y_true, y_pred, average='weighted')

print(f'Accuracy: {accuracy}')
print(f'Precision: {precision}')
print(f'Recall: {recall}')
print(f'F1-score: {f1}')
```

```
278/278 [==============================] - 6s 17ms/step
Accuracy: 0.7130287648054145
Precision: 0.7612363952872192
Recall: 0.7130287648054145
F1-score: 0.7342564335147488
```

The achieved metrics indicate that the LSTM model performs reasonably well in classifying sentiment from textual reviews. The precision and recall scores are fairly balanced, suggesting that the model maintains consistency in correctly predicting both positive and negative sentiments.

However, there might be room for improvement, particularly in handling more nuanced cases or fine-tuning the model architecture and hyperparameters.

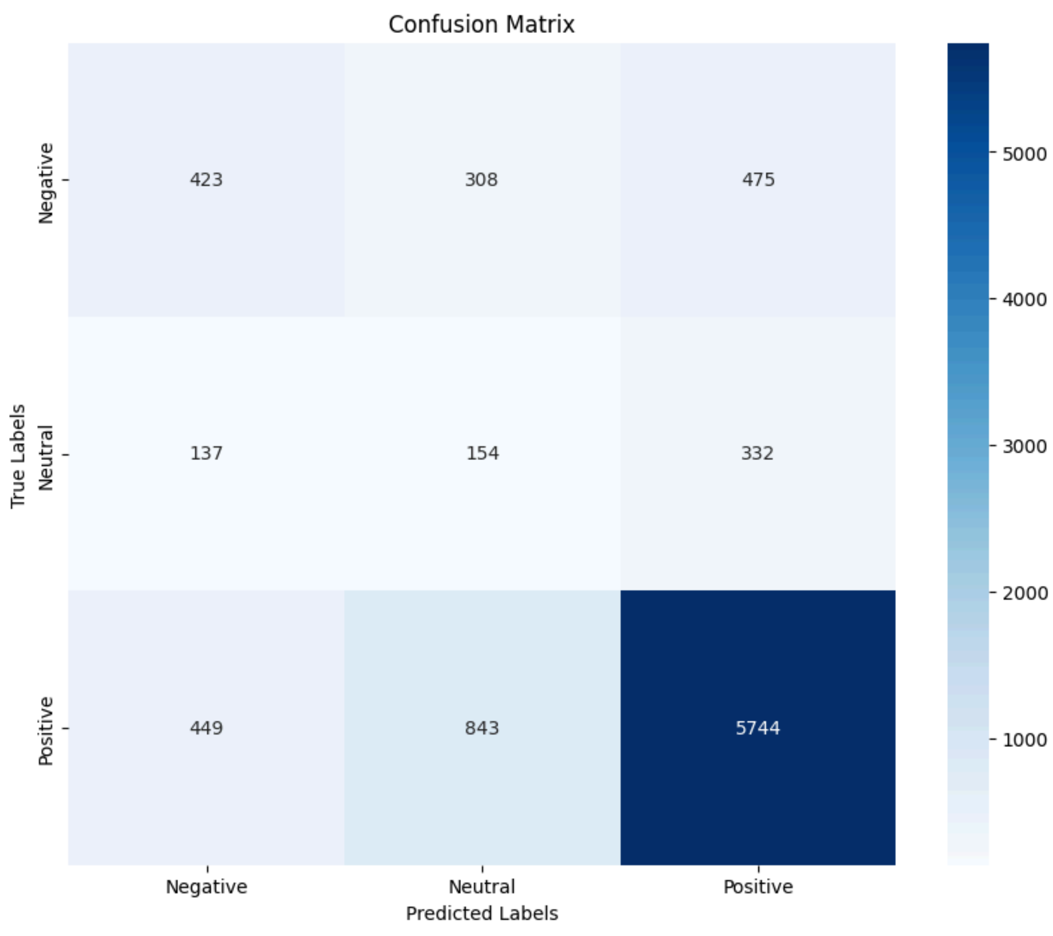
Accuracy: The proportion of correctly predicted sentiments out of the total predictions made. An accuracy of 0.7130 indicates that approximately 71.3% of predictions match the true labels.

Precision: Reflects the model's ability to correctly predict positive and negative sentiments. A precision score of 0.7612 means that, on average, 76.1% of predictions for each class are correct.

Recall: Indicates how well the model captures true positives and true negatives. A recall of 0.7130 implies that the model correctly identifies 71.3% of all actual positive and negative instances.

F1-score: Harmonic mean of precision and recall, providing a balance between the two metrics. An F1-score of 0.7343 suggests overall good performance across the classes.

```
# confusion matrix
class_names = ['Negative', 'Neutral', 'Positive']

conf_matrix = confusion_matrix(y_true, y_pred)

plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()
```



```
# Identify wrong predictions
wrong_predictions = np.where(y_pred != y_true)[0]

# Extract the corresponding review texts, ratings, and sentiments
wrong_samples = test_df.iloc[wrong_predictions].copy()

# Add predicted sentiment to the wrong samples using .loc
wrong_samples.loc[:, 'predicted_sentiment'] = y_pred[wrong_predictions]

# Map numeric sentiments to string labels (optional, for better readability)
sentiment_mapping = {0: 'Negative', 1: 'Neutral', 2: 'Positive'}
wrong_samples.loc[:, 'sentiment'] = wrong_samples['sentiment'].map(sentiment_mapping)
wrong_samples.loc[:, 'predicted_sentiment'] = wrong_samples['predicted_sentiment'].map(sentiment_mapping)

# Display some of the wrong predictions
wrong_samples[['reviews.rating', 'reviews.text', 'sentiment', 'predicted_sentiment']].head(10)
```

	reviews.rating	reviews.text	sentiment	predicted_sentiment
0	3	I order 3 of them and one of the item is bad q...	Neutral	Positive
2	5	Well they are not Duracell but for the price i...	Positive	Neutral
8	3	These do not hold the amount of high power jui...	Neutral	Negative
11	3	When I first started getting the Amazon basic ...	Neutral	Positive
14	5	we have many things that need aa battery they ...	Positive	Neutral
22	5	They last as long as Duracell batteries in my ...	Positive	Negative
24	1	These do not last long at all very cheap batte...	Negative	Neutral
26	4	These Amazon batteries did the job although I ...	Positive	Neutral
29	3	these were under a light we thought they were ...	Neutral	Positive
38	5	These last as well as Energizer- half the price.	Positive	Neutral

These examples highlight areas where the model may have misclassified sentiments, such as:

- Contextual Understanding: Difficulty in understanding nuanced or subtle expressions in reviews.
- Reviewer's Tone: Variability in how sentiment is expressed, which might not always align with the rating scale.
- Domain-specific Knowledge: Challenges in recognizing domain-specific phrases or jargon that signify sentiment.

To improve model performance, further exploration of misclassified examples could guide adjustments in preprocessing steps, model architecture, or training strategies. Additionally, considering more advanced techniques like ensemble methods or leveraging pre-trained language models might enhance the model's ability to capture intricate sentiment patterns in reviews.

```
# model_save_path = '/content/drive/My Drive/final_LSTM_model_final'
# model_4.save(model_save_path)
```

Transformer approach (HuggingFace API)

Processing the dataframe

The dataset initially contains reviews categorized into 36 distinct categories. However, these categories are grouped into 4 primary categories, suggesting a higher-level classification scheme.

```
print(f"Original Categories: {len(df['categories'].unique())}")
print(f"Primary Categories: {len(df['primaryCategories'].unique())}")
```

```
Original Categories: 36
Primary Categories: 4
```

We then aggregate reviews for each combination of primary category and rating. It transforms individual reviews into concatenated strings.

```
concatenated_reviews = []
# Iterate over unique primary categories
for category in df['primaryCategories'].unique():
    # Filter DataFrame for the current category
    category_df = df[df['primaryCategories'] == category]

    # Iterate over unique ratings within the current category
    for rating in category_df['reviews.rating'].unique():
        # Filter DataFrame for the current category and rating
        group = category_df[category_df['reviews.rating'] == rating]

        # Concatenate reviews into a single string or list
        concatenated_reviews.append({
            'primaryCategories': category,
            'reviews.rating': rating,
            'concatenated_reviews': " ".join(group['reviews.text'].tolist())
        })

concatenated_reviews_df = pd.DataFrame(concatenated_reviews)
```

```
concatenated_reviews_df.head()
```

	primaryCategories	reviews.rating	concatenated_reviews
0	Tablets	5.0	This product so far has not disappointed. My c...
1	Tablets	4.0	I've had my Fire HD 8 two weeks now and I love...
2	Tablets	2.0	Didn't have some of the features I was looking...
3	Tablets	1.0	i Bought this around black friday for $60 hopi...
4	Tablets	3.0	I was hoping to use Google launcher with this ...

We also aggregate the columns 'primaryCategories' and 'reviews.rating' to reduce the dataframe size.

This aggregation allows for a more condensed representation of review sentiment and content within each category and rating group and facilitates subsequent analysis such as summarization tasks.

We also apply text preprocessing to clean and tokenize the concatenated_reviews column using a custom function preprocess_text.

```
# Join 'reviews.rating' and 'primaryCategories' into a single 'id' column
concatenated_reviews_df['id'] = concatenated_reviews_df['primaryCategories'] + ', ' + concatenated_reviews_df['reviews.r

# Create a new DataFrame with 'id' and 'concatenated_reviews' columns
processed_df = concatenated_reviews_df[['id', 'concatenated_reviews']].copy()  # Use .copy() to avoid chained assignment

# Apply preprocess_text function using .loc
processed_df.loc[:, 'concatenated_reviews'] = processed_df['concatenated_reviews'].apply(preprocess_text)
```

```
# Print the processed DataFrame
processed_df.head(20)
```

	id	concatenated_reviews
0	Tablets, 5.0	product far disappointed child love use like a...
1	Tablets, 4.0	ive fire hd 8 two week love tablet great value...
2	Tablets, 2.0	didnt feature looking returned next day may go...
3	Tablets, 1.0	bought around black friday 60 hoping would awe...
4	Tablets, 3.0	hoping use google launcher tablet really locke...
5	E-readers, 5.0	lightweight portable excellent battery life li...
6	E-readers, 4.0	first ereader didnt know odd refresh took litt...
7	E-readers, 1.0	upgrade mean three year old kindle outperforme...
8	E-readers, 3.0	th size difference noticeable squarish rather ...
9	E-readers, 2.0	easy carry purse pocket doesnt anything better...
10	Accessories, 4.0	finally received kindle lighted leather cover ...
11	Accessories, 3.0	owned kindle keyboard year purchased leather l...
12	Accessories, 5.0	read every single review cover decided buy gla...
13	Accessories, 1.0	dont option password ask buying apps authorize...
14	Accessories, 2.0	q whats difference 1999 amazon 5w usb official...
15	Streaming Devices & Smart Speakers, 5.0	purchased nephew must say awesome buy kid pric...
16	Streaming Devices & Smart Speakers, 4.0	love fact parental control manage tablet usage...
17	Streaming Devices & Smart Speakers, 3.0	could download apps needed control tv bought p...
18	Streaming Devices & Smart Speakers, 2.0	full disclosure ive ipads past needed android ...
19	Streaming Devices & Smart Speakers, 1.0	stay away certified refurbished amazon fire tv...

Final Summarization Model

After some hits and misses we settled on a model based on the Flan-T5-base. We utilize HuggingFace's AutoTokenizer and AutoModelForSeq2SeqLM to load the transformer model 'google/flan-t5-base' pre-trained for sequence-to-sequence tasks.

We implemented batch processing to facilitate the processing of large volumes of text, comprehensive and accurate summarization of concatenated reviews.

```
# Clear GPU memory
gc.collect()
torch.cuda.empty_cache()

# Initialize the summarizer pipeline
tokenizer = AutoTokenizer.from_pretrained("google/flan-t5-base")
model = AutoModelForSeq2SeqLM.from_pretrained("google/flan-t5-base")
summarizer = pipeline("summarization", model=model, tokenizer=tokenizer, device=0)

# Function to summarize concatenated reviews using datasets
def summarize_with_prompt_batch(reviews, batch_size=8):
    # Create a Dataset from reviews
    ds = Dataset.from_dict({'text': reviews})

    # Tokenize and truncate the input text
    ds = ds.map(lambda x: tokenizer(x['text'], truncation=True, padding='max_length', max_length=512), batched=True, bat

    summaries = []
    for i in range(0, len(ds), batch_size):
        batch = ds.select(range(i, min(i + batch_size, len(ds))))
        inputs = {key: torch.tensor(batch[key]).to(model.device) for key in tokenizer.model_input_names}

        # Generate summaries
        summary_ids = model.generate(**inputs, max_length=150, min_length=50, num_beams=4, early_stopping=True)

        # Decode summaries
        summaries.extend(tokenizer.batch_decode(summary_ids, skip_special_tokens=True))

    return summaries

```

We then divide the long text into smaller chunks to manage memory and computational resources during summarization

```python
# Function to process concatenated reviews in manageable chunks
def process_in_chunks(concatenated_reviews, chunk_size=512):
    tokens = tokenizer.tokenize(concatenated_reviews)
    chunks = []
    for i in range(0, len(tokens), chunk_size):
        chunk_tokens = tokens[i:i+chunk_size]
        chunk_text = tokenizer.convert_tokens_to_string(chunk_tokens)
        chunks.append(chunk_text)
    return chunks


```

The reviews are summarized and grouped so each entry provides a concise summary of concatenated customer reviews for its respective product categories and ratings.

```python
# Grouping and summarizing reviews
def group_and_summarize_reviews(df, batch_size=8):
    final_summaries = []

    # Group by primaryCategories and reviews.rating
    grouped = df.groupby(['primaryCategories', 'reviews.rating'])

    for (category, rating), group in grouped:
        concatenated_reviews = " ".join(group['concatenated_reviews'].tolist())

        # Process reviews in chunks
        chunks = process_in_chunks(concatenated_reviews)

        # Summarize each chunk in batches
        chunk_summaries = summarize_with_prompt_batch(chunks, batch_size=batch_size)

        # Combine summaries into a final summary
        final_summary = " ".join(chunk_summaries)

        final_summaries.append({
            'Category': category,
            'Rating': rating,
            'Summary': final_summary
        })

    return pd.DataFrame(final_summaries)


```

The function create_meta_summaries generates a new dataframe with the 'Category', 'Rating' and the 'Meta Summary'. The 'Meta Summary' provides a high-level overview by condensing the already generated summaries with a predefined prompt.

```python
def create_meta_summaries(df, shot_prompt, batch_size=8, max_words=500):
    meta_summaries = []

    # Generate meta-summary for each category and rating
    for idx, row in df.iterrows():
        category = row['Category']
        rating = row['Rating']
        combined_summaries = row['Summary']

        # Add the prompt to the combined summaries
        combined_summaries_with_prompt = shot_prompt + " " + combined_summaries

        # Process combined summaries in chunks
        chunks = process_in_chunks(combined_summaries_with_prompt)

        # Summarize the combined summaries in batches
        meta_summary = " ".join(summarize_with_prompt_batch(chunks, batch_size=batch_size))

        # Ensure the meta-summary is not longer than 500 words
        words = meta_summary.split()
        if len(words) > max_words:
            meta_summary = " ".join(words[:max_words])

        meta_summaries.append({
            'Category': category,
            'Rating': rating,
            'Meta_Summary': meta_summary
        })

    return pd.DataFrame(meta_summaries)

```

```
# Process the DataFrame
processed_df = group_and_summarize_reviews(concatenated_reviews_df)
```

```
Token indices sequence length is longer than the specified maximum sequence length for this model (2882 > 512). Running
Map:   0%|          | 0/6 [00:00<?, ? examples/s]
Map:   0%|          | 0/3 [00:00<?, ? examples/s]
Map:   0%|          | 0/5 [00:00<?, ? examples/s]
Map:   0%|          | 0/8 [00:00<?, ? examples/s]
Map:   0%|          | 0/21 [00:00<?, ? examples/s]
Map:   0%|          | 0/4 [00:00<?, ? examples/s]
Map:   0%|          | 0/5 [00:00<?, ? examples/s]
Map:   0%|          | 0/14 [00:00<?, ? examples/s]
Map:   0%|          | 0/86 [00:00<?, ? examples/s]
Map:   0%|          | 0/296 [00:00<?, ? examples/s]
Map:   0%|          | 0/14 [00:00<?, ? examples/s]
Map:   0%|          | 0/15 [00:00<?, ? examples/s]
Map:   0%|          | 0/42 [00:00<?, ? examples/s]
Map:   0%|          | 0/223 [00:00<?, ? examples/s]
Map:   0%|          | 0/661 [00:00<?, ? examples/s]
Map:   0%|          | 0/26 [00:00<?, ? examples/s]
Map:   0%|          | 0/27 [00:00<?, ? examples/s]
Map:   0%|          | 0/84 [00:00<?, ? examples/s]
Map:   0%|          | 0/367 [00:00<?, ? examples/s]
Map:   0%|          | 0/660 [00:00<?, ? examples/s]
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-37-39ebd91683b5> in <cell line: 19>()
     17
     18 # Create meta-summaries with the one-shot prompt
---> 19 meta_summaries_df = create_meta_summaries(processed_df, shot_prompt)

<ipython-input-36-644bc51f1f96> in create_meta_summaries(df, shot_prompt, batch_size)
     13
     14         # Summarize the combined summaries with the one-shot prompt in batches
---> 15         meta_summary = " ".join(summarize_with_prompt_batch(chunks, prompt=shot_prompt, batch_size=batch_size))
     16
     17         meta_summaries.append({

TypeError: summarize_with_prompt_batch() got an unexpected keyword argument 'prompt'
```

```
# Display the resulting DataFrame with summaries
processed_df[['Category', 'Rating', 'Summary']]
```

	Category	Rating	Summary
0	Accessories	1.0	You don't need this. Just plug your tablet int...
1	Accessories	2.0	5W USB Official OEM Charger and Power Adapter ...
2	Accessories	3.0	It's a joke that Amazon doesn't just ship a ch...
3	Accessories	4.0	Kindle. The light is activated by the Kindle i...
4	Accessories	5.0	My husband's previous tablet died a sudden dea...
5	E-readers	1.0	My Father In Law - this was the most frustrati...
6	E-readers	2.0	It is too heavy for its size! I purchased it t...
7	E-readers	3.0	Not a favorite purchase, had trouble w/shuttin...
8	E-readers	4.0	Kindle Oasis is a lightweight, long-lasting e-...
9	E-readers	5.0	Kindle Oasis is very small and portable & fits...
10	Streaming Devices & Smart Speakers	1.0	BB store for warranty claim, they said to BB S...
11	Streaming Devices & Smart Speakers	2.0	It's a good tablet, but it's not a perfect tab...
12	Streaming Devices & Smart Speakers	3.0	I am not too happy with the Kindle. It's ok bu...
13	Streaming Devices & Smart Speakers	4.0	I like my tablet very much,it does all the thi...
14	Streaming Devices & Smart Speakers	5.0	I've ever used. It's a great tablet for a kid....
15	Tablets	1.0	I have tried resetting my phone and it still d...
16	Tablets	2.0	Fire HD8 we purchased died within 1 week of pu...
17	Tablets	3.0	The Fire HD8 (2016 model) did not have a lot o...
18	Tablets	4.0	I love this tablet. It's a good size and has a...
19	Tablets	5.0	kindle fire 8....this is what my 9 yr old gran...

Using the one-shot approach allows us to improve summaries by providing a predefined prompt that directs the model in generating concise and pertinent insights. This prompt defines the structure and context for summarizing concatenated reviews, ensuring that the model produces meta-summaries that adhere to the prompt's specifications. In doing so, we effectively distill voluminous customer feedback into brief yet informative snippets that capture key aspects of product reviews. This method not only enhances the efficiency of summarization but also supports better decision-making by providing clear and actionable insights from complex textual data.

```
#Define a one-shot prompt
shot_prompt = """
Example 1: purchased expanded memory 64gb sd card 30 cover 20 biggest issue unit slow compared ipad tends freeze every o
Summary: The Fire tablet has received negative feedback due to poor battery life and frequent technical issues, and alth

Example 2: upgrade mean three year old kindle outperformed oasisbattery life better week light lowest setting magnetic c
Summary: Customers are highly dissatisfied with the Kindle and Fire tablets, citing poor battery life, weak magnetic con

Example 3: product far disappointed child love use like adult past two week return due frozen screen reset work battery
Summary: Many users are unhappy with the Fire tablets, highlighting issues like frozen screens, poor battery life, and f

Summarize the following using less or equal to 500 words:
"""

meta_summaries_df = create_meta_summaries(processed_df, shot_prompt)
```

```
Map:   0%|          | 0/1 [00:00<?, ? examples/s]
Map:   0%|          | 0/1 [00:00<?, ? examples/s]
Map:   0%|          | 0/1 [00:00<?, ? examples/s]
Map:   0%|          | 0/2 [00:00<?, ? examples/s]
Map:   0%|          | 0/3 [00:00<?, ? examples/s]
Map:   0%|          | 0/1 [00:00<?, ? examples/s]
Map:   0%|          | 0/1 [00:00<?, ? examples/s]
Map:   0%|          | 0/2 [00:00<?, ? examples/s]
Map:   0%|          | 0/13 [00:00<?, ? examples/s]
Map:   0%|          | 0/43 [00:00<?, ? examples/s]
Map:   0%|          | 0/3 [00:00<?, ? examples/s]
Map:   0%|          | 0/3 [00:00<?, ? examples/s]
Map:   0%|          | 0/6 [00:00<?, ? examples/s]
Map:   0%|          | 0/32 [00:00<?, ? examples/s]
Map:   0%|          | 0/95 [00:00<?, ? examples/s]
Map:   0%|          | 0/4 [00:00<?, ? examples/s]
Map:   0%|          | 0/4 [00:00<?, ? examples/s]
Map:   0%|          | 0/12 [00:00<?, ? examples/s]
Map:   0%|          | 0/48 [00:00<?, ? examples/s]
Map:   0%|          | 0/86 [00:00<?, ? examples/s]
```

```
1 meta_summaries_df[['Category', 'Rating', 'Meta_Summary']]
```

	Category	Rating	Meta_Summary
0	Accessories	1.0	I can honestly say that I don't recommend this...
1	Accessories	2.0	Kindle Fire HDX 8.9 Charger and Power Adapter ...
2	Accessories	3.0	Don't buy this charger. It's a joke that Amazo...
3	Accessories	4.0	Great product and I'm very happy with it. I've...
4	Accessories	5.0	I've been using it for a few years now and it'...
5	E-readers	1.0	Kindle Voyage - this was the stumbling block I...
6	E-readers	2.0	I don't like it. It's a good tablet but it tak...
7	E-readers	3.0	Kindle Voyager is the best. Though it doesn't ...
8	E-readers	4.0	Kindle Fire. Kindle Oasis. Fire Tablet. Amazon...
9	E-readers	5.0	Kindle Oasis is very small and portable & fits...
10	Streaming Devices & Smart Speakers	1.0	I would not recommend this item. Get a Google ...
11	Streaming Devices & Smart Speakers	2.0	Not impressed. Poor voice recognition. Not ver...
12	Streaming Devices & Smart Speakers	3.0	Amazon does not have. I'll stick with Google h...
13	Streaming Devices & Smart Speakers	4.0	I like my tablet very much,it does all the thi...
14	Streaming Devices & Smart Speakers	5.0	I'm a big fan of the Kindle Fire 7 and it's a ...
15	Tablets	1.0	I don't think I'd buy this tablet again. It's ...
16	Tablets	2.0	Fire HD8 is a great tablet, it's not worth the...
17	Tablets	3.0	Fire HD8 (2016 model) did not have a lot of op...
18	Tablets	4.0	Love this tablet. It's a good size and has a n...
19	Tablets	5.0	Love it..actually read my first e-book on it, ...

3. Model Evaluation

Preparation steps

The meta_summaries_df dataframe, which contains the summarized reviews, is joined with the original dataframe df, and stored on the new dataframe merged_df.

With this, we're able to assigned a summarized review to each original review according to their category and rating.

Then, the columns of merged_df are re-sorted into a new dataframe sorted_df and export it to a .csv file.

```
meta_summaries_df = pd.read_csv('/content/drive/MyDrive/Project1/Consumer Reviews of Amazon Products/meta_summaries.csv'
```

```
# Merge both dataframes on 'primaryCategories'
merged_df = pd.merge(meta_summaries_df[['Category', 'Rating', 'Summary']], df[['id', 'reviews.date', 'name', 'primaryCat
                         'categories',
                         'reviews.rating', 'reviews.title', 'reviews.text']], left_on=['Category', 'Rating'], right_on=[

# Drop the duplicate 'primaryCategories' column
merged_df.drop(columns=['primaryCategories', 'reviews.rating'], inplace=True)

# Rename columns for clarity
merged_df.rename(columns={'reviews.rating': 'Rating', 'reviews.text': 'Text'}, inplace=True)

```

```
merged_df.head()
```

	Category	Rating	Summary	id	reviews.date	name	asins	categories	reviews.title
0	Accessories	1.0	I can honestly say that I don't recommend this...	AVqkIj9snnc1JgDc3khU	2017-05-26T00:00:00.000Z	Amazon 5W USB Official OEM Charger and Power A...	B01AHB9C1E	Tablets,Fire Tablets,Computers & Tablets,All T...	Not really statisfied with features
1	Accessories	1.0	I can honestly say that I don't recommend this...	AVsRjfwAU2_QcyX9PHqe	2017-08-01T00:00:00.000Z	Amazon 5W USB Official OEM Charger and Power A...	B01J2G4VBG	Amazon Devices & Accessories,Amazon Device Acc...	Do you need it
2	Accessories	1.0	I can honestly say that I	AVsRjfwAU2_QcyX9PHqe	2017-06-	Amazon 5W USB Official OEM	B01J2G4VBG	Amazon Devices & Accessories,Amazon	cheap - poor fit in

```
# Reorder columns
desired_columns_order = ['id', 'reviews.date', 'name',
                         'Category', 'categories',
                         'Rating', 'reviews.title', 'Text', 'Summary']

sorted_df = merged_df[desired_columns_order]
sorted_df.head()
```

	id	reviews.date	name	Category	categories	Rating	reviews.title	Text	Summary
0	AVqkIj9snnc1JgDc3khU	2017-05-26T00:00:00.000Z	Amazon 5W USB Official OEM Charger and Power A...	Accessories	Tablets,Fire Tablets,Computers & Tablets,All T...	1.0	Not really statisfied with features	Dont have option for password ask you before b...	I can honestly say that I don't recommend this...
1	AVsRjfwAU2_QcyX9PHqe	2017-08-01T00:00:00.000Z	Amazon 5W USB Official OEM Charger and Power A...	Accessories	Amazon Devices & Accessories,Amazon Device Acc...	1.0	Do you need it	Most buyers will have a phone or other charger...	I can honestly say that I don't recommend this...
			Amazon 5W USB					cheap -	I can

```
# dataframe sorted_df: export to csv

sorted_df.to_csv('dataset_with_summaries.csv', index=False)

```

Evaluation and Results

To evaluate the quality of the generated summaries, we sampled a balanced dataset from each category and rating combination to ensure representative evaluation.

We then used ROUGE and BLEU scores to measure the quality of generated text against reference texts.

The ROUGE metrics focus on evaluating the overlap of n-grams between the generated summary or translation and one or more reference texts. They include ROUGE-1 (unigram overlap), ROUGE-2 (bigram overlap), and ROUGE-L (longest common subsequence), each providing insights into different aspects of content overlap and coherence.

BLEU Scores calculates precision by comparing n-gram matches between the generated output and reference translations, penalizing overly repetitive or unnatural translations. Usually higher scores indicate better translation quality in terms of lexical similarity to human-generated references.

Testing against the original review

ROUGE SCORE

```

# Define the number of samples to take per category and rating
samples_per_category_rating = 10

# Initialize ROUGE scorer
scorer = rouge_scorer.RougeScorer(['rouge1', 'rouge2', 'rougeL'], use_stemmer=True)

# Initialize lists to store ROUGE scores
rouge1_scores = []
rouge2_scores = []
rougeL_scores = []

# Function to sample balanced data from each category and rating combination
def sample_balanced_data(group):
    # Sample the data ensuring balance
    if len(group) <= samples_per_category_rating:
        return group.sample(len(group))
    else:
        return group.sample(samples_per_category_rating)
```

```
        return group.sample(samples_per_category_rating)

# Group by 'Category' and 'Rating' and apply sampling
sampled_df = sorted_df.groupby(['Category', 'Rating']).apply(sample_balanced_data).reset_index(drop=True)

# Iterate over each row in the sampled DataFrame
for index, row in sampled_df.iterrows():
    original_text = row['Text']
    summary = row['Summary']

    # Calculate ROUGE scores
    scores = scorer.score(original_text, summary)

    # Extract and store ROUGE scores
    rouge1_scores.append(scores['rouge1'].fmeasure)
    rouge2_scores.append(scores['rouge2'].fmeasure)
    rougeL_scores.append(scores['rougeL'].fmeasure)

# Print average ROUGE scores
print(f"Average ROUGE-1 Score: {sum(rouge1_scores) / len(rouge1_scores)}")
print(f"Average ROUGE-2 Score: {sum(rouge2_scores) / len(rouge2_scores)}")
print(f"Average ROUGE-L Score: {sum(rougeL_scores) / len(rougeL_scores)}")
```

```
Average ROUGE-1 Score: 0.13021523629127776
Average ROUGE-2 Score: 0.022686860533284642
Average ROUGE-L Score: 0.07913672171943575
```

BLEU Score

This metric evaluates the similarity between the generated summary and the original text based on n-gram precision. It provides a single number that represents the quality of the generated summary in terms of how closely it matches the reference text.

```

# Initialize lists to store BLEU scores
bleu_scores = []

# Iterate over each row in the DataFrame
for index, row in sorted_df.iterrows():
    original_text = row['Text']
    summary = row['Summary']

    # Tokenize the original text and summary
    original_tokens = nltk.word_tokenize(original_text.lower())
    summary_tokens = nltk.word_tokenize(summary.lower())

    # Calculate BLEU score
    bleu_score = sentence_bleu([original_tokens], summary_tokens)

    # Store the BLEU score
    bleu_scores.append(bleu_score)

# Print average BLEU score
print(f"Average BLEU Score: {sum(bleu_scores) / len(bleu_scores)}")
```

```
Average BLEU Score: 0.0020095479205393817
```

```
# Plotting both ROUGE and BLEU scores
plt.figure(figsize=(10, 6))

# ROUGE scores
rouge_labels = ['ROUGE-1', 'ROUGE-2', 'ROUGE-L']
rouge_scores = [sum(rouge1_scores) / len(rouge1_scores),
                sum(rouge2_scores) / len(rouge2_scores),
                sum(rougeL_scores) / len(rougeL_scores)]
plt.bar(rouge_labels, rouge_scores, color=['blue', 'green', 'orange'], label='ROUGE Scores')

# BLEU score
bleu_score = sum(bleu_scores) / len(bleu_scores)
plt.bar('BLEU', bleu_score, color='red', label='BLEU Score')

plt.title('Evaluation Metrics Comparison')
plt.xlabel('Metrics')
```