**COMP 262**

**NATURAL LANGUAGE PROCESSING & RECOMMENDER SYSTEMS**

**Group 1**

**Andy Bandela**

**Mahfuzur Rahman**

**Rasheed Adeyemi**

**Dinh Hoang Viet Phuong**

**FINAL PROJECT REPORT**

A screen shot of a computer code

Description automatically generated

Import necessary libraries:

* pandas as pd: Used for data manipulation and analysis.
* gzip: To read gzip compressed files.
* json: To parse JSON data format.

Data Initialization

* data = []: An empty list to hold the JSON objects that will be read from the file.
* gzip.open('AMAZON\_FASHION.json.gz', 'r'): Opens the gzip compressed file in read mode.
* Loop through each line in the file to extract JSON data.
* json.loads(line.decode('utf-8')): Decode each line from UTF-8 and convert from JSON string to Python dictionary.
* Exception handling (try-except block) to manage potential errors during parsing, such as malformed JSON.
* pd.DataFrame(data): Converts the list data, which contains dictionaries, into a pandas DataFrame where keys become column headers and values are the corresponding data entries.

A screenshot of a computer

Description automatically generated

Printing the Dataset Overview:

* print("Dataset Overview:"): This line is printing a string to act as a title or header before displaying the data.
* print(df.head()): Utilizes the head() method from pandas to display the first five rows of the DataFrame df.

Output Analysis:

* The dataset appears to consist of customer reviews from Amazon Fashion, with columns for rating ('overall'), if the purchase was verified ('verified'), the time of the review ('reviewTime'), and other identifiers such as 'reviewerID', 'asin', 'reviewerName'.
* Review content is divided into 'reviewText' for the full comment, and 'summary' for a shorter version.
* 'unixReviewTime' is a timestamp for when the review was left.
* The 'vote' column likely indicates the number of helpful votes a review received.
* 'style' and 'image' columns are present but not populated in the displayed rows, suggesting they contain optional information.
* Missing Data Note: Acknowledge the presence of missing data, denoted by NaN, especially in the 'vote', 'style', and 'image' columns. This suggests that not all reviews have associated voting data, style descriptions, or images.
* Data Type Observation: Point out the presence of various data types within the dataset, including floating-point numbers for ratings, booleans for the 'verified' status, integers for 'unixReviewTime', and strings for text fields.

A screenshot of a computer program

Description automatically generated

Dataset Shape

* Code Explanation: df.shape is a property of the DataFrame that returns a tuple representing the dimensionality (rows, columns).
* Observation: The output reveals the dataset has 883,636 entries and 12 features (columns).

Column Names

* Code Explanation: df.columns returns an Index object containing the column labels of the DataFrame.
* Observation: The dataset includes columns such as 'overall', 'verified', 'reviewTime', 'reviewerID', 'asin', 'reviewerName', 'reviewText', 'summary', 'unixReviewTime', 'vote', 'style', and 'image'.

Dataset Information

* Code Explanation: df.info() method prints information about the DataFrame including the index dtype and columns, non-null values, and memory usage.
* Observation:
* Data types include float64, bool, int64, and object.
* Non-null counts indicate potential missing values in 'reviewerName', 'reviewText', 'summary', 'vote', 'style', and 'image' columns.
* Memory usage is approximately 75.0+ MB, suggesting the dataset's size in memory, which is important for understanding the computational resources required for processing.

Dropping 'vote' and 'image' columns because they contain 90-96% of missing values (vote - 79900, image - 28807), and their descriptions also indicates that they are not crucial. 'vote' column is the helpful votes of the review and 'image' column is the images that users post after they have received the product.

A screenshot of a computer program

Description automatically generated

Dropping Columns

* Code Explanation: The df.drop() method is used to drop the columns named 'vote' and 'image' from the DataFrame.
* Reasoning: Provide rationale for dropping these columns, such as a large number of missing values or irrelevance to the analysis.

DataFrame Shape Before Deduplication

* Code Explanation: df.shape is called to show the number of rows and columns before removing duplicate entries.
* Columns After Dropping: Confirm that the columns 'vote' and 'image' have been removed, updating the DataFrame to 10 columns.

Filling Missing Values

* Code Explanation: df['reviewText'].fillna(df['summary'], inplace=True) fills missing 'reviewText' entries with the corresponding 'summary' values. The fillna() function is also used with a dictionary to fill missing 'style' entries with an empty string, and any remaining missing 'reviewText' entries are also filled.
* Reasoning: Explain the decision to use 'summary' to fill 'reviewText', possibly due to their similar content nature. Justify the use of an empty string for 'style', which could be due to the absence of a suitable substitute.

A screenshot of a computer program

Description automatically generated

Total Number of Reviews

* Code Explanation: len(df) is used to count all entries in the DataFrame, equating to the total number of reviews.
* Result: The dataset contains 883,636 reviews.

Average Rating

* Code Explanation: df['overall'].mean() calculates the mean of the 'overall' column, which represents the rating given in each review.
* Result: The average rating across all reviews is approximately 3.91 (rounded to two decimal places).

Number of Unique Categories (ASINs)

* Code Explanation: len(df['asin'].unique()) calculates the count of unique values in the 'asin' column, which are Amazon Standard Identification Numbers representing unique products.
* Result: There are 186,189 unique product categories in the dataset.

Number of Unique Users (Reviewer IDs)

* Code Explanation: len(df['reviewerID'].unique()) finds the number of unique 'reviewerID' values, indicating the count of distinct reviewers.
* Result: The dataset includes reviews from 749,233 unique users.

A screenshot of a computer program

Description automatically generated

Distribution of Reviews Per Day

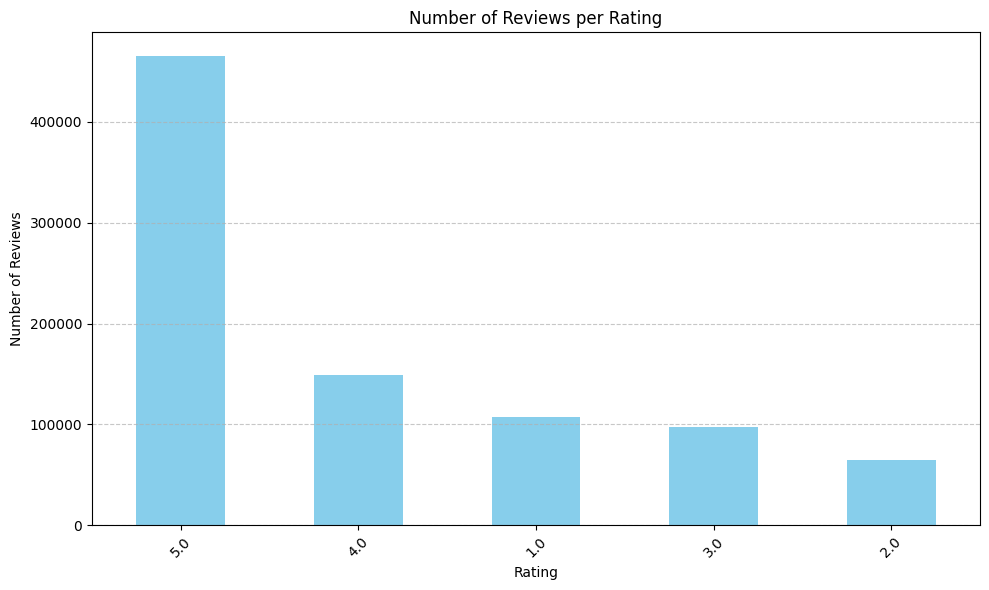
* Code Explanation: df['reviewTime'].value\_counts() is used to count the number of reviews for each unique date in the 'reviewTime' column.
* Result: The output indicates varied numbers of reviews per day, with certain dates having a high frequency of reviews, such as January 15, 2016, with 3,241 reviews, and other dates with only one review.

Distribution of Reviews Per Rating

* Code Explanation: df['overall'].value\_counts() computes the count of each unique rating value in the 'overall' column.
* Result: The distribution of ratings shows that the majority of reviews have a 5.0 rating, followed by 4.0 and then lower ratings, with the fewest reviews having a 2.0 rating.

A screenshot of a computer program

Description automatically generated



Visualizing the Distribution of Reviews per Rating

* The bar chart displays a descending number of reviews from 5-star to 2-star ratings, with 5-star ratings being the most common.
* This visual indicates a trend toward higher ratings among the reviews, which is consistent with the descriptive statistics obtained earlier.

A screen shot of a computer

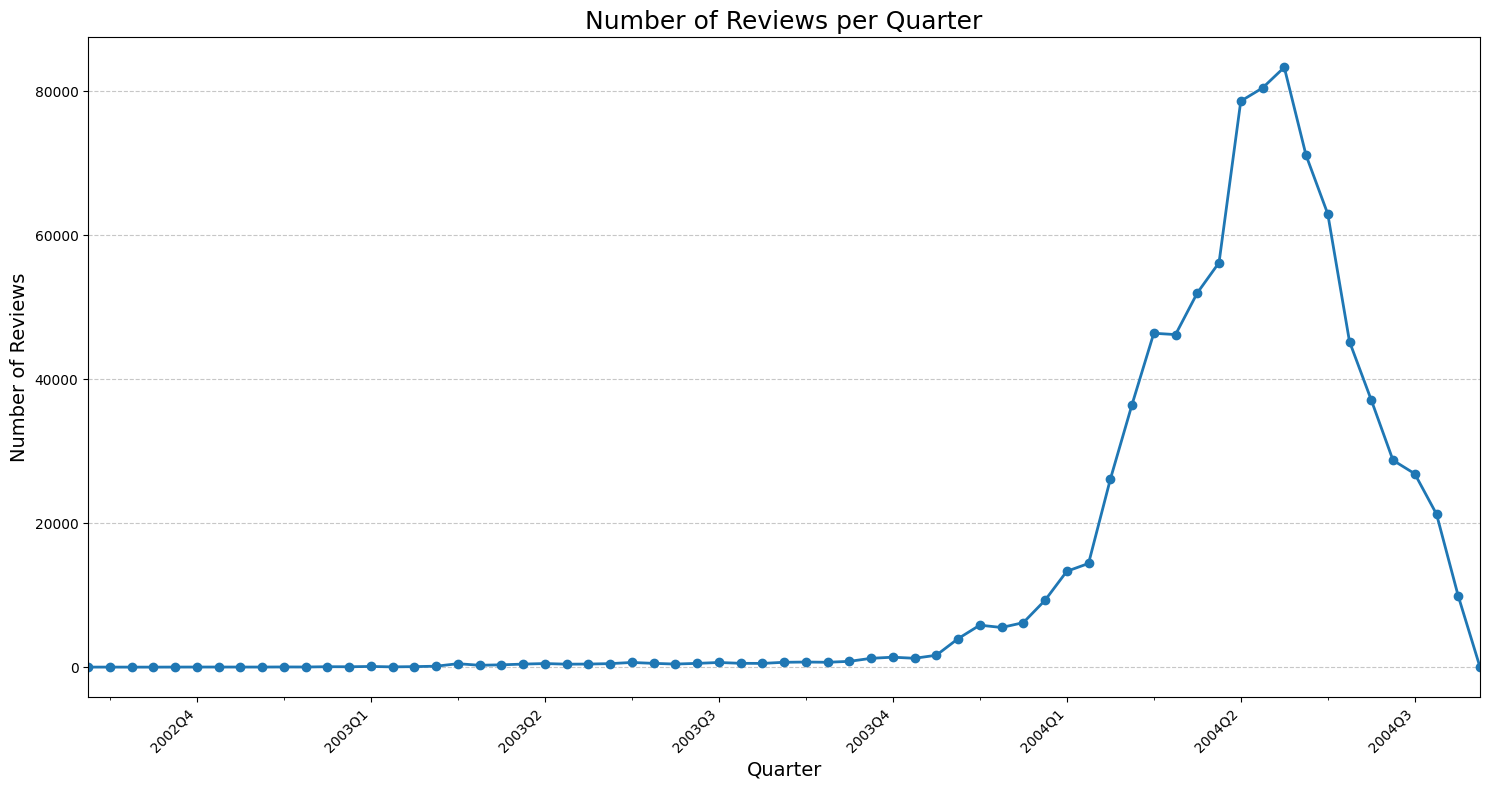
Description automatically generated

Identifying the Product Category with the Most 5-Star Ratings

* Filtering for 5-Star Ratings: The DataFrame df is filtered to create a subset ratings\_of\_5 that includes only the entries where the 'overall' rating equals 5.
* Grouping by Product ID: This subset is then grouped by the 'asin' column, which represents the unique identifier for each product category, and a count of the number of reviews per product is computed.
* Identifying the Maximum: The product with the maximum number of 5-star reviews is identified using the idxmax() function on the grouped data.
* Extracting Reviews for the Top Product: The original DataFrame df is then filtered again to obtain all reviews for the product category with the highest number of 5-star ratings.
* Counting Total Reviews: The total number of reviews for this top-rated product is calculated by taking the length of the resulting DataFrame.
* Outcome: The ASIN B00RLSCLJM corresponds to the product category with the highest number of 5-star ratings, amounting to 3,638 reviews.

A computer screen shot of a program code

Description automatically generated



Trend Analysis of Reviews per Quarter

* The 'reviewTime' column in the dataset was converted to datetime format and then to quarterly periods, allowing for grouping and counting of reviews by quarter. A line graph was created to visualize this data, showing the trend of review submissions over time.
* The graph reveals a significant peak in the number of reviews during a recent quarter, followed by a steep decline. This trend suggests a surge in activity, possibly due to external influences or internal promotions, which is crucial for understanding customer engagement patterns and can inform future business strategies.

A screenshot of a computer program

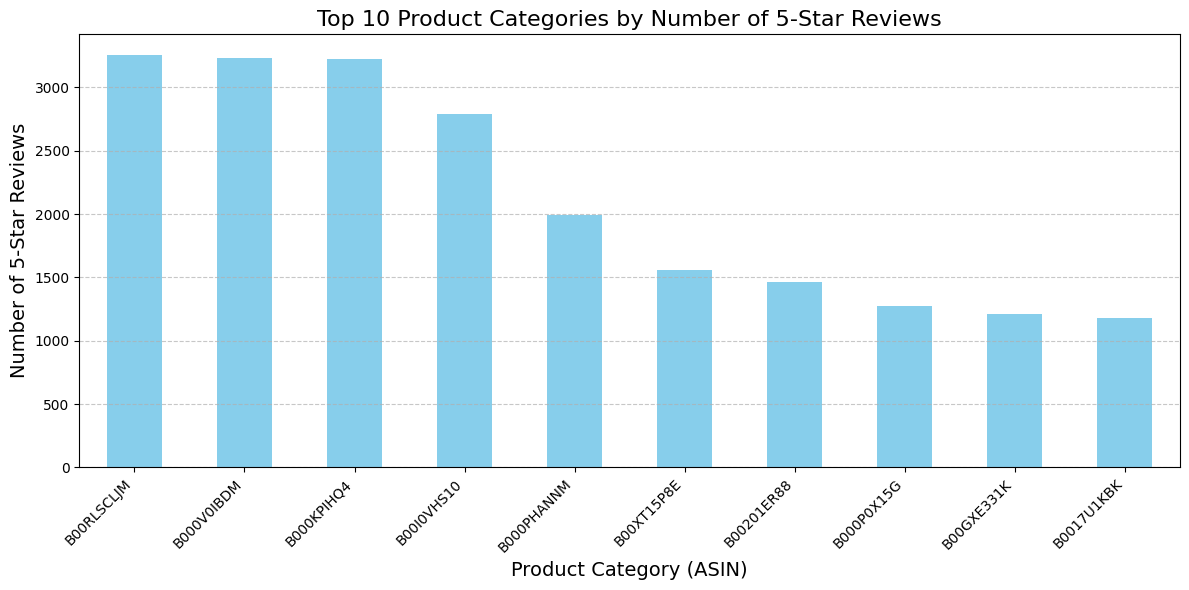
Description automatically generated

The accompanying image visualize the top 10 product categories with the most reviews in a bar chart:

* The data is first processed to count the number of reviews for each product using value\_counts() on the 'asin' column.
* The top 10 products with the most reviews are selected using head(10).
* These top 10 are then plotted as a bar chart with the number of reviews on the y-axis and the product categories (represented by ASINs) on the x-axis.
* The resulting visualization shows that the product with ASIN B000V0IBDM has the highest number of reviews, followed closely by B000KPIHQ4.

A screenshot of a computer program

Description automatically generated

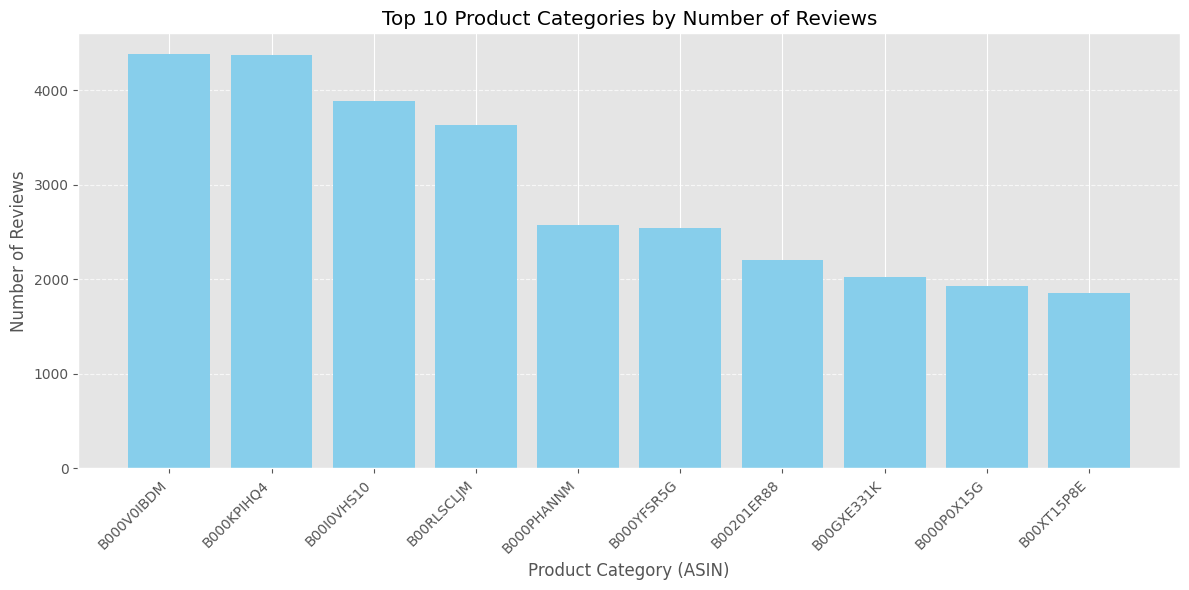


The code and the corresponding bar chart represent the top 10 product categories with the highest number of 5-star reviews:

* The DataFrame is filtered to include only 5-star reviews, then a count of these reviews per product is calculated. The 10 products with the most 5-star reviews are identified and visualized in a bar chart.
* In the bar chart, each bar corresponds to a product category (denoted by ASIN) and the height reflects the number of 5-star reviews. The chart is formatted with a clear title, axis labels, and the x-axis labels are rotated for better readability. The data indicate that the product with ASIN B00RLSCLJM has received the most 5-star

A screen shot of a computer program

Description automatically generated



The bar chart and code provide a detailed analysis of the top 10 product categories by the number of reviews on Amazon. The code processes the dataset to:

* Convert 'reviewTime' to a datetime format.
* Create a new column 'reviewLength' to capture the length of each review.
* Group the data by 'asin' and calculate the count and average of the 'overall' ratings and the average review length.
* Sort the products based on the review count to identify the most reviewed products.
* The bar chart plots these top 10 ASINs against their respective review counts. The products are sorted in descending order, showing the ASIN B000V0IBDM with the highest number of reviews and B000KPIHQ4 a close second, indicating their popularity or high levels of customer engagement.

A graph with numbers and text

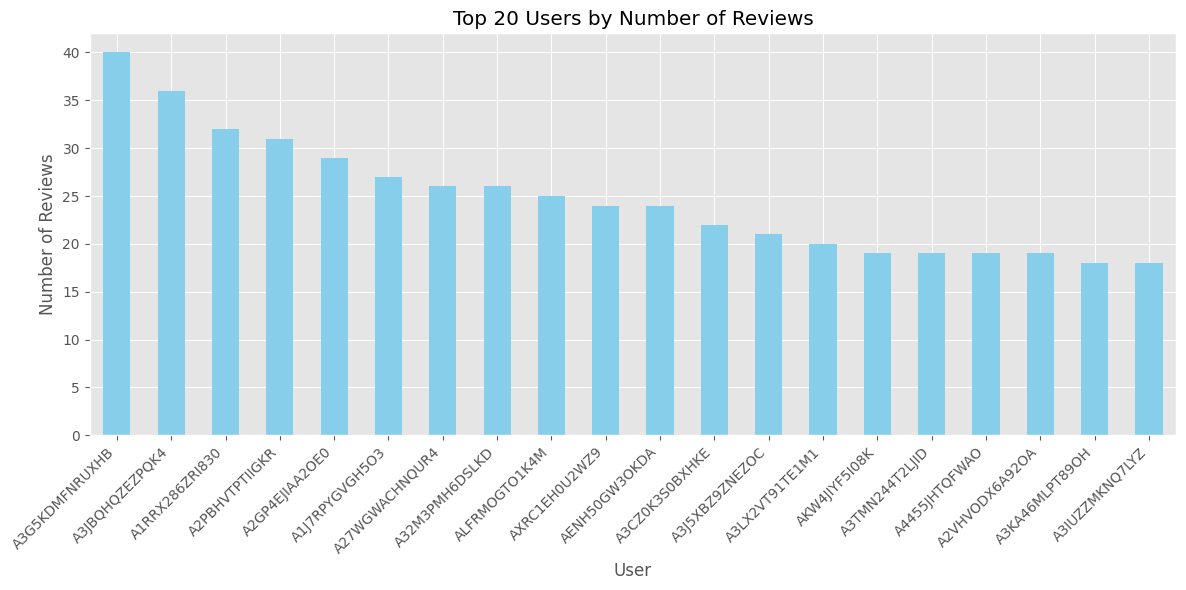
Description automatically generated with medium confidence

The histogram represents the distribution of the number of reviews across all product categories on a logarithmic scale for the number of products:

* The x-axis shows the number of reviews received by products.
* The y-axis represents the number of products, adjusted to a logarithmic scale to better visualize a wide range of values.
* The histogram illustrates that a large number of products have a relatively small number of reviews, while only a few products have a very high number of reviews.
* The bin size for the histogram is set to 50, providing a granular look at the distribution.
* This visualization is useful for understanding the review dynamics across different products, indicating that most products tend to receive fewer reviews, a common trait in e-commerce platforms where few items receive the bulk of attention.

A screenshot of a computer program

Description automatically generated



The bar chart visualizes the top 20 users by the number of reviews they have written, and the code outlines the process to derive this information:

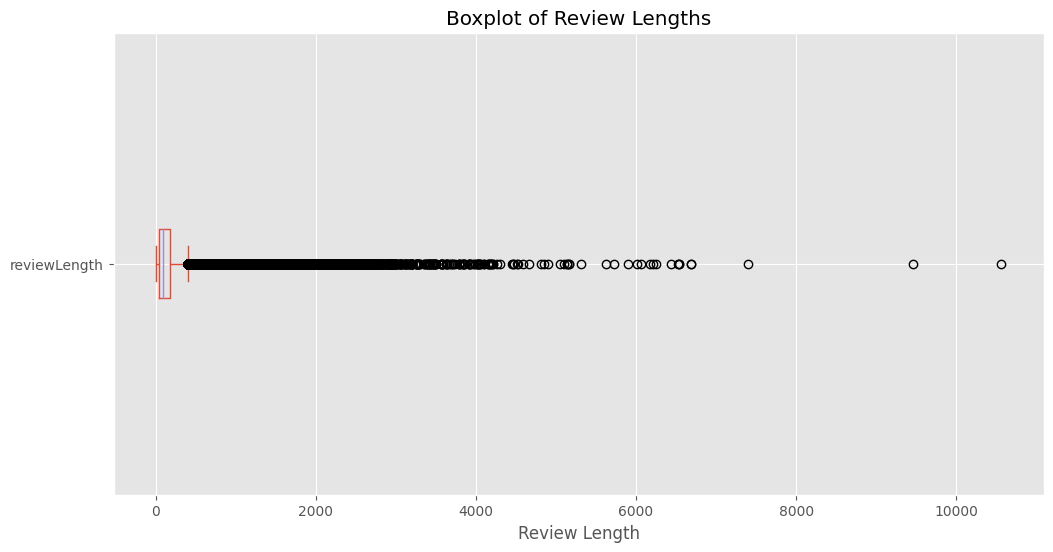
* The value\_counts() method is used on the 'reviewerID' column to calculate the number of reviews per user.
* The resulting series is printed to show the distribution of reviews among users.
* Only the top 20 most active users are selected for visualization to maintain clarity.
* A bar chart is plotted with users on the x-axis and the number of reviews on the y-axis, highlighting the users with the most reviews.
* The chart shows a descending order of review counts starting from the user with the highest number of reviews. It illustrates the disparity in activity levels among users, with a select few contributing a high volume of reviews.

A screenshot of a computer program

Description automatically generated

A graph showing a distribution of review length

Description automatically generated

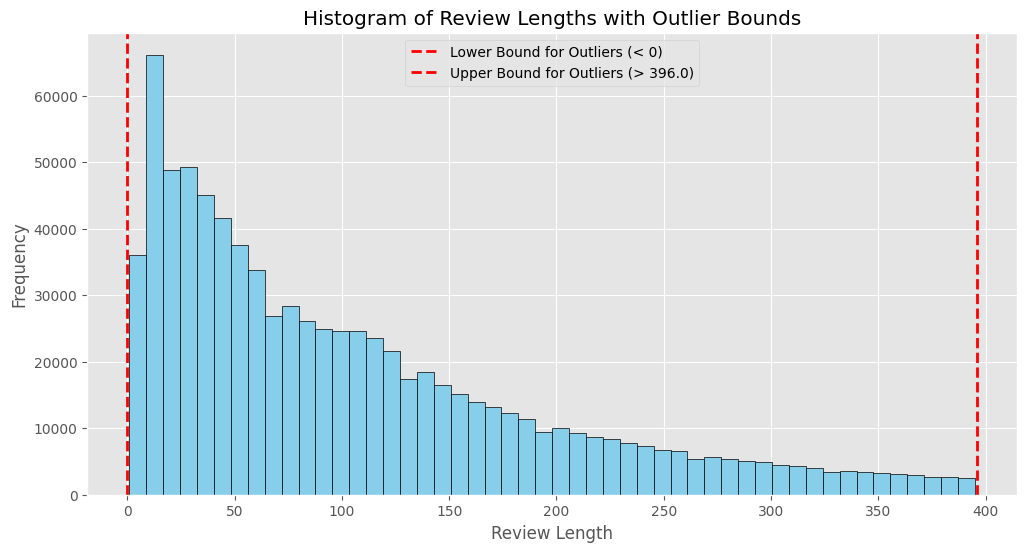


The code calculates the length of each review and then generates a histogram and a boxplot to visualize the distribution of review lengths across the dataset:

* The reviewLength column is created by applying a function to calculate the length of the text in each review.
* Summary statistics for review lengths are printed, showing metrics like mean, standard deviation, and quartiles.
* The histogram displays the frequency of review lengths, with a higher concentration of shorter reviews and a right-skewed distribution.
* The boxplot highlights the median, quartiles, and potential outliers in review lengths, with the presence of extreme outliers indicated by points beyond the whiskers.
* The summary statistics indicate that while the average (mean) review length is around 148 characters, the distribution is wide, with a maximum review length of 10,565 characters. This suggests that most reviews are concise, but there are a few very lengthy reviews. The boxplot further emphasizes the skewness in the data and helps identify outliers in review lengths.

A screen shot of a computer screen

Description automatically generated



The code and histogram visualize the lengths of reviews, with a focus on identifying and displaying outliers:

* The code computes the first and third quartiles (Q1 and Q3) of the reviewLength distribution, then calculates the interquartile range (IQR).
* Outliers are defined as those review lengths that fall below Q1 - 1.5 \* IQR or above Q3 + 1.5 \* IQR.
* A filtered dataset excluding these outliers is created for a clearer histogram.
* The histogram is plotted with vertical dashed lines marking the lower and upper bounds for outliers.
* The histogram shows the distribution of review lengths, with the majority of reviews being concise and only a few being very lengthy.
* In the histogram, the dashed red lines indicate the boundaries outside of which reviews are considered outliers. This visualization helps to understand the typical review length while acknowledging the presence of unusually long reviews.

A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

The code block calculates the length of summary texts in the dataset and identifies potential outliers based on statistical measures:

* Using vectorized operations, the length of each summary is calculated with str.len(), handling missing values by filling them with 0.
* The statistical summary of these lengths indicates that the mean summary length is approximately 21.6 characters, with a maximum length of 663 characters.
* The interquartile range (IQR) is computed to determine the spread of the middle 50% of the data.
* The upper and lower bounds for outliers are defined as 1.5 times the IQR above the third quartile and below the first quartile, respectively.
* Potential outliers are identified as those summary lengths that fall outside these bounds, with 61,814 potential outliers detected.
* The current shape of the DataFrame is confirmed to be 883,636 rows and 12 columns before any duplicate removal.

A screenshot of a computer program

Description automatically generated

The code snippet is designed to clean up the dataset by converting certain columns to strings and removing duplicate entries:

* It converts specified columns to string types to ensure consistency in data type, particularly for columns that may be involved in identifying duplicates.
* The script then identifies duplicate reviews using the .duplicated() method, which can indicate that the same review was recorded more than once.
* A total of 14,520 duplicate reviews were initially detected in the DataFrame.
* Duplicates are dropped based on a subset of key attributes ('reviewerID', 'asin', 'reviewText', 'reviewTime'), which would indicate a duplicate record if all these fields match between two entries.
* After removing duplicates, the DataFrame is re-indexed to maintain sequential indexing.
* The final shape of the DataFrame is reported to have 876,103 rows and 12 columns, indicating the removal of duplicates from the dataset.
* This data cleaning step is crucial for ensuring the integrity of any subsequent analysis, as duplicates could skew results and lead to incorrect interpretations.

A screenshot of a computer program

Description automatically generated

The code snippet refines the dataset by excluding reviews from users whose purchases were not verified:

* The DataFrame is filtered to include only rows where the 'verified' column is True, indicating the reviewer made a verified purchase.
* The index of the DataFrame is reset to account for the removed rows, ensuring the index remains sequential and no gaps are present.
* After filtering, the updated DataFrame size is reported, which now contains 821,660 rows, reflecting the removal of unverified user reviews.
* This step is an essential part of data preparation, particularly for analyses where the authenticity of the review is significant, such as sentiment analysis or product recommendation systems. By focusing on verified reviews, the analysis will be based on more reliable data points.

A screenshot of a computer

Description automatically generated

Function for Labeling Sentiment:

* Description: This function takes a rating value and labels it as either 'Positive', 'Neutral', or 'Negative' based on certain conditions.
* Parameters: rating: The rating value of the product.
* Returns: The sentiment label corresponding to the rating.
* Code Explanation: If the rating is greater than or equal to 4, it's labeled as 'Positive'. If the rating is exactly 3, it's labeled as 'Neutral'. Otherwise, it's labeled as 'Negative'.

Applying Sentiment Labeling Function:

* Description: This code applies the label\_sentiment function to create a new column named 'sentiment' in the DataFrame df.
* Operation: The apply function is used on the 'overall' column of the DataFrame to apply the label\_sentiment function to each value in that column.
* The result is stored in a new column named 'sentiment'.

Selection of Columns for Sentiment Analysis:

* Description: This code selects the 'reviewText' and 'summary' columns from the DataFrame for sentiment analysis.
* Purpose: These columns are chosen because they typically contain the main content of the reviews, which can provide insights into sentiment.

A screen shot of a computer

Description automatically generated

This code segment identifies potential outliers in the lengths of review texts:

* Methodology: It calculates the length of each review in the 'reviewText' column.
* Outliers are detected based on the interquartile range (IQR) method.
* The length of each review text in the 'reviewText' column is computed, with missing values filled as empty strings.
* The first quartile (Q1), third quartile (Q3), and interquartile range (IQR) are calculated from the review lengths.
* Upper and lower bounds for potential outliers are defined based on the quartiles and IQR.
* Reviews with lengths exceeding the upper bound or falling below the lower bound are considered potential outliers.

Results:

* Number of potential outliers: 56332
* Outlier review lengths:
* List of indices and corresponding lengths of potential outliers in the 'reviewText' column.

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

Subset Selection:

* Description: This code segment selects a subset of the original data containing a minimum of 2000 reviews.
* It checks if the total number of reviews in the DataFrame df exceeds 2000.
* If the condition is met, a random sample of 2000 reviews is taken from df using the sample method with a specified random state for reproducibility.
* If the total number of reviews is 2000 or less, the entire dataset is retained.
* The shape of the resulting DataFrame df\_subset is displayed, showing it contains 2000 rows and 13 columns.

Data Exploration:

* Description: This code segment explores the selected subset of data, examining data types, missing values, and summary statistics for numerical columns.
* Data types of columns in the DataFrame df\_subset are captured.
* Missing values in each column of df\_subset are counted using the isnull().sum() method.
* Summary statistics (count, mean, standard deviation, minimum, 25th percentile, median, 75th percentile, and maximum) for numerical columns in df\_subset are computed using the describe() method.

A screenshot of a computer program

Description automatically generated

Text Preprocessing:

* Conversion of 'reviewTime' to Datetime Format:
  + Description: This code converts the 'reviewTime' column from string format to datetime format for better handling of time-related operations.
  + Operation: The pd.to\_datetime() function from the Pandas library is applied to the 'reviewTime' column in the DataFrame df\_subset.
* Creation of 'reviewLength' Column:
  + Description: This code computes the length of each review text and stores it in a new column named 'reviewLength'.
  + Operation: The apply() method is used on the 'reviewText' column of df\_subset to apply a lambda function that calculates the length of each review text.

Text Lowercasing:

* Description: This code converts all characters in the 'reviewText' and 'summary' columns to lowercase to standardize text for further analysis.
* Operations:
  + The apply() method is used on the 'reviewText' column to apply a lambda function that converts each review text to lowercase.
  + Similarly, the 'summary' column is converted to lowercase using a lambda function.

Final Shape and Sample:

* Description: This code checks the final shape of the preprocessed DataFrame and displays a sample of preprocessed data to ensure that preprocessing steps were applied correctly.
* Operations:
  + The shape attribute is used to retrieve the dimensions of the DataFrame df\_subset after preprocessing.
  + The .head() method is applied to select the first few rows of the DataFrame df\_subset containing columns: 'reviewText', 'summary', 'reviewTime', and 'reviewLength'.

A screen shot of a computer program

Description automatically generated

Initialize variables for stop words and lemmatization.

* Create a set of English stop words using stopwords.words('english').
* Instantiate a WordNetLemmatizer object for lemmatization.

Text Normalization Function:

* Description: Function to normalize text by tokenization, stop word removal, and lemmatization.
* Tokenize the input text using word\_tokenize() from NLTK.
* Remove stop words and apply lemmatization to each word.
* Return the normalized text as a string.

Normalization Applied to Columns:

* Description: Apply text normalization to 'reviewText' and 'summary' columns.
* Apply the normalize\_text function to each element in the 'reviewText' and 'summary' columns using the apply() method.

A screenshot of a computer program

Description automatically generated

One-Hot Encoding:

* Description: Encode categorical variables 'style' and 'sentiment' using one-hot encoding.
* pd.get\_dummies() is used to create dummy variables for the 'style' and 'sentiment' columns.
* Prefixes 'style' and 'sentiment' are added to the dummy variable column names.
* style\_encoded and sentiment\_encoded DataFrames containing one-hot encoded columns.

Concatenation:

* Description: Concatenate the one-hot encoded DataFrames with the original DataFrame.
* pd.concat() is used to concatenate df\_subset, style\_encoded, and sentiment\_encoded along the columns axis.
* The resulting DataFrame is stored in df\_ml.

Dropping Original Columns:

* Description: Drop the original 'style' and 'sentiment' columns from the DataFrame.
* The drop() method is used to remove 'style' and 'sentiment' columns from df\_ml.
* The columns are dropped along the columns axis (axis=1).

TF-IDF Transformation:

* Description: Transform text data ('reviewText' and 'summary') into numerical features using TF-IDF vectorization.
* The 'reviewText' and 'summary' columns are concatenated into a single 'combined\_text' column.
* TfidfVectorizer is initialized with specified parameters (max\_features, min\_df, max\_df).
* The fit\_transform() method is applied to 'combined\_text' to create the TF-IDF feature matrix tfidf\_matrix.

A screen shot of a computer program

Description automatically generated

Dataset Splitting:

* Extract the 'overall' rating column as the target variable y.
* Split tfidf\_matrix and y into training and testing sets using train\_test\_split().
* Parameters:
  + test\_size=0.3, random\_state=42, stratify=y.
* Results:
  + X\_train, X\_test: Training and testing feature matrices.
  + y\_train, y\_test: Training and testing target variables.

A screenshot of a computer program

Description automatically generated

Model 1: Logistic Regression

Logistic Regression Model Initialization:

* Description: Initialize the logistic regression model.
* Operation: LogisticRegression from sklearn.linear\_model is instantiated with parameters random\_state=42 for reproducibility and max\_iter=1000 to set the maximum number of iterations. The logistic regression model is stored in the variable log\_reg.

Model Training:

* Description: Train the logistic regression model.
* Operation: The fit() method is called on the logistic regression model (log\_reg) with training data (X\_train and y\_train) as arguments.

Prediction on Testing Set:

* Description: Make predictions on the testing set using the trained logistic regression model.
* Operation: The predict() method is used to predict the ratings for the testing set features (X\_test), and the predicted ratings are stored in y\_pred\_log\_reg.

Model Evaluation:

* Description: Evaluate the performance of the logistic regression model.
* Operation: Calculate the accuracy score using accuracy\_score(y\_test, y\_pred\_log\_reg). Print the classification report using classification\_report(y\_test, y\_pred\_log\_reg) to display precision, recall, F1-score, and support for each class, along with macro and weighted averages.
* Results:
  + Logistic Regression Accuracy: 0.7167 (approximately)
  + Classification report showing precision, recall, F1-score, and support for each rating class, as well as macro and weighted averages.

A screenshot of a computer program

Description automatically generated

Model 2: Support Vector Machine (SVM)

SVM Model Initialization:

* Description: Initialize the support vector machine (SVM) model.
* Operation: SVC (Support Vector Classifier) from sklearn.svm is instantiated with parameters kernel='linear' for using a linear kernel and random\_state=42 for reproducibility. The SVM model is stored in the variable svm\_model.

Model Training:

* Description: Train the SVM model.
* Operation: The fit() method is called on the SVM model (svm\_model) with training data (X\_train and y\_train) as arguments.

Prediction on Testing Set:

* Description: Make predictions on the testing set using the trained SVM model.
* Operation: The predict() method is used to predict the ratings for the testing set features (X\_test), and the predicted ratings are stored in y\_pred\_svm.

Model Evaluation:

* Description: Evaluate the performance of the SVM model.
* Operation:
  + Calculate the accuracy score using accuracy\_score(y\_test, y\_pred\_svm).
  + Print the classification report using classification\_report(y\_test, y\_pred\_svm) to display precision, recall, F1-score, and support for each class, along with macro and weighted averages.

Results:

* SVM Accuracy: 0.7083 (approximately)
* Classification report showing precision, recall, F1-score, and support for each rating class, as well as macro and weighted averages.

A computer screen shot of a program

Description automatically generated

A screen shot of a computer program

Description automatically generated

Performance Metrics - Logistic Regression:

Logistic Regression Accuracy: 0.7166666666666667

Logistic Regression Precision: 0.7183321713273687

Logistic Regression Recall: 0.7166666666666667

Logistic Regression F1 Score: 0.6890921489446685

Confusion Matrix (Logistic Regression):

[[ 53 0 2 4 21]

[ 10 18 6 1 11]

[ 5 2 25 8 23]

[ 2 0 1 37 63]

[ 1 0 2 8 297]]

Performance Metrics - SVM:

SVM Accuracy: 0.7083333333333334

SVM Precision: 0.6962745533293526

SVM Recall: 0.7083333333333334

SVM F1 Score: 0.6824280146885713

Confusion Matrix (SVM):

[[ 56 1 5 4 14]

[ 9 20 7 2 8]

[ 4 4 26 6 23]

[ 2 0 6 33 62]

[ 6 0 3 9 290]]

A screenshot of a computer program

Description automatically generated

Index(['reviewText', 'summary', 'sentiment', 'overall'], dtype='object')

Performance Metrics - Logistic Regression: Old Data

Logistic Regression Accuracy: 0.792

Logistic Regression Precision: 0.7740562640121962

Logistic Regression Recall: 0.792

Logistic Regression F1 Score: 0.7597709776140653

Confusion Matrix (Logistic Regression):

[[ 27 0 0 3 4]

[ 4 2 3 0 11]

[ 7 4 26 6 65]

[ 0 0 8 65 74]

[ 7 0 5 7 672]]

Performance Metrics - SVM : Old Data

SVM Regression Accuracy: 0.803

SVM Regression Precision: 0.8000255981775053

SVM Regression Recall: 0.803

SVM Regression F1 Score: 0.7785729907869403

Confusion Matrix (SVM Regression):

[[ 30 0 3 0 1]

[ 7 2 2 0 9]

[ 9 4 41 2 52]

[ 3 3 8 60 73]

[ 9 0 5 7 670]]

To enhance the rating values of your data using review data, as suggested by practices in state-of-the-art recommender systems based on user reviews:

**1. Sentiment Analysis**

This involves analyzing the text of the reviews to determine the sentiment expressed by the user. The sentiment can provide a more nuanced view of the user's opinion than the raw rating alone. For example, a 3-star rating accompanied by highly positive text could be treated differently from a 3-star rating with mixed or negative text.

Implementation Steps:

* Extract Sentiment: Use natural language processing (NLP) tools to extract sentiment scores from review texts.
* Adjust Ratings: Modify the original rating based on the sentiment score. For instance, if the sentiment analysis yields a very positive sentiment for a medium rating, you might increase the numerical rating slightly.

**2. Semantic Analysis**

Beyond sentiment, understanding the semantics of the review can provide insights into what specific aspects of the product or service the users are commenting on (e.g., quality, reliability, customer service).

Implementation Steps:

* Topic Modeling: Apply techniques like LDA (Latent Dirichlet Allocation) to identify common topics discussed in reviews.
* Rating Adjustment Based on Topics: Adjust the ratings based on the relevance and positivity of comments related to important topics. For example, positive comments on critical aspects like durability in a car review could enhance the overall rating.

**3. Weighted Ratings Based on Review Helpfulness**

Users often rate reviews based on their helpfulness. Reviews deemed more helpful can be given a greater influence on the overall rating of a product.

Implementation Steps:

* Calculate Helpfulness Scores: Develop a metric or use existing 'helpful' counts to assign scores to each review.
* Weight Ratings by Helpfulness: Adjust the influence of each review on the overall rating according to its helpfulness score.

**4. Temporal Weighting**

Recent reviews can be more indicative of the current state of the product or service than older reviews. Giving more weight to recent reviews can provide a more accurate picture.

Implementation Steps:

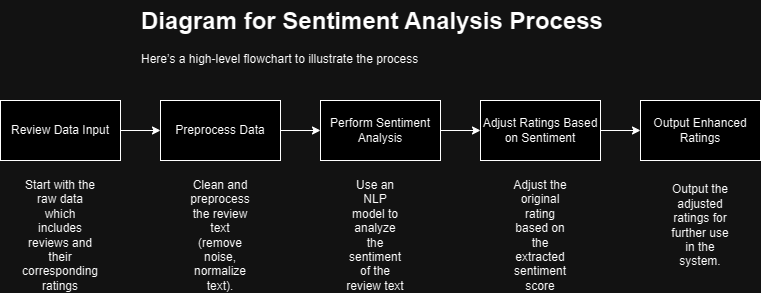
* Determine Review Recency: Calculate how recent each review is.
* Apply Decay Function: Use a decay function to decrease the influence of older reviews on the overall rating.

**5. Review Length and Detail**

Longer, more detailed reviews can be more informative than shorter, vague ones. These reviews can be given more weight in calculating the overall rating.

Implementation Steps:

* Assess Review Length and Depth: Develop metrics to evaluate the length and detail level of each review.
* Adjust Influence Based on Depth: Increase the weight of longer, more detailed reviews in the overall rating calculation.
* By integrating these methods, you can enhance the standard rating values in your data, making them more reflective of the true sentiment and opinions expressed by users in their reviews. This enriched data can significantly improve the performance of recommender systems by providing a deeper understanding of user preferences and product qualities.



**Pseudo-code for Enhancing Ratings via Sentiment Analysis**

A screen shot of a computer program

Description automatically generated

The code aims to process product reviews by loading data, preprocessing text, analyzing sentiment, and then enhancing the numerical ratings based on the sentiment derived from the review texts. This approach combines quantitative and qualitative data to potentially provide a more nuanced rating system for reviews.

Code Explanation

* Loading Data: The load\_data function is designed to read data from a gzipped JSON file and convert it into a Pandas DataFrame. This function iterates over each line in the file, assuming that each line is a separate JSON object, and appends it to a list which is then converted to a DataFrame.
* Text Preprocessing: The preprocess\_text function handles basic text preprocessing by converting the text to lowercase and removing leading and trailing spaces. If the text is missing (NaN), it returns an empty string.
* Sentiment Analysis: The analyze\_sentiment function utilizes TextBlob, a simple Python library for processing textual data. It calculates the polarity of the sentiment in the text, which ranges from -1 (very negative) to +1 (very positive).
* Enhancing Ratings Based on Sentiment:
  + The enhance\_ratings\_with\_sentiment function first preprocesses the review texts and calculates their sentiment scores.
  + It then adjusts the original ratings based on these sentiment scores using the adjust\_rating function:
  + If the sentiment score is greater than 0.1, the original rating is increased by 1, capped at a maximum of 5.
  + If the sentiment score is less than -0.1, the original rating is decreased by 1, with a minimum possible rating of 1.
  + Ratings remain unchanged for sentiment scores between -0.1 and 0.1.
* Rating Adjustment Execution and Output: Finally, the enhance\_ratings\_with\_sentiment function is called with a DataFrame df containing reviews, and the first few modified ratings are printed to verify the adjustments.

A screenshot of a computer program

Description automatically generated

Result Explanation

The result is a DataFrame displaying original and enhanced ratings alongside the sentiment scores:

* Review 1: Positive sentiment (0.25) and a high original rating (5.0) which remains at 5.0 as it cannot exceed this value.
* Review 2: Negative sentiment (-0.155) with a low original rating (2.0). The enhanced rating is lowered to 1.0, reflecting the negative sentiment.
* Review 3: Neutral sentiment (0.0), hence the original rating (2.0) remains unchanged.
* Review 4: Similar to the first, with positive sentiment and an original rating of 5.0, so it stays at 5.0.
* Review 5: Positive sentiment (0.23125) with a high original rating (4.0). The enhanced rating is increased to 5.0.

A screen shot of a computer program

Description automatically generated

The goal of this script is to efficiently summarize long product reviews using advanced natural language processing techniques, specifically leveraging a pretrained model from Hugging Face's transformers library. This process helps in extracting concise summaries from verbose customer feedback, making it easier for potential customers and analysts to quickly grasp the key points.

Code Explanation

* Selecting Long Reviews: The function select\_long\_reviews is designed to identify reviews that exceed a specified word count threshold (default is 100 words). It adds a new column word\_count to the DataFrame df, which represents the number of words in each review. The function then filters these reviews to get the longest ones, limited by num\_reviews parameter (default is 10).
* Initializing the Summarization Model: The initialize\_summarizer function sets up a text summarization pipeline using a model specified by model\_name, defaulting to "sshleifer/distilbart-cnn-12-6". This model is a distilled version of a larger BART model trained on a dataset tailored for summarization tasks, making it efficient and effective for generating summaries.
* Summarizing Reviews: summarize\_reviews takes a list of review texts and a summarization pipeline. It processes each review through the pipeline, which internally adjusts its processing parameters such as max\_length and min\_length to control the length of the summary. It ensures that the summaries are neither too long nor too short relative to the original review length.
* Execution and Output:
  + The script first selects 10 long reviews from a DataFrame df.
  + It then initializes the summarization pipeline.
  + Each selected review is summarized, and these summaries are stored in a new column summary in the long\_reviews DataFrame.
  + Finally, the original reviews and their respective summaries are printed, allowing for easy comparison.

A screen shot of a computer program

Description automatically generated

The script is designed to interact with customer reviews that contain questions, identifying such reviews, and then generating responses automatically using a conversational model. This approach is valuable for automating customer service tasks, enhancing engagement, and providing quick answers to potential concerns raised in reviews.

Code Explanation:

* Identifying Question-Containing Reviews: The find\_question\_review function scans a DataFrame for reviews that likely contain questions. This is achieved by checking for the presence of question words or a question mark in the text of each review. If a review meeting these criteria is found, it randomly selects one such review to return. If no suitable review is found, it returns a default message stating no question-like review is found.
* Initializing the Response Model: The initialize\_response\_model function sets up a conversational model from Hugging Face's transformers library, specifically using "microsoft/DialoGPT-medium". This model is designed for generating human-like responses in a conversational context, using a tokenizer to convert text to a format the model can process, and a model to generate text based on input.
* Generating Responses: The generate\_response function takes a question as input, processes it through the tokenizer and model to generate a response. The function adjusts parameters like maximum length and response diversity through the temperature setting. The function also cleans up the output by removing repetitions of the input question in the generated response, which can sometimes occur with language models.
* Execution and Output:
  + The script initializes the conversational model and tokenizer.
  + It searches the DataFrame df for a review containing a question.
  + If such a review is found, the script generates a response using the initialized model; otherwise, it outputs a message indicating that no response is needed.
  + Finally, it prints the identified question and the generated response.

Result:

