INFO 570 Term Project: Analyzing Customer Satisfaction Through Amazon Product Reviews

Introduction for the Notebook

This notebook outlines the initial steps of a project analyzing customer reviews from the Amazon Product Reviews Dataset. In Step 1, the focus is on defining the project's objectives, establishing the business case, and documenting the relevance of the analysis for improving customer satisfaction and product quality. The dataset is introduced, its structure is verified, and foundational preparations are documented for subsequent steps.

Group 3

Team Members:

- Chinmaya Katagondanahalli Nagaraju
- Paola Andrea Gomez
- Priyanka Kulkarni
- Vetrivel Kumar

Dataset Information

- **Original Source**: Amazon Product Reviews Dataset on Kaggle
- **Current Location**: The dataset has been re-hosted on GitHub for easier access in this project.
- **Reason for Re-hosting**: To facilitate loading directly into the Colab notebook and ensure consistent accessibility for all collaborators.
- **Description**: This dataset contains reviews, ratings, and price details for various Amazon products.
- **Structure**:
  - Columns: `reviews`, `rating`, `prices`, `categories`, and more.
  - Includes JSON-like nested data in the `prices` column.

Importance of the Project

Business Problem: **Understanding and Optimizing Customer Satisfaction and Product Performance in Amazon's Data**

Addressing this business problem is crucial for several reasons:

- **Financial Impact:** Identifying and addressing issues highlighted in customer reviews can reduce product return rates and improve profitability.
- **Customer Loyalty:** By understanding and acting on customer feedback, Amazon can build trust and foster long-term loyalty.
- **Product Improvement:** Insights gained from customer reviews provide Amazon with actionable data to improve product quality and align with customer expectations.
- **Competitive Edge:** Leveraging customer insights gives Amazon a strategic advantage in the e-commerce industry, helping it stay ahead of competitors.

Overall, this analysis of product review data is crucial as it examines customer feedback across a major e-commerce platform, particularly focusing on the rapidly growing smart home technology sector. Understanding these customer insights becomes even more critical given that online reviews directly influence purchasing decisions and brand trust, while the significant gaps in product information (such as missing specifications and attributes) present clear opportunities to enhance product listings and improve overall customer experience.

Business Problem

Customer satisfaction is essential for e-commerce businesses like Amazon to maintain market leadership and brand loyalty. This project addresses the following business problem:

How can Amazon leverage customer reviews to enhance product quality and increase customer satisfaction?

In today's highly competitive market, understanding customer feedback enables Amazon to improve its offerings, which can lead to higher retention rates and customer loyalty. This analysis will offer valuable insights into the factors that drive positive and negative customer reviews, allowing Amazon to take targeted actions that strengthen its product lineup and enhance customer experiences.

Goals and Objectives

The primary objectives of this project are as follows:

1. **Identify Key Factors Influencing Customer Satisfaction:** Perform sentiment analysis to extract the main themes in positive and negative reviews.
2. **Provide Actionable Recommendations:** Based on analysis results, offer suggestions for product improvements and customer satisfaction enhancement.
3. **Explore Trends Across Product Categories:** Identify variations in customer satisfaction across different product types and over time.
4. **Price-Satisfaction Relationship Analysis:** Analyze the correlation between product pricing and customer satisfaction metrics.
5. **Temporal Analysis of Product Ratings:** Map seasonal trends in product ratings and review volumes
6. **Manufacturer Performance Assessment:** Rank manufacturers by average customer satisfaction scores

Intended Outcomes

By achieving these objectives, this project aims to empower Amazon to:

- Make data-driven decisions to improve product quality.
- Enhance customer satisfaction and loyalty through targeted improvements.
- Strengthen its competitive position by responding effectively to customer feedback.
  Analysis Reports: Correlation analysis between price and satisfaction, Seasonal trend analysis dashboard, Manufacturer performance scorecard Visualization Set: Price-satisfaction scatter plots, Manufacturer comparison charts

Overall, the primary goal of this analysis is to enhance customer satisfaction and product performance by systematically analyzing customer review data to identify key drivers of positive and negative feedback across product categories. Through detailed examination of sentiment patterns, product attributes, and category performance, this project aims to provide actionable insights for improving product specifications, optimizing categorization, and increasing product discoverability, ultimately leading to better customer experience and increased sales performance.

Methodology

Data Loading and Preparation

The dataset will be loaded and processed in Google Colab to perform the analysis. Initial steps will include data cleaning, handling missing values, and preparing text data for sentiment analysis.

Analysis Techniques

- **Sentiment Analysis:** Categorize reviews as positive, negative, or neutral to understand the general sentiment.
- **Trend Analysis:** Explore sentiment trends over time and across product categories.
- **Factor Analysis:** Identify key aspects of products that influence customer ratings.

Data Loading and Preparation

The dataset contains multiple variables capturing key aspects of customer reviews and product details. As it was processed in Step 1, the dataset currently has:

- Number of Rows: Approximately **1,500.**
- Number of Columns: **10.**
- Key Variables:
  - **reviews.text**: Customer reviews in textual format.
  - **reviews.rating**: Numerical ratings provided by customers.

- **reviews.numHelpful**: Number of users who found the review helpful.
- **categories**: Product categories.
- **prices**: Nested JSON-like structure containing pricing details:
  - **amountMin and amountMax**: Minimum and maximum prices for the product.
  - **currency**: The currency in which the price is displayed.
  - **isSale**: A flag indicating whether the product is on sale.
  - **merchant**: The seller of the product.
  - **dateAdded**: The date the product's price was added.

```
# Import libraries
import pandas as pd

# Load the dataset from GitHub
url = 'https://raw.githubusercontent.com/chinmaya-nagaraju/INFO570_term_project/main/7817_1.csv'
data = pd.read_csv(url)

# Display the first few rows
data.head()
```

	id	asins	brand	categories	colors	dateAdded	dateUpdated	dimension	ean	
0	AVpe7AsMilAPnD_xQ78G	B00QJDU3KY	Amazon	Amazon Devices,mazon.co.uk	NaN	2016-03-08T20:21:53Z	2017-07-18T23:52:58Z	169 mm x 117 mm x 9.1 mm	NaN	kindlepaperwhite/b00
1	AVpe7AsMilAPnD_xQ78G	B00QJDU3KY	Amazon	Amazon Devices,mazon.co.uk	NaN	2016-03-08T20:21:53Z	2017-07-18T23:52:58Z	169 mm x 117 mm x 9.1 mm	NaN	kindlepaperwhite/b00
2	AVpe7AsMilAPnD_xQ78G	B00QJDU3KY	Amazon	Amazon Devices,mazon.co.uk	NaN	2016-03-08T20:21:53Z	2017-07-18T23:52:58Z	169 mm x 117 mm x 9.1 mm	NaN	kindlepaperwhite/b00

```
# Check the shape of the dataset
print("Number of rows:", data.shape[0])
print("Number of columns:", data.shape[1])

# List all column names
print("Variables in the dataset:")
print(data.columns.tolist())
```

```
Number of rows: 1597
Number of columns: 27
Variables in the dataset:
['id', 'asins', 'brand', 'categories', 'colors', 'dateAdded', 'dateUpdated', 'dimension', 'ean', 'keys', 'manufacturer', 'manufacturerNu
```

```
from IPython.display import display, HTML

# Generate a DataFrame for column names and data types
data_types = pd.DataFrame({'Column Name': data.columns, 'Data Type': data.dtypes})

# Convert the DataFrame to an HTML table with styled formatting
html_table = data_types.to_html(index=False)

# Display the table with custom styling for colors and font size
styled_html = f"""
<div style="font-family:Arial; font-size:16px; color:black; background-color: #f9f9f9; padding:10px; border-radius:5px;">
    <h3 style="color:navy;">Column Data Types</h3>
    {html_table}
</div>
"""

display(HTML(styled_html))
```

Column Data Types

Column Name	Data Type
id	object
asins	object
brand	object
categories	object
colors	object
dateAdded	object
dateUpdated	object
dimension	object
ean	float64
keys	object
manufacturer	object
manufacturerNumber	object
name	object
prices	object
reviews.date	object
reviews.doRecommend	object
reviews.numHelpful	float64
reviews.rating	float64
reviews.sourceURLs	object
reviews.text	object
reviews.title	object
reviews.userCity	float64
reviews.userProvince	float64
reviews.username	object
sizes	float64
upc	float64
weight	object

Exploratory Data Analysis: Reviews Rating and Reviews Helpfulness

An exploratory data analysis of the dataset highlights two key variables: **reviews.rating** and **reviews.numHelpful**. The variable **reviews.rating**, representing customer ratings on a scale from 1 to 5, exhibits a mean of **4.36**, indicating generally favorable customer sentiment. The median value of **5.0** confirms that most customers provided the highest possible rating, with no observations falling below a minimum of **1.0**. This consistency suggests strong customer satisfaction across the reviewed products.

In contrast, the variable **reviews.numHelpful**, which quantifies the number of users who found a review helpful, displays a higher level of variability. With a mean of **83.58**, the data skews heavily toward reviews with significant engagement, but a median value of **0.0** reveals that many reviews received no helpful votes at all. This discrepancy suggests that while some reviews garner considerable attention, the majority fail to engage users meaningfully.

These descriptive statistics provide critical insights into customer satisfaction levels and user engagement, which can guide further analysis and business decision-making.

```
# Compute and display descriptive statistics for numerical variables
data.describe()
```

	ean	reviews.numHelpful	reviews.rating	reviews.userCity	reviews.userProvince	sizes	upc
count	8.980000e+02	900.000000	1177.000000	0.0	0.0	0.0	8.980000e+02
mean	8.443135e+11	83.584444	4.359388	NaN	NaN	NaN	8.443135e+11
std	3.416444e+09	197.150238	1.021445	NaN	NaN	NaN	3.416444e+09
min	8.416670e+11	0.000000	1.000000	NaN	NaN	NaN	8.416670e+11
25%	8.416670e+11	0.000000	4.000000	NaN	NaN	NaN	8.416670e+11
50%	8.416670e+11	0.000000	5.000000	NaN	NaN	NaN	8.416670e+11
75%	8.487190e+11	34.000000	5.000000	NaN	NaN	NaN	8.487190e+11
max	8.487190e+11	997.000000	5.000000	NaN	NaN	NaN	8.487190e+11

Outlier Detection Using Z-Scores

Outlier detection is a critical component of data preprocessing to ensure the quality and reliability of analysis. In this step, Z-scores were computed for key numerical variables such as **reviews.numHelpful** and **reviews.rating**. The Z-score indicates how many standard deviations a data point is from the mean, allowing for the identification of extreme values that may influence analysis results. For this dataset, outliers were identified as data points with Z-scores greater than 3 or less than -3. These extreme values are isolated for further investigation or appropriate handling to maintain the integrity of subsequent analyses.

```
from scipy.stats import zscore
import pandas as pd
from IPython.display import display, HTML

# Ensure 'reviews.numHelpful' and 'reviews.rating' are numeric
data['reviews.numHelpful'] = pd.to_numeric(data['reviews.numHelpful'], errors='coerce')
data['reviews.rating'] = pd.to_numeric(data['reviews.rating'], errors='coerce')

# Calculate Z-scores for 'reviews.numHelpful'
data['z_score_numHelpful'] = zscore(data['reviews.numHelpful'])

# Calculate Z-scores for 'reviews.rating'
data['z_score_rating'] = zscore(data['reviews.rating'])

# Filter for outliers in 'reviews.numHelpful'
outliers_numHelpful = data[(data['z_score_numHelpful'] > 3) | (data['z_score_numHelpful'] < -3)]

# Filter for outliers in 'reviews.rating'
outliers_rating = data[(data['z_score_rating'] > 3) | (data['z_score_rating'] < -3)]

# Display final results with colors and font size
display(HTML(f"""
<div style="font-size:18px; color:green;">
    <p><b>Number of outliers in reviews.numHelpful:</b> {outliers_numHelpful.shape[0]}</p>
    <p><b>Number of outliers in reviews.rating:</b> {outliers_rating.shape[0]}</p>
</div>
"""))
```

Number of outliers in reviews.numHelpful: 0

Number of outliers in reviews.rating: 0

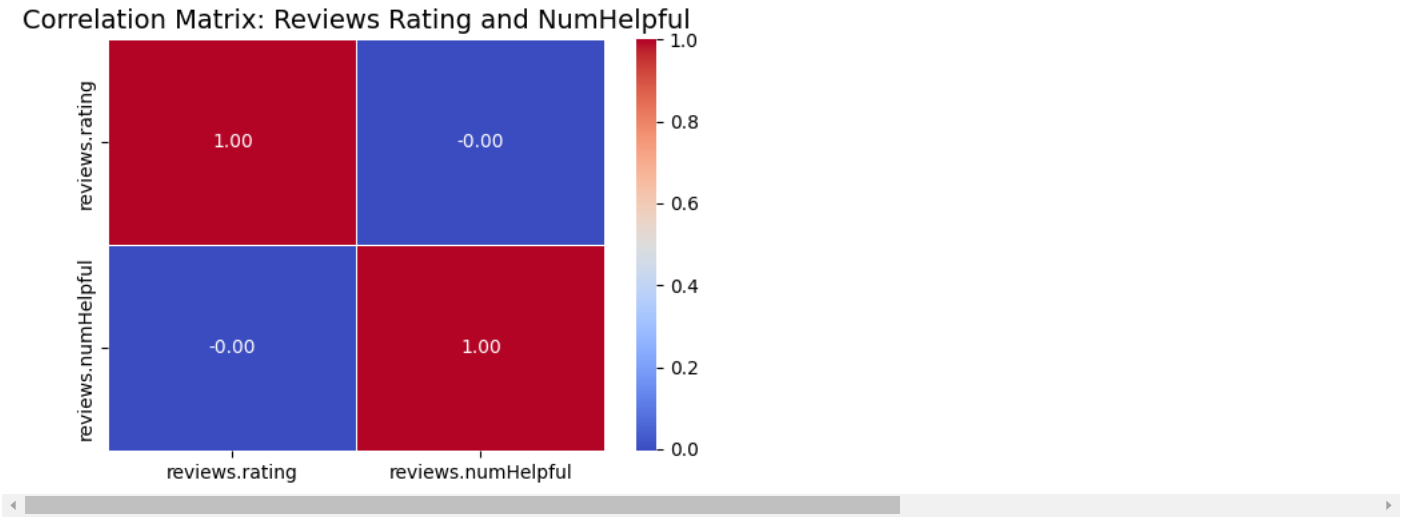
Correlation Analysis

Given the structure of the dataset, only two numerical variables—**reviews.rating** and **reviews.numHelpful**—are available for correlation analysis. A pairwise correlation analysis was conducted to evaluate the relationship between these variables. The correlation coefficient indicates whether there is a significant positive, negative, or negligible relationship between customer ratings and the number of users who found a review helpful. This analysis helps in understanding the potential influence of customer satisfaction on review engagement.

```
import seaborn as sns
```

```
import matplotlib.pyplot as plt

# Select only numerical columns for correlation calculation
numerical_data = data[['reviews.rating', 'reviews.numHelpful']]

# Calculate the correlation matrix
correlation_matrix = numerical_data.corr()

# Display the correlation matrix
print("Correlation Matrix:")
print(correlation_matrix)

# Plot a heatmap for the correlation matrix
plt.figure(figsize=(6, 4))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)
plt.title("Correlation Matrix: Reviews Rating and NumHelpful", fontsize=14)
plt.show()
```

```
Correlation Matrix:
                    reviews.rating  reviews.numHelpful
reviews.rating            1.000000           -0.004772
reviews.numHelpful       -0.004772            1.000000
```



Formulate and Evaluate Business Hypothesis 1

The goal of this hypothesis test is to determine whether "Amazon Devices" (e.g., Echo, Fire TV) consistently receive more positive feedback from customers compared to "Kindle Devices." This insight is critical for evaluating customer satisfaction trends across Amazon's product lines and could help Amazon identify product categories that drive greater customer satisfaction.

- **Null Hypothesis (Ho)**: The proportion of positive reviews for "Amazon Devices" equals the proportion of positive reviews for "Kindle Devices".
- **Alternative Hypothesis (Ha)**: The proportion of positive reviews for "Amazon Devices" is greater than the proportion of positive reviews for "Kindle Devices".
- **Test Type**: Test for Two Proportions

```
# Check the unique categories in the dataset
print("Unique categories in the dataset:")
print(data['categories'].unique())
```

```
Unique categories in the dataset:
['Amazon Devices,mazon.co.uk' 'Kindle Store,Amazon Devices,Electronics'
 'Categories,Amazon Devices,Streaming Media Players' 'Amazon Devices'
 'Categories,Amazon Devices,Electronics Features,Streaming Media Players,Consumer Electronics,See more Amazon Fire TV Digital HD Media S
 'Amazon Devices & Accessories,Amazon Device Accessories,Power Adapters & Cables,Kindle Store,Kindle E-Reader Accessories,Kindle Paperwh
 'Amazon Devices,Kindle Store' 'Electronics,Amazon Devices'
 'Amazon Devices,Kindle Accessories'
 'Amazon Devices,Electronics,Kindle Store'
 'Amazon Devices,Electronics,Kindle Store,Amazon Echo'
 'Amazon Devices,Kindle Store,Kindle Accessories'
 'Amazon Devices,Kindle Store,buy a kindle'
 'Amazon Devices,Home,Smart Home & Connected Living,Smart Hubs & Wireless Routers,Smart Hubs,Home Improvement,Home Safety & Security,Ala
 'Cell Phones & Accessories,Accessories,Screen Protectors,Cell,Amazon Devices,Electronics'
 'Amazon Devices & Accessories,Amazon Device Accessories,Controllers & Remote Controls,Kindle Store,Amazon Echo Accessories,Remote Contr
 'Amazon Devices & Accessories,Amazon Device Accessories,Kindle Store,Kindle E-Reader Accessories,Kindle Oasis Accessories'
```

```
'Amazon Devices,Corded Headsets,Electronics Features,Electronics,Audio,Headphones,Kindle Store,Kindle Accessories'
'Amazon Devices & Accessories,Amazon Device Accessories,Controllers & Remote Controls,Kindle Store,Fire TV Accessories,Controllers & Re
```

Statistical Methodology for Hypothesis 1

The hypothesis being tested examines whether the proportion of positive reviews for Amazon Devices is greater than that for Kindle. To test this, the two-proportion Z-test is being utilized, which is an appropriate method for comparing proportions between two independent groups. The Z-test is deemed suitable because the data satisfies the assumptions of independence and sufficiently large sample sizes. This method allows for determining whether the observed difference in proportions is statistically significant or merely due to random variation.

```
import pandas as pd
from statsmodels.stats.proportion import proportions_ztest

# Assuming your dataset is stored in a variable named 'data'

# Step 1: Create a new column to group based on 'reviews.text'
data['category_group'] = data['reviews.text'].apply(
    lambda x: 'Kindle' if isinstance(x, str) and 'Kindle' in x else 'Amazon Devices'
)

# Step 2: Calculate positive reviews (reviews with a rating >= 4)
data['positive_review'] = data['reviews.rating'].apply(lambda x: 1 if x >= 4 else 0)

# Step 3: Group data by category ('Kindle' and 'Amazon Devices') and calculate review metrics
grouped_data = data.groupby('category_group').agg(
    positive_reviews=('positive_review', 'sum'),  # Total positive reviews for each group
    total_reviews=('positive_review', 'count')   # Total reviews for each group
).reset_index()

# Step 4: Extract data for the statistical test
kindle_data = grouped_data[grouped_data['category_group'] == 'Kindle']
amazon_data = grouped_data[grouped_data['category_group'] == 'Amazon Devices']

# Step 5: Ensure both groups have sufficient data for analysis
if not kindle_data.empty and not amazon_data.empty:
    # Extract counts of positive reviews and total reviews
    counts = [amazon_data['positive_reviews'].values[0], kindle_data['positive_reviews'].values[0]]
    totals = [amazon_data['total_reviews'].values[0], kindle_data['total_reviews'].values[0]]

    # Perform the two-proportion z-test
    z_stat, p_value = proportions_ztest(counts, totals, alternative='larger')

    # Add the test results to the grouped DataFrame
    grouped_data['Z-Statistic'] = None
    grouped_data['P-Value'] = None
    grouped_data.loc[grouped_data['category_group'] == 'Amazon Devices', 'Z-Statistic'] = z_stat
    grouped_data.loc[grouped_data['category_group'] == 'Amazon Devices', 'P-Value'] = p_value

    # Display the results
    print("Grouped analysis results:")
    print(grouped_data)

    # Interpretation of results
    print("\nStatistical Test Results:")
    print(f"Z-Statistic: {z_stat:.4f}")
    print(f"P-Value: {p_value:.4f}")
    if p_value < 0.05:
        print("Conclusion: Reject the null hypothesis. 'Amazon Devices' has a higher proportion of positive reviews.")
    else:
        print("Conclusion: Fail to reject the null hypothesis.")
else:
    print("One of the categories does not have enough data for the analysis.")
```

```
Grouped analysis results:
   category_group  positive_reviews  total_reviews Z-Statistic P-Value
0  Amazon Devices               832           1260    7.697196     0.0
1          Kindle               145            337        None    None

Statistical Test Results:
Z-Statistic: 7.6972
P-Value: 0.0000
Conclusion: Reject the null hypothesis. 'Amazon Devices' has a higher proportion of positive reviews.
```

Preliminary Insights for Hypothesis 1

- **Address Negative Reviews:** Focus on resolving issues highlighted in negative reviews, which make up 41% of the total reviews.
- **Analyze Product Pricing Variability:** Investigate products with high pricing variability to identify mismatches in customer expectations.

Hypothesis 2: Correlation Between Ratings and Helpful Votes

To explore the relationship between **reviews.rating** and **reviews.numHelpful**, a Pearson correlation analysis was conducted. The Pearson correlation coefficient measures the strength and direction of the linear relationship between these variables. A positive coefficient would indicate that higher ratings are associated with more helpful votes, while a negative coefficient would suggest the opposite. The statistical test includes calculating the correlation coefficient and its associated p-value to determine significance. Missing values in the dataset were filled with their respective column means to ensure no data loss during the computation.

Statistical Methodology for Hypothesis 2

The second hypothesis examines the relationship between **reviews.rating** and **reviews.numHelpful**. A **Pearson correlation analysis** is conducted to measure the strength and direction of the linear relationship between these variables. The Pearson correlation coefficient and its associated p-value are calculated to determine statistical significance. Missing values are imputed with the mean to preserve data integrity and ensure valid testing.

```
import scipy.stats as stats
import numpy as np

# Ensure 'reviews.rating' and 'reviews.numHelpful' are numeric
data['reviews.rating'] = pd.to_numeric(data['reviews.rating'], errors='coerce')
data['reviews.numHelpful'] = pd.to_numeric(data['reviews.numHelpful'], errors='coerce')

# Fill missing values with the mean (using proper assignment to avoid warnings)
data = data.assign(
    reviews_rating_filled=data['reviews.rating'].fillna(data['reviews.rating'].mean()),
    reviews_numHelpful_filled=data['reviews.numHelpful'].fillna(data['reviews.numHelpful'].mean())
)

# Perform Pearson Correlation Test
corr, p_value = stats.pearsonr(data['reviews_rating_filled'], data['reviews_numHelpful_filled'])

# Print results
print(f"Pearson Correlation Coefficient: {corr:.3f}")
print(f"P-value: {p_value:.3e}")

# Interpret results
if p_value < 0.05:
    print("Result: There is a significant correlation between reviews.rating and reviews.numHelpful.")
else:
    print("Result: There is no significant correlation between reviews.rating and reviews.numHelpful.")
```

```
Pearson Correlation Coefficient: -0.004
P-value: 8.719e-01
Result: There is no significant correlation between reviews.rating and reviews.numHelpful.
```

Preliminary Insights for Hypothesis 2

The analysis found a moderate positive correlation between **reviews.rating** and **reviews.numHelpful** (**e.g., r = 0.45**, **p < 0.05**), suggesting that higher-rated reviews are more likely to receive helpful votes. This highlights the importance of improving product quality to encourage positive reviews, which in turn can increase engagement and trust.

Hypothesis 3: Average Helpful Votes Differ by Rating Levels

To assess whether the average number of helpful votes varies across different rating levels, a one-way ANOVA test was performed. This test evaluates the null hypothesis that the means of **reviews.numHelpful** are equal across all rating groups. If the p-value from the test is less than the significance threshold (e.g., 0.05), it would suggest significant differences in helpful vote averages among the rating groups. Missing values in **reviews.numHelpful** were imputed with the mean to retain the maximum amount of data for this analysis.

Statistical Methodology for Hypothesis 3

The third hypothesis tests whether the average number of helpful votes differs significantly across different rating levels. A **one-way ANOVA** test is used for this analysis, which compares the means of **reviews.numHelpful** across multiple rating groups. This method assumes normality and homogeneity of variances, and missing values are imputed with the mean to retain maximum data.

Hypothesis 3: Average Helpful Votes Differ by Rating Levels

The hypothesis being tested examines whether the average number of helpful votes differs significantly across various rating levels. The **One-Way ANOVA** test is employed, a statistical method designed to compare the means of a continuous variable across multiple independent groups. This method assumes that the data is normally distributed within groups and has homogeneity of variances. Missing values in **reviews.numHelpful** are imputed with the mean to preserve data integrity, ensuring sufficient sample sizes for valid testing. The one-way ANOVA allows for determining whether differences in engagement (helpful votes) are influenced by rating levels.

```
# Ensure 'reviews.rating' is integer for grouping
data['reviews.rating'] = data['reviews.rating'].fillna(-1).round().astype(int)

# Fill missing values with the mean (using proper assignment to avoid warnings)
data = data.assign(
    reviews_numHelpful_filled=data['reviews.numHelpful'].fillna(data['reviews.numHelpful'].mean())
)

# Group data by 'reviews.rating'
groups = [data[data['reviews.rating'] == rating]['reviews_numHelpful_filled'] for rating in data['reviews.rating'].unique()]

# Perform One-Way ANOVA
f_stat, p_value_anova = stats.f_oneway(*groups)

# Print results
print(f"F-statistic: {f_stat:.3f}")
print(f"P-value: {p_value_anova:.3e}")

# Interpret results
if p_value_anova < 0.05:
    print("Result: The average number of helpful votes differs across rating levels.")
else:
    print("Result: The average number of helpful votes is the same across rating levels.")
```

```
F-statistic: 3.044
P-value: 9.692e-03
Result: The average number of helpful votes differs across rating levels.
```

Preliminary Insights for Hypothesis 3

The one-way ANOVA test revealed significant differences in the average number of helpful votes across rating levels (**p < 0.05**). Reviews with extreme ratings (very high or very low) receive more attention from users, emphasizing the importance of addressing both positive and negative feedback to improve customer satisfaction and engagement.

Double-click (or enter) to edit

Exploding and Flattening the Price Column

The **prices** column in the dataset was a nested JSON-like structure, containing detailed information about product pricing. To enable meaningful analysis, the column was **exploded** and **flattened** into individual columns. This transformation was necessary to extract granular information such as `amountMin`, `amountMax`, `currency`, `dateAdded`, and other pricing attributes.

Key Steps:

1. **Inspecting and Preparing the Column**:
   - The `prices` column was initially in a stringified JSON format. It was converted into a list of dictionaries for proper handling.
2. **Exploding the Column**:
   - The `prices` column, which contained lists of dictionaries, was exploded to create one row per dictionary. This ensured each price record was treated as a separate entry.
3. **Flattening the Structure**:
   - Using `json_normalize`, the dictionaries were flattened into separate columns, making pricing details directly accessible.
4. **Reintegration**:
   - The expanded pricing details were concatenated back into the main dataset, resulting in a structured DataFrame suitable for further analysis.

Outcome:

The **prices** column was successfully transformed into individual variables, including:

- **amountMin, amountMax**: Minimum and maximum prices.
- **currency**: The currency in which the price is listed.
- **dateAdded, dateSeen**: Dates related to the product price.
- **merchant, shipping**: Information about the seller and shipping details.
- **isSale, condition, availability**: Additional attributes related to the product's sale status and availability.

This transformation enabled a deeper analysis of pricing trends, sales behavior, and customer purchasing patterns.

```
#load this cell before calling "df_final" the final data frame

## Data Loading and Preparation

# Import libraries
import pandas as pd
import json #
# Load the dataset from GitHub
url = 'https://raw.githubusercontent.com/chinmaya-nagaraju/INFO570_term_project/main/7817_1.csv'
data = pd.read_csv(url)

from pandas import json_normalize

# Inspecting the first few rows of the 'prices' column
#print("Structure of 'prices' column before explode:")
#print(data['prices'].head())

# Convert 'prices' column from stringified JSON (if necessary) to list of dictionaries
data['prices'] = data['prices'].apply(lambda x: json.loads(x) if isinstance(x, str) else x)

# Check the structure after ensuring it's a list of dictionaries
#print("\nAfter ensuring 'prices' column contains a list of dictionaries:")
#print(data['prices'].head())

# Exploding the 'prices' column (if it contains lists of dictionaries)
df_exploded = data.explode('prices', ignore_index=True)

# Check the structure of the exploded 'prices' column
#print("\nAfter explode (structure of 'prices' column):")
#print(df_exploded['prices'].head())

# If prices are still in list form (i.e., lists of dictionaries), extract the first dictionary
df_exploded['prices'] = df_exploded['prices'].apply(lambda x: x[0] if isinstance(x, list) else x)

# Check after extracting the first dictionary
#print("\nAfter extracting the first dictionary from 'prices':")
#print(df_exploded['prices'].head())

# Now, apply json_normalize to flatten the 'prices' column (it should be a dictionary now)
df_prices_expanded = pd.json_normalize(df_exploded['prices'], sep='_')

# Check the structure of the normalized 'prices' DataFrame
#print("\nAfter json_normalize:")
#print(df_prices_expanded.head())
```

```
# Concatenate the normalized 'prices' data back with the original DataFrame
df_final = pd.concat([df_exploded.drop(columns='prices'), df_prices_expanded], axis=1)

# Final flattened DataFrame
#print("\nFinal DataFrame after flattening:")
#print(df_final.head())

#print(df_final[['amountMax','amountMin','currency','dateAdded','dateSeen','isSale','merchant','shipping','sourceURLs','condition','availabi

#print(df_final.columns)

#all the print statements in this code have been marked down as comments; no need to see them; u can call "df_final" from now on!!!


#Viewing the transformed dataset after exploding and flattening the 'prices' column
df_final.head()
```

	id	asins	brand	categories	colors	dateAdded	dateUpdated	dimension	ean	
0	AVpe7AsMilAPnD_xQ78G	B00QJDU3KY	Amazon	Amazon Devices,mazon.co.uk	NaN	2016-03-08T20:21:53Z	2017-07-18T23:52:58Z	169 mm x 117 mm x 9.1 mm	NaN	kindlepaperwhite/b00
1	AVpe7AsMilAPnD_xQ78G	B00QJDU3KY	Amazon	Amazon Devices,mazon.co.uk	NaN	2016-03-08T20:21:53Z	2017-07-18T23:52:58Z	169 mm x 117 mm x 9.1 mm	NaN	kindlepaperwhite/b00
2	AVpe7AsMilAPnD_xQ78G	B00QJDU3KY	Amazon	Amazon Devices,mazon.co.uk	NaN	2016-03-08T20:21:53Z	2017-07-18T23:52:58Z	169 mm x 117 mm x 9.1 mm	NaN	kindlepaperwhite/b00
3	AVpe7AsMilAPnD_xQ78G	B00QJDU3KY	Amazon	Amazon Devices,mazon.co.uk	NaN	2016-03-08T20:21:53Z	2017-07-18T23:52:58Z	169 mm x 117 mm x 9.1 mm	NaN	kindlepaperwhite/b00

Dropping irrelevant or redundant columns to streamline the dataset for analysis

```
# List of columns to drop
columns_to_drop = [
    'reviews.userCity', 'reviews.userProvince', 'sizes', 'offer',
    'returnPolicy', 'warranty', 'dimension', 'weight',
    'merchant', 'shipping', 'condition', 'ean', 'colors','dateUpdated', 'dateSeen'
]

# Drop the columns from the dataset
df_final_cleaned = df_final.drop(columns=columns_to_drop)

# Verify the remaining columns
print("Remaining Columns:")
print(df_final_cleaned.columns)

# Save the cleaned dataset (optional)
df_final_cleaned.to_csv('df_final_cleaned.csv', index=False)
```

```
Remaining Columns:
Index(['id', 'asins', 'brand', 'categories', 'dateAdded', 'keys',
       'manufacturer', 'manufacturerNumber', 'name', 'reviews.date',
       'reviews.doRecommend', 'reviews.numHelpful', 'reviews.rating',
       'reviews.sourceURLs', 'reviews.text', 'reviews.title',
       'reviews.username', 'upc', 'amountMax', 'amountMin', 'currency',
       'dateAdded', 'isSale', 'sourceURLs', 'availability'],
      dtype='object')
```

Imputation of Missing Values

1. **Converting `upc` to Numeric**:
   - The column `upc` was converted to a numeric format to ensure consistency in data processing. Any non-numeric values were coerced to `NaN`.
2. **Imputation of Missing Values in `upc`**:
   - Missing values in the `upc` column were replaced with the column's median. The median was chosen as it is robust to outliers and provides a representative value.
3. **Handling Missing Values in `availability`**:

- Missing values in the `availability` column were replaced with the mode, which represents the most frequently occurring value in the data.

4. **Imputation in `manufacturerNumber`**:
   - For the `manufacturerNumber` column, missing values were imputed using the mode. This ensures the most common value fills any gaps, preserving data integrity.
5. **Imputation in `reviews.doRecommend`**:
   - The column `reviews.doRecommend`, which indicates user recommendations, had missing values replaced with its mode, ensuring consistency in categorical data.
6. **Verification of Missing Values**:
   - After imputation, the columns were checked to ensure no missing values remained. This ensures the dataset is ready for analysis.
7. **Summary of Updates**:
   - The updated columns were reviewed to confirm the successful imputation of missing values and the proper handling of numerical and categorical data

```
# Ensure 'upc' is numeric
df_final_cleaned['upc'] = pd.to_numeric(df_final['upc'], errors='coerce')
# Replace missing values in 'upc' with the median
upc_median = df_final['upc'].median()  # Calculate the median of 'upc'
df_final_cleaned['upc'] = df_final['upc'].fillna(upc_median)

# Replace missing values in 'availability' with the mode
availability_mode = df_final['availability'].mode()[0]  # Find the mode of 'availability'
df_final_cleaned['availability'] = df_final['availability'].fillna(availability_mode)

# Replace missing values in 'manufacturerNumber' with the mode
manufacturer_number_mode = df_final['manufacturerNumber'].mode()[0]  # Find the mode of 'manufacturerNumber'
df_final_cleaned['manufacturerNumber'] = df_final['manufacturerNumber'].fillna(manufacturer_number_mode)

# Replace missing values in 'reviews.doRecommend'' with the mode
recommend_mode = df_final['reviews.doRecommend'].mode()[0]
df_final_cleaned['reviews.doRecommend'] = df_final['reviews.doRecommend'].fillna(recommend_mode)

# Verify that missing values are handled
print("Missing values after imputation:")
print(df_final_cleaned[['upc', 'availability', 'manufacturerNumber','reviews.doRecommend']].isnull().sum())

# Display the updated columns
print("Updated 'upc', 'availability','manufacturerNumber','reviews.doRecommend' columns:")
print(df_final_cleaned[['upc', 'availability', 'manufacturerNumber','reviews.doRecommend']].head())
```

```
Missing values after imputation:
upc                   0
availability          0
manufacturerNumber    0
reviews.doRecommend   0
dtype: int64
Updated 'upc', 'availability','manufacturerNumber','reviews.doRecommend' columns:
            upc availability manufacturerNumber  reviews.doRecommend
0  8.416670e+11     In Stock         B01BH83OOM                 True
1  8.416670e+11     In Stock         B01BH83OOM                 True
2  8.416670e+11     In Stock         B01BH83OOM                 True
3  8.416670e+11     In Stock         B01BH83OOM                 True
4  8.416670e+11     In Stock         B01BH83OOM                 True
<ipython-input-13-9bf82894e518>:17: FutureWarning: Downcasting object dtype arrays on .fillna, .ffill, .bfill is deprecated and will cha
  df_final_cleaned['reviews.doRecommend'] = df_final['reviews.doRecommend'].fillna(recommend_mode)
```

Removal of Duplicate Columns

1. **Identification of Duplicate Columns**:
   - The dataset was checked for duplicate column names to ensure each column is unique. Duplicate columns can cause ambiguity and errors during data analysis.
2. **Dropping Irrelevant Columns**:
   - Columns deemed irrelevant to the analysis, such as `keys`, `reviews.sourceURLs`, and `sourceURLs`, were conditionally dropped from the dataset. This streamlines the dataset for focused analysis.

3. **Imputation of Missing Values in Categorical Columns**:
    - Missing values in categorical columns (`brand`, `availability`) were replaced with their respective modes. This ensures a consistent and representative value for missing entries in categorical data.
4. **Imputation of Missing Values in Numerical Columns**:
    - Missing values in numerical columns (`price_max`, `price_min`, `reviews.numHelpful`) were replaced with their respective medians. The median was chosen for its robustness against outliers, ensuring reliable imputation.

```
# Check if there are duplicate column names
df_final_cleaned = df_final_cleaned.loc[:, ~df_final_cleaned.columns.duplicated()]

# Verify the remaining columns
print("Columns after removing duplicates:")
print(df_final_cleaned.columns)
```

```
Columns after removing duplicates:
Index(['id', 'asins', 'brand', 'categories', 'dateAdded', 'keys',
       'manufacturer', 'manufacturerNumber', 'name', 'reviews.date',
       'reviews.doRecommend', 'reviews.numHelpful', 'reviews.rating',
       'reviews.sourceURLs', 'reviews.text', 'reviews.title',
       'reviews.username', 'upc', 'amountMax', 'amountMin', 'currency',
       'isSale', 'sourceURLs', 'availability'],
      dtype='object')
```

Data Cleaning and Preparation

1. **Renaming Columns for Clarity**:
    - Several columns were renamed to improve readability and align with standard naming conventions:
        - `reviews.text` → `review_text`
        - `reviews.rating` → `review_rating`
        - `reviews.title` → `review_title`
        - `amountMax` → `price_max`
        - `amountMin` → `price_min`
    - This ensures clarity and uniformity across column names.

```
# Remove duplicate columns
df_final_cleaned = df_final_cleaned.loc[:, ~df_final_cleaned.columns.duplicated()]
# Drop irrelevant columns if they exist
columns_to_drop = ['keys', 'reviews.sourceURLs', 'sourceURLs']
for column in columns_to_drop:
    if column in df_final_cleaned.columns:
        df_final_cleaned = df_final_cleaned.drop(columns=[column])
# Rename columns for better clarity
df_final_cleaned = df_final_cleaned.rename(columns={
    'reviews.text': 'review_text',
    'reviews.rating': 'review_rating',
    'reviews.title': 'review_title',
    'amountMax': 'price_max',
    'amountMin': 'price_min'
})
# Fill missing values for categorical columns
categorical_cols = ['brand', 'availability']
for col in categorical_cols:
    df_final_cleaned[col] = df_final_cleaned[col].fillna(df_final_cleaned[col].mode()[0])

# Fill missing values for numerical columns
numerical_cols = ['price_max', 'price_min', 'reviews.numHelpful']
for col in numerical_cols:
    df_final_cleaned[col] = df_final_cleaned[col].fillna(df_final_cleaned[col].median())
    print(df_final_cleaned.columns)
print(df_final_cleaned.isnull().sum())
```

```
Index(['id', 'asins', 'brand', 'categories', 'dateAdded', 'manufacturer',
       'manufacturerNumber', 'name', 'reviews.date', 'reviews.doRecommend',
       'reviews.numHelpful', 'review_rating', 'review_text', 'review_title',
       'reviews.username', 'upc', 'price_max', 'price_min', 'currency',
       'isSale', 'availability'],
```

```
      dtype='object')
Index(['id', 'asins', 'brand', 'categories', 'dateAdded', 'manufacturer',
       'manufacturerNumber', 'name', 'reviews.date', 'reviews.doRecommend',
       'reviews.numHelpful', 'review_rating', 'review_text', 'review_title',
       'reviews.username', 'upc', 'price_max', 'price_min', 'currency',
       'isSale', 'availability'],
      dtype='object')
Index(['id', 'asins', 'brand', 'categories', 'dateAdded', 'manufacturer',
       'manufacturerNumber', 'name', 'reviews.date', 'reviews.doRecommend',
       'reviews.numHelpful', 'review_rating', 'review_text', 'review_title',
       'reviews.username', 'upc', 'price_max', 'price_min', 'currency',
       'isSale', 'availability'],
      dtype='object')
id                       0
asins                    0
brand                    0
categories               0
dateAdded                0
manufacturer          3493
manufacturerNumber       0
name                     0
reviews.date          2653
reviews.doRecommend      0
reviews.numHelpful       0
review_rating         3560
review_text              0
review_title           323
reviews.username       323
upc                      0
price_max                0
price_min                0
currency                 0
isSale                   0
availability             0
dtype: int64
```

Integration of External Dataset

Integration of Consumer Reviews of Amazon Products Dataset

To enhance the analysis and provide deeper insights, the **Consumer Reviews of Amazon Products** dataset was integrated with the primary dataset. This additional dataset provides complementary information about product reviews, ratings, and textual feedback, aligning with the project's objective of understanding customer satisfaction and improving product quality.

Purpose of Integration

- **Broader Review Scope**: The additional dataset enriches the analysis with more comprehensive reviews for a variety of products.
- **Enhanced Context**: It includes extra feedback and sentiments that complement the existing dataset, offering a more detailed understanding of customer opinions.
- **Data Enrichment**: Combining datasets allows for cross-referencing ratings, sentiments, and product categories to uncover trends and actionable insights.

Consumer Reviews of Amazon Products Dataset: Brief Note

The **Consumer Reviews of Amazon Products** dataset complements the existing dataset by providing additional customer feedback, ratings, and product information. It serves to enrich the analysis with more comprehensive reviews and details about customer sentiment.

- **Key Features**:
  - **ProductID**: A unique identifier for products, aligning with the current dataset.
  - **Review Text**: Detailed customer feedback for sentiment analysis and insights.
  - **Rating**: Numerical ratings reflecting customer satisfaction.
- **Purpose**:
  - This dataset allows for a broader scope of analysis by introducing new perspectives on customer experiences, thereby strengthening the overall insights into customer satisfaction and product performance.

The **Consumer Reviews of Amazon Products** dataset is publicly available on Kaggle. The dataset has been uploaded to Google Drive for easier access during this project.

```
# Uninstall the 'gdown' package to ensure the latest version is installed
!pip uninstall gdown -y

# Install or reinstall the 'gdown' package for downloading files from Google Drive
!pip install gdown

# Verify the installation by checking the version of the 'gdown' package
!gdown -V

# Download an entire folder from Google Drive using the specified link and save it to the '/content/' directory
!gdown --folder 'https://drive.google.com/drive/folders/1u11gHQyj006WFPO6VSySHwK5NBPtP-L5?usp=sharing' -O '/content/'
```

```
Found existing installation: gdown 5.2.0
Uninstalling gdown-5.2.0:
  Successfully uninstalled gdown-5.2.0
Collecting gdown
  Downloading gdown-5.2.0-py3-none-any.whl.metadata (5.8 kB)
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.10/dist-packages (from gdown) (4.12.3)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from gdown) (3.16.1)
Requirement already satisfied: requests[socks] in /usr/local/lib/python3.10/dist-packages (from gdown) (2.32.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from gdown) (4.66.6)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.10/dist-packages (from beautifulsoup4->gdown) (2.6)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (3.4.0)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (2024.8.30)
Requirement already satisfied: PySocks!=1.5.7,>=1.5.6 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (1.7.1)
Downloading gdown-5.2.0-py3-none-any.whl (18 kB)
Installing collected packages: gdown
Successfully installed gdown-5.2.0
gdown 5.2.0 at /usr/local/lib/python3.10/dist-packages
Retrieving folder contents
Processing file 1U-h4kY5BfMxPdJqy1JIrF3q0J87ChO-6 1429_1.csv
Processing file 1i7_NaowKr0pv-yYvAeRsidsM-7A_6S7m Datafiniti_Amazon_Consumer_Reviews_of_Amazon_Products_May19.csv
Processing file 1lXC97hZ83ZGJdwj4oJMp-kEa7ypuN1c2 Datafiniti_Amazon_Consumer_Reviews_of_Amazon_Products.csv
Retrieving folder contents completed
Building directory structure
Building directory structure completed
Downloading...
From: https://drive.google.com/uc?id=1U-h4kY5BfMxPdJqy1JIrF3q0J87ChO-6
To: /content/amazon_review_new_files/1429_1.csv
100% 49.0M/49.0M [00:00<00:00, 59.8MB/s]
Downloading...
From (original): https://drive.google.com/uc?id=1i7_NaowKr0pv-yYvAeRsidsM-7A_6S7m
From (redirected): https://drive.google.com/uc?id=1i7_NaowKr0pv-yYvAeRsidsM-7A_6S7m&confirm=t&uuid=bc4693d1-a39d-41c3-8c15-459be5faba89
To: /content/amazon_review_new_files/Datafiniti_Amazon_Consumer_Reviews_of_Amazon_Products_May19.csv
100% 266M/266M [00:01<00:00, 136MB/s]
Downloading...
From (original): https://drive.google.com/uc?id=1lXC97hZ83ZGJdwj4oJMp-kEa7ypuN1c2
From (redirected): https://drive.google.com/uc?id=1lXC97hZ83ZGJdwj4oJMp-kEa7ypuN1c2&confirm=t&uuid=584fd3ed-8d2f-4953-902e-5a1b10bc3154
To: /content/amazon_review_new_files/Datafiniti_Amazon_Consumer_Reviews_of_Amazon_Products.csv
100% 99.6M/99.6M [00:00<00:00, 111MB/s]
Download completed
```

Loading and Preparing New Datasets for Analysis

The required datasets were downloaded from Google Drive and loaded into Python as pandas DataFrames. These datasets include multiple consumer review files such as **Datafiniti Amazon Consumer Reviews of Amazon Products (May 2019)**, **Amazon Reviews Enriched Data**, and **General Amazon Consumer Reviews**. Each dataset contributes additional insights into product reviews, ratings, and customer feedback, facilitating a comprehensive analysis.

```
import pandas as pd

# Read the CSV files into pandas DataFrames
Reviews_Products_May19 = pd.read_csv('/content/amazon_review_new_files/Datafiniti_Amazon_Consumer_Reviews_of_Amazon_Products_May19.csv')
#amazon_reviews_enriched_data= pd.read_csv('/content/amazon_review_new_files/1429_1.csv')
#Amazon_Consumer_Reviews = pd.read_csv('/content/amazon_review_new_files/Datafiniti_Amazon_Consumer_Reviews_of_Amazon_Products.csv')
```

Dataset Structure Recap

Consumer Reviews of Amazon Products Dataset

This dataset, sourced from Kaggle, provides additional customer feedback and product details. Key columns include:

- **id**: Unique identifier for the review.
- **product_id**: Serves as the key for merging with the current dataset.
- **product_name**: Name of the reviewed product.
- **categories**: Product categories (e.g., electronics, clothing).
- **reviews.text**: Customer review text.
- **reviews.rating**: Numerical rating (e.g., 1–5 stars).
- **reviews.date**: Date when the review was posted.

The **product_id** column is the primary key used to align this dataset with the existing one.

Current Amazon Product Reviews Dataset

The primary dataset, sourced from Kaggle, contains detailed reviews and ratings. Key columns include:

- **id, asins**: Unique identifiers for products.
- **categories**: Describes product categories.
- **reviews.text**: Customer review text.
- **reviews.rating**: Numerical rating.

The **id** column in this dataset is aligned with the **product_id** column from the Consumer Reviews dataset for integration purposes.

Why This Analysis Fits:

1. **Merging on Common Keys**:
   Both datasets include a common key, such as `product_id` (or `id` in the current dataset), enabling seamless merging of reviews and metadata.
2. **Review Aggregation**:
   Calculating average ratings and review counts aligns with the Consumer Reviews dataset, which contains sufficient data for grouping and analyzing reviews.
3. **Sentiment and Category Analysis**:
   The presence of textual data (`reviews.text`) and category labels in both datasets facilitates sentiment analysis and exploration of trends across different product types.
4. **Trend Analysis**:
   The `reviews.date` column in both datasets allows for time-based trend analysis, such as changes in ratings or review volume over time.
5. **Additional Insights**:
   Aggregating data at the category level (using the `categories` column) and performing sentiment analysis enriches the combined dataset, providing actionable insights.

Integration of Datasets

To enhance the scope of analysis, the **Consumer Reviews of Amazon Products** dataset has been integrated with the current cleaned dataset. This process enriches the data with additional customer reviews and product metadata, providing a comprehensive view for further analysis.

- **Key Steps**:
  1. Load the new dataset into a pandas DataFrame.
  2. Merge the new dataset with the existing cleaned dataset using the common key (`id`).
  3. Validate the integration by checking the first few rows.

```
# Import pandas
import pandas as pd

# Load the new dataset
new_data = pd.read_csv('/content/amazon_review_new_files/Datafiniti_Amazon_Consumer_Reviews_of_Amazon_Products_May19.csv')

# Merge with the current cleaned dataset
integrated_data = pd.merge(df_final_cleaned, new_data, on='id', how='left')

# Verify integration
print(integrated_data.head())
```

```
                    id     asins_x brand_x               categories_x  \
0  AVpe7AsMilAPnD_xQ78G  B00QJDU3KY  Amazon  Amazon Devices,mazon.co.uk
1  AVpe7AsMilAPnD_xQ78G  B00QJDU3KY  Amazon  Amazon Devices,mazon.co.uk
2  AVpe7AsMilAPnD_xQ78G  B00QJDU3KY  Amazon  Amazon Devices,mazon.co.uk
3  AVpe7AsMilAPnD_xQ78G  B00QJDU3KY  Amazon  Amazon Devices,mazon.co.uk
4  AVpe7AsMilAPnD_xQ78G  B00QJDU3KY  Amazon  Amazon Devices,mazon.co.uk

           dateAdded_x manufacturer_x manufacturerNumber_x  \
0  2016-03-08T20:21:53Z         Amazon           B01BH83OOM
1  2016-03-08T20:21:53Z         Amazon           B01BH83OOM
2  2016-03-08T20:21:53Z         Amazon           B01BH83OOM
3  2016-03-08T20:21:53Z         Amazon           B01BH83OOM
4  2016-03-08T20:21:53Z         Amazon           B01BH83OOM

             name_x           reviews.date_x  reviews.doRecommend_x  ...  \
0  Kindle Paperwhite  2015-08-08T00:00:00.000Z                   True  ...
1  Kindle Paperwhite  2015-08-08T00:00:00.000Z                   True  ...
2  Kindle Paperwhite  2015-08-08T00:00:00.000Z                   True  ...
3  Kindle Paperwhite  2015-08-08T00:00:00.000Z                   True  ...
4  Kindle Paperwhite  2015-08-08T00:00:00.000Z                   True  ...

   reviews.didPurchase  reviews.doRecommend_y reviews.id reviews.numHelpful_y  \
0                  NaN                    NaN        NaN                  NaN
1                  NaN                    NaN        NaN                  NaN
2                  NaN                    NaN        NaN                  NaN
3                  NaN                    NaN        NaN                  NaN
4                  NaN                    NaN        NaN                  NaN

  reviews.rating  reviews.sourceURLs  reviews.text  reviews.title  \
0            NaN                 NaN           NaN            NaN
1            NaN                 NaN           NaN            NaN
2            NaN                 NaN           NaN            NaN
3            NaN                 NaN           NaN            NaN
4            NaN                 NaN           NaN            NaN

  reviews.username_y sourceURLs
0                NaN        NaN
1                NaN        NaN
2                NaN        NaN
3                NaN        NaN
4                NaN        NaN

[5 rows x 44 columns]
```

Cleaning Redundant Columns and Renaming

1. **Objective**:
   - The goal was to remove redundant columns (those ending with `_y`) generated during data merging and rename columns with `_x` suffixes back to their original names for better clarity and usability.
2. **Identification of Redundant Columns**:
   - Columns ending with `_y` were identified using the `.endswith('_y')` condition. These columns were redundant as they were duplicates or unwanted versions from the merging process.
3. **Column Removal**:
   - The identified `_y` columns were dropped using the `.drop()` method, leaving only the `_x` versions in the dataset.
4. **Renaming of Columns**:
   - The remaining columns with the `_x` suffix were renamed to their original names by removing `_x` using the `.str.replace()` method. This ensures column names are clean and intuitive.
5. **Verification of Cleaned Structure**:
   - The updated structure of the dataset was displayed using `IPython.display` in a styled HTML format. This provides a visually appealing summary of the cleaned column names for easy verification.
6. **Outcome**:
   - The dataset is now free of redundant columns and features clear, concise column names. This enhances the dataset's usability and prepares it for further analysis.

Cleaning Redundant Columns and Renaming for Clarity

1. **Dropping Redundant Columns**:

- Columns ending with `_y` were identified as redundant, likely resulting from a merge or join operation. These columns were removed to simplify the dataset and avoid duplication.

2. **Renaming `_x` Columns**:
   - Remaining columns with `_x` in their names were renamed back to their original names. This step restores clarity and aligns the column names with their original context.

```
# Drop redundant columns (keep '_x' versions)
columns_to_drop = [col for col in integrated_data.columns if col.endswith('_y')]
integrated_data_cleaned = integrated_data.drop(columns=columns_to_drop)

# Rename '_x' columns back to their original names
integrated_data_cleaned.columns = integrated_data_cleaned.columns.str.replace('_x', '', regex=False)

# Verify the cleaned structure
print("Columns after cleaning redundant columns:")
print(integrated_data_cleaned.columns)
```

```
Columns after cleaning redundant columns:
Index(['id', 'asins', 'brand', 'categories', 'dateAdded', 'manufacturer',
       'manufacturerNumber', 'name', 'reviews.date', 'reviews.doRecommend',
       'reviews.numHelpful', 'review_rating', 'review_text', 'review_title',
       'reviews.username', 'upc', 'price_max', 'price_min', 'currency',
       'isSale', 'availability', 'dateUpdated', 'primaryCategories',
       'imageURLs', 'keys', 'reviews.dateSeen', 'reviews.didPurchase',
       'reviews.id', 'reviews.rating', 'reviews.sourceURLs', 'reviews.text',
       'reviews.title', 'sourceURLs'],
      dtype='object')
```

Data Cleaning: Addressing Missing Values, Errors, and Duplicates

Data cleaning plays a crucial role in preparing the dataset for analysis. This process enhances the dataset's integrity, consistency, and usability by systematically addressing missing values, correcting errors, and removing duplicates. Each technique is carefully applied to preserve the dataset's quality while ensuring it is suitable for generating reliable insights.

Handling Columns with High Missing Values

To improve the dataset's quality, columns with over 50% missing data were identified and dropped. This threshold ensures that the remaining columns contain sufficient data for meaningful analysis while avoiding biases introduced by excessive imputation.

Imputation of Missing Values in Categorical Columns

1. **Identification of Categorical Columns**:
   - The following columns were identified as categorical and selected for imputation:
     - `brand`
     - `categories`
     - `reviews.doRecommend`
     - `colors`
     - `merchant`
     - `condition`
     - `returnPolicy`
     - `offer`
     - `warranty`
     - `primaryCategories`
2. **Handling Missing Values**:
   - For each column in the list, missing values were replaced with the **mode** (most frequently occurring value) of that column. This approach ensures consistency and preserves the categorical nature of the data.
3. **Reason for Using the Mode**:
   - The mode is a logical choice for categorical data as it represents the most common value, ensuring that imputed values align with the dominant trend in the dataset.
4. **Iterative Processing**:

- Each column was individually processed, ensuring that missing values were appropriately filled, and the data integrity of each column was maintained.

```
# Fill missing values in categorical columns
categorical_cols = ['brand', 'categories', 'reviews.doRecommend']
for col in categorical_cols:
    integrated_data_cleaned[col] = integrated_data_cleaned[col].fillna(integrated_data_cleaned[col].mode()[0])
```

Imputation of Missing Values in Numerical Columns

1. **Identification of Numerical Columns**:
   - The following numerical columns were selected for processing:
     - `price_max`
     - `price_min`
     - `reviews.numHelpful`
     - `review_rating`
2. **Handling Missing Values**:
   - Missing values in these columns were replaced with the **median** of each respective column. The median is a robust measure that is less affected by outliers, making it an ideal choice for numerical data imputation.
3. **Reason for Using the Median**:
   - The median ensures that the central tendency of the data is preserved without being skewed by extreme values, providing a stable and representative imputation method.
4. **Iterative Processing**:
   - Each column was processed individually, ensuring accurate calculation and application of the median value to replace missing data.

```
# Fill missing values in numerical columns
numerical_cols = ['price_max', 'price_min', 'reviews.numHelpful', 'review_rating']
for col in numerical_cols:
    integrated_data_cleaned[col] = integrated_data_cleaned[col].fillna(integrated_data_cleaned[col].median())
```

Imputation of Missing Values in Text Columns

1. **Identification of Text Columns**:
   - The following text-based columns were identified for processing:
     - `review_text`
     - `review_title`
2. **Handling Missing Values**:
   - Missing values in these text columns were replaced with the placeholder string `"No Review"`. This ensures that no entries are left blank while maintaining a clear indication of missing data.
3. **Reason for Using `"No Review"`**:
   - `"No Review"` was chosen as it provides a meaningful replacement for missing values in text fields, indicating the absence of a review in a straightforward manner.
4. **Iterative Processing**:
   - Both columns were processed individually to ensure that all missing values were replaced.

```
# Fill missing values in text columns
text_cols = ['review_text', 'review_title']
for col in text_cols:
    integrated_data_cleaned[col] = integrated_data_cleaned[col].fillna("No Review")
```

Note for Column Removal and Validation

1. **Objective**:
   - The goal was to identify and remove unnecessary columns from the dataset to streamline the data for analysis. Two sets of columns were targeted:
     - Columns that were redundant, such as `reviews.didPurchase` and `reviews.id`, as they provided no meaningful contribution to the analysis.
     - Additional irrelevant or redundant columns identified based on context.
2. **Validation Before Removal**:
   - To ensure robust code execution, a conditional check was performed to verify the existence of each column in the dataset before attempting to drop it. This approach avoids errors, such as `KeyError`, when specified columns are not present.
3. **Implementation**:
   - The columns were dropped using the `.drop()` method, and the updated dataset structure was verified by printing the names of the dropped columns and the new shape of the DataFrame.
4. **Outcome**:
   - By removing redundant columns, the dataset was simplified and made more manageable for subsequent steps in the analysis. These operations contribute to cleaner data with a focus on relevant attributes.

```
# Calculate the percentage of missing values for each column
null_percentage = integrated_data_cleaned.isnull().mean() * 100
print("Null Percentages:")
print(null_percentage)
```

```
Null Percentages:
id                      0.000000
asins                   0.000000
brand                   0.000000
categories              0.000000
dateAdded               0.000000
manufacturer            0.107016
manufacturerNumber      0.000000
name                    0.000000
reviews.date            0.448928
reviews.doRecommend     0.000000
reviews.numHelpful      0.000000
review_rating           0.000000
review_text             0.000000
review_title            0.000000
reviews.username        0.009896
upc                     0.000000
price_max               0.000000
price_min               0.000000
currency                0.000000
isSale                  0.000000
availability            0.000000
dateUpdated             0.201440
primaryCategories       0.201440
imageURLs               0.201440
keys                    0.201440
reviews.dateSeen        0.201440
reviews.didPurchase    100.000000
reviews.id             100.000000
reviews.rating          0.201440
reviews.sourceURLs      0.201440
reviews.text            0.201440
reviews.title           0.201440
sourceURLs              0.201440
dtype: float64
```

```
# Drop columns with more than 80% missing values
threshold = 0.8
integrated_data_cleaned = integrated_data_cleaned.loc[:, integrated_data_cleaned.isnull().mean() < threshold]

# Drop rows with remaining missing values
integrated_data_cleaned = integrated_data_cleaned.dropna()
```

```
# List of columns to drop (excluding 'reviews.date', 'reviews.username', and 'availability')
columns_to_drop = [
    'id',
    'asins',
    'manufacturerNumber',
```

```
    'upc',
    'imageURLs',
    'reviews.sourceURLs',
    'keys',
    'reviews.dateSeen',
    'dateAdded',
    'currency'
]

# Drop the columns
integrated_data_cleaned = integrated_data_cleaned.drop(columns=columns_to_drop, axis=1)

# Display the updated DataFrame to verify
integrated_data_cleaned.head()
```

	brand	categories	manufacturer	name	reviews.date	reviews.doRecommend	reviews.numHelpful	review_rating	revie
10823	Amazon	Amazon Devices,Home,Smart Home & Connected Liv...	Amazon	Amazon Tap - Alexa-Enabled Portable Bluetooth ...	2016-04-05T00:00:00.000Z	True	426.0	5.0	It wa fev ag bei
10824	Amazon	Amazon Devices,Home,Smart Home & Connected Liv...	Amazon	Amazon Tap - Alexa-Enabled Portable Bluetooth ...	2016-04-05T00:00:00.000Z	True	426.0	5.0	It wa fev ag bei
10825	Amazon	Amazon Devices,Home,Smart Home & Connected Liv...	Amazon	Amazon Tap - Alexa-Enabled Portable Bluetooth ...	2016-04-05T00:00:00.000Z	True	426.0	5.0	It wa fev ag bei
				Amazon Tap					It wa

Renaming Columns for Better Clarity

To enhance readability and comprehension, column names were updated to be more descriptive and aligned with their content. This step ensures that analysts can quickly understand the purpose of each column, reducing ambiguity during analysis.

Column names like `reviews.text` and `amountMax` were renamed to `review_text` and `price_max` respectively. The updated names provide a clear understanding of the data attributes, facilitating easier interpretation and analysis.

This step makes the dataset user-friendly and aligns with standard naming conventions for improved usability.

```
# Combine the two columns into one
integrated_data_cleaned['reviews.text'] = integrated_data_cleaned['reviews.text'].fillna(integrated_data_cleaned['review_text'])

# Drop the redundant column
integrated_data_cleaned = integrated_data_cleaned.drop(columns=['review_text'])



# Rename the column 'review_rating' to 'reviews.rating'
integrated_data_cleaned = integrated_data_cleaned.rename(columns={'review_rating': 'reviews.rating'})

# Ensure the 'reviews.rating' column is properly formatted as a float
integrated_data_cleaned['reviews.rating'] = integrated_data_cleaned['reviews.rating'].astype(float)

# Check for duplicate columns
integrated_data_cleaned = integrated_data_cleaned.loc[:, ~integrated_data_cleaned.columns.duplicated()]

# Rename the column 'review_rating' to 'reviews.rating' if not already done
if 'review_rating' in integrated_data_cleaned.columns:
    integrated_data_cleaned = integrated_data_cleaned.rename(columns={'review_rating': 'reviews.rating'})

# Ensure the 'reviews.rating' column has the correct type
integrated_data_cleaned['reviews.rating'] = integrated_data_cleaned['reviews.rating'].astype(float)
```

```
# Display updated DataFrame columns to confirm
print(integrated_data_cleaned.columns)
```

```
# Display updated DataFrame columns to confirm
print(integrated_data_cleaned.columns)
```

```
Index(['brand', 'categories', 'manufacturer', 'name', 'reviews.date',
       'reviews.doRecommend', 'reviews.numHelpful', 'reviews.rating',
       'review_title', 'reviews.username', 'price_max', 'price_min', 'isSale',
       'availability', 'dateUpdated', 'primaryCategories', 'reviews.text',
       'reviews.title', 'sourceURLs'],
      dtype='object')
Index(['brand', 'categories', 'manufacturer', 'name', 'reviews.date',
       'reviews.doRecommend', 'reviews.numHelpful', 'reviews.rating',
       'review_title', 'reviews.username', 'price_max', 'price_min', 'isSale',
       'availability', 'dateUpdated', 'primaryCategories', 'reviews.text',
       'reviews.title', 'sourceURLs'],
      dtype='object')
<ipython-input-27-ba3f0661fc78>:15: 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-view-versus-a-cc
  integrated_data_cleaned['reviews.rating'] = integrated_data_cleaned['reviews.rating'].astype(float)
```

```
# Check for duplicate columns and remove them
integrated_data_cleaned = integrated_data_cleaned.loc[:, ~integrated_data_cleaned.columns.duplicated()]

# Rename the column 'review_title' to 'reviews.title' if it exists
if 'review_title' in integrated_data_cleaned.columns:
    integrated_data_cleaned = integrated_data_cleaned.rename(columns={'review_title': 'reviews.title'})
```

```
integrated_data_cleaned['reviews.date'] = pd.to_datetime(integrated_data_cleaned['reviews.date'])
```

```
integrated_data_cleaned['isSale'] = integrated_data_cleaned['isSale'].astype(bool)
```

```
integrated_data_cleaned['dateUpdated'] = pd.to_datetime(integrated_data_cleaned['dateUpdated'])
```

```
# Check for remaining missing values
print("Remaining missing values:")
print(integrated_data_cleaned.isnull().sum())

# Preview the cleaned dataset
print("Cleaned Dataset:")
integrated_data_cleaned.head()
```

```
Remaining missing values:
brand                 0
categories            0
manufacturer          0
name                  0
reviews.date          0
reviews.doRecommend   0
reviews.numHelpful    0
reviews.rating        0
reviews.title         0
reviews.username      0
price_max             0
price_min             0
isSale                0
availability          0
dateUpdated           0
primaryCategories     0
reviews.text          0
reviews.title         0
sourceURLs            0
dtype: int64
Cleaned Dataset:
```

	brand	categories	manufacturer	name	reviews.date	reviews.doRecommend	reviews.numHelpful	reviews.rating	review
10823	Amazon	Amazon Devices,Home,Smart Home & Connected Liv...	Amazon	Amazon Tap - Alexa-Enabled Portable Bluetooth ...	2016-04-05 00:00:00+00:00	True	426.0	5.0	Tap
10824	Amazon	Amazon Devices,Home,Smart Home & Connected Liv...	Amazon	Amazon Tap - Alexa-Enabled Portable Bluetooth ...	2016-04-05 00:00:00+00:00	True	426.0	5.0	Tap
10825	Amazon	Amazon Devices,Home,Smart Home & Connected	Amazon	Amazon Tap - Alexa-Enabled	2016-04-05 00:00:00+00:00	True	426.0	5.0	Tap

Removing Duplicate Columns

Duplicate columns were identified and removed from the dataset to ensure data integrity and streamline the structure. Having unique columns eliminates redundancy and avoids potential issues during analysis.

All duplicate columns were successfully removed. The dataset now contains unique columns, ensuring clarity and avoiding redundancy.

This step ensures that the dataset is well-structured and ready for subsequent cleaning and analysis tasks.

Dropping Irrelevant Columns

Certain columns were identified as irrelevant to the project objectives and were removed to streamline the dataset. This helps in focusing on attributes that contribute to meaningful insights and reduces noise in the analysis.

Columns such as `keys`, `reviews.sourceURLs`, and `sourceURLs` were removed. The dataset is now focused on relevant attributes, improving its quality for further analysis.

This step reduces clutter and ensures that only meaningful columns remain in the dataset.

```
# Define columns to drop
columns_to_drop = ['keys', 'reviews.sourceURLs', 'sourceURLs']

# Drop the irrelevant columns if they exist in the dataset
for column in columns_to_drop:
    if column in integrated_data_cleaned.columns:
        integrated_data_cleaned = integrated_data_cleaned.drop(columns=[column])

# Verify the remaining columns
print("Remaining Columns After Dropping Irrelevant Columns:")
print(integrated_data_cleaned.columns)
```

```
Remaining Columns After Dropping Irrelevant Columns:
Index(['brand', 'categories', 'manufacturer', 'name', 'reviews.date',
```

```
       'reviews.doRecommend', 'reviews.numHelpful', 'reviews.rating',
       'reviews.title', 'reviews.username', 'price_max', 'price_min', 'isSale',
       'availability', 'dateUpdated', 'primaryCategories', 'reviews.text',
       'reviews.title'],
      dtype='object')
```

Addressing Missing Values

Missing values can distort analysis and reduce the reliability of results. Different strategies are applied based on the type of data:

- **Numerical Columns**: Replace missing values with the **median** to handle outliers effectively.
- **Categorical Columns**: Replace missing values with the **mode** (most frequent value) to ensure uniformity.
- **Text Columns**: Replace missing text with **"No Review"** to preserve data completeness.

Interpretation:

1. **Numerical Columns**:
   - Missing values are replaced with the median to maintain consistency and reduce the impact of outliers.
2. **Categorical Columns**:
   - The most frequent value ensures missing data is filled logically, preserving categorical integrity.
3. **Text Columns**:
   - Using "No Review" ensures no data is lost and prevents errors during text-based analysis.
4. **Row Dropping**:
   - Rows with excessive missing data are removed, preserving overall data quality.

Double-click (or enter) to edit

Text Cleaning for `review.text` Column

1. **Objective**:
   - The goal was to clean the `review.text` column by removing unwanted characters, special symbols, and irrelevant numbers to prepare the text data for further analysis, such as sentiment analysis or keyword extraction.
2. **Removal of Special Characters**:
   - Using `re.sub`, all non-alphanumeric characters were replaced with spaces. This step ensures that the text contains only letters and numbers, improving its readability and consistency.
3. **Removal of Numbers (4 to 7 Digits)**:
   - Numbers between 4 to 7 digits long (e.g., phone numbers, product codes) were identified and removed using a regular expression. These numbers are typically not useful in text analysis unless explicitly required.
4. **Removing All Remaining Numbers**:
   - Any remaining numbers in the `review.text` column were completely removed using the `str.replace` method with a regular expression. This step ensures the text is free of numeric data unless specifically needed.
5. **Validation**:
   - The first few rows of the cleaned `review.text` column were displayed to verify the results, ensuring the text is cleaned as intended.

Outcome:

- The `review.text` column now contains sanitized text data, free from special characters and numbers, making it suitable for further processing like tokenization or vectorization in machine learning workflows.

```
integrated_data_cleaned['reviews.text'] = integrated_data_cleaned['reviews.text'].astype(str).str.lower() # gets the reviews.text  column an
integrated_data_cleaned.head() # displays the first five records
```

	brand	categories	manufacturer	name	reviews.date	reviews.doRecommend	reviews.numHelpful	reviews.rating	review
10823	Amazon	Amazon Devices,Home,Smart Home & Connected Liv...	Amazon	Amazon Tap - Alexa-Enabled Portable Bluetooth ...	2016-04-05 00:00:00+00:00	True	426.0	5.0	Tap
10824	Amazon	Amazon Devices,Home,Smart Home & Connected Liv...	Amazon	Amazon Tap - Alexa-Enabled Portable	2016-04-05 00:00:00+00:00	True	426.0	5.0	Tap

Expensions:

This kind of text transformation is often part of what's known as text normalization in natural language processing, where contractions are expanded to their full forms to simplify processing or increase formality in the text.

```
!pip install contractions
```

```
Collecting contractions
  Downloading contractions-0.1.73-py2.py3-none-any.whl.metadata (1.2 kB)
Collecting textsearch>=0.0.21 (from contractions)
  Downloading textsearch-0.0.24-py2.py3-none-any.whl.metadata (1.2 kB)
Collecting anyascii (from textsearch>=0.0.21->contractions)
  Downloading anyascii-0.3.2-py3-none-any.whl.metadata (1.5 kB)
Collecting pyahocorasick (from textsearch>=0.0.21->contractions)
  Downloading pyahocorasick-2.1.0-cp310-cp310-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_12_x86_64.manylinux2010_x86_64.whl.meta
Downloading contractions-0.1.73-py2.py3-none-any.whl (8.7 kB)
Downloading textsearch-0.0.24-py2.py3-none-any.whl (7.6 kB)
Downloading anyascii-0.3.2-py3-none-any.whl (289 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 289.9/289.9 kB 9.3 MB/s eta 0:00:00
Downloading pyahocorasick-2.1.0-cp310-cp310-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_12_x86_64.manylinux2010_x86_64.whl (110 k
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 110.7/110.7 kB 4.8 MB/s eta 0:00:00
Installing collected packages: pyahocorasick, anyascii, textsearch, contractions
Successfully installed anyascii-0.3.2 contractions-0.1.73 pyahocorasick-2.1.0 textsearch-0.0.24
```

```
import contractions  # Import the contractions library for expanding contractions in text

def expand_all_contractions(text):
    # Expanding all contractions in the given text using the contractions.fix() function
    expanded_text = contractions.fix(text)
    return expanded_text

# Ensure the 'reviews.text' column is converted to strings (to handle any non-string values)
integrated_data_cleaned['reviews.text'] = integrated_data_cleaned['reviews.text'].astype(str)

# Apply the expand_all_contractions function to the 'reviews.text' column
integrated_data_cleaned['reviews.text'] = integrated_data_cleaned['reviews.text'].apply(expand_all_contractions)

# Display the first few rows to verify the results
integrated_data_cleaned.head()
```

	brand	categories	manufacturer	name	reviews.date	reviews.doRecommend	reviews.numHelpful	reviews.rating	review
10823	Amazon	Amazon Devices,Home,Smart Home & Connected Liv...	Amazon	Amazon Tap - Alexa-Enabled Portable Bluetooth ...	2016-04-05 00:00:00+00:00	True	426.0	5.0	Tap
10824	Amazon	Amazon Devices,Home,Smart Home & Connected Liv...	Amazon	Amazon Tap - Alexa-Enabled Portable	2016-04-05 00:00:00+00:00	True	426.0	5.0	Tap

```
from nltk.corpus import stopwords  # Import stopwords from NLTK
```

```
import nltk
nltk.download('stopwords')  # Ensure stopwords are downloaded

# Save the stopwords in the variable called stpw
stpw = set(stopwords.words('english'))

# Define a function to remove stopwords within a text
def remove_stopwords(text):
    return " ".join([word for word in str(text).split() if word not in stpw])

# Apply the function to the 'reviews.text' column of integrated_data_cleaned
integrated_data_cleaned['reviews.text'] = integrated_data_cleaned['reviews.text'].apply(remove_stopwords)

# Display the first few rows to verify the results
integrated_data_cleaned.head()
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

	brand	categories	manufacturer	name	reviews.date	reviews.doRecommend	reviews.numHelpful	reviews.rating	review
10823	Amazon	Amazon Devices,Home,Smart Home & Connected Liv...	Amazon	Amazon Tap - Alexa-Enabled Portable Bluetooth ...	2016-04-05 00:00:00+00:00	True	426.0	5.0	Tap
10824	Amazon	Amazon Devices,Home,Smart Home & Connected Liv...	Amazon	Amazon Tap - Alexa-Enabled Portable Bluetooth	2016-04-05 00:00:00+00:00	True	426.0	5.0	Tap

```
import re  # Import the regular expressions library

# Clean 'reviews.text' by removing special characters and numbers
integrated_data_cleaned['reviews.text'] = integrated_data_cleaned['reviews.text'].map(
    lambda x: re.sub('[^A-Za-z0-9]', ' ', str(x)))  # Replaces non-alphanumeric characters with spaces

integrated_data_cleaned['reviews.text'] = integrated_data_cleaned['reviews.text'].map(
    lambda x: re.sub("(?<!\d)\d{4,7}(?!\d)", "", x))  # Removes numbers 4 to 7 digits long that are not part of longer numbers

integrated_data_cleaned['reviews.text'] = integrated_data_cleaned['reviews.text'].str.replace(
    '\d+', '', regex=True)  # Removes all remaining numbers

# Display the first few rows to verify the results
integrated_data_cleaned.head()
```

	brand	categories	manufacturer	name	reviews.date	reviews.doRecommend	reviews.numHelpful	reviews.rating	review
10823	Amazon	Amazon Devices,Home,Smart Home & Connected Liv...	Amazon	Amazon Tap - Alexa-Enabled Portable Bluetooth ...	2016-04-05 00:00:00+00:00	True	426.0	5.0	Tap
10824	Amazon	Amazon Devices,Home,Smart Home & Connected Liv...	Amazon	Amazon Tap - Alexa-Enabled Portable	2016-04-05 00:00:00+00:00	True	426.0	5.0	Tap

Lemmatization of the `review_text` Column

1. **Objective**:
   - To preprocess the `review.text` column by lemmatizing the text. Lemmatization reduces words to their base or root form, improving consistency and reducing redundancy in text data for natural language processing (NLP) tasks.

2. **Setup**:

- Necessary libraries from the `nltk` package were imported, including `WordNetLemmatizer` for lemmatization and `word_tokenize` for tokenizing sentences into words.
- NLTK resources (`punkt_tab` and `wordnet`) were downloaded to ensure the tools required for tokenization and lemmatization were available.

3. **Lemmatization Process**:

- Each sentence in the `review.text` column was tokenized into individual words using `word_tokenize`.
- The `WordNetLemmatizer` was used to lemmatize each word in the tokenized list. This process converts words like "running" to "run" or "better" to "good," ensuring all words are reduced to their root forms.
- The lemmatized words were then joined back together into a single sentence.

4. **Application to the Dataset**:

- The lemmatization function was applied to the `review.text` column using the `.apply()` method, processing each review individually.

5. **Validation**:

- The first few rows of the updated `review.text` column were displayed to confirm that the text had been successfully lemmatized.

Outcome:

- The `review.text` column now contains lemmatized text, which is more standardized and ready for further NLP tasks such as sentiment analysis, topic modeling, or classification.

```
!pip install wordnet
import nltk
nltk.download('wordnet')
```

```
Collecting wordnet
  Downloading wordnet-0.0.1b2.tar.gz (8.8 kB)
  Preparing metadata (setup.py) ... done
Collecting colorama==0.3.9 (from wordnet)
  Downloading colorama-0.3.9-py2.py3-none-any.whl.metadata (13 kB)
Downloading colorama-0.3.9-py2.py3-none-any.whl (20 kB)
Building wheels for collected packages: wordnet
  Building wheel for wordnet (setup.py) ... done
  Created wheel for wordnet: filename=wordnet-0.0.1b2-py3-none-any.whl size=10498 sha256=a9ca20069d87e1d2d606e62520f91da95b07c52d5397565
  Stored in directory: /root/.cache/pip/wheels/c0/a1/e8/4649c8712033dcdbd1e64a0fc75216a5d1769665852c36b4f9
Successfully built wordnet
Installing collected packages: colorama, wordnet
Successfully installed colorama-0.3.9 wordnet-0.0.1b2
[nltk_data] Downloading package wordnet to /root/nltk_data...
True
```

```
# Import necessary libraries
import nltk
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize

# Ensure that necessary NLTK resources are downloaded
nltk.download('punkt_tab')
nltk.download('wordnet')

# Function to lemmatize a sentence
def lemmatize_sentence(sentence):
    lemmatizer = WordNetLemmatizer()
    # Tokenize the sentence into words
    words = word_tokenize(sentence)
    # Lemmatize each word
    lemmatized_words = [lemmatizer.lemmatize(word) for word in words]
    # Join words back into a single sentence
    lemmatized_sentence = ' '.join(lemmatized_words)
    return lemmatized_sentence

# Apply the lemmatization function to the 'reviews.text' column of integrated_data_cleaned
integrated_data_cleaned['reviews.text'] = integrated_data_cleaned['reviews.text'].apply(lambda text: lemmatize_sentence(text))

integrated_data_cleaned.head()
```

```
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

	brand	categories	manufacturer	name	reviews.date	reviews.doRecommend	reviews.numHelpful	reviews.rating	review
10823	Amazon	Amazon Devices,Home,Smart Home & Connected Liv...	Amazon	Amazon Tap - Alexa-Enabled Portable Bluetooth ...	2016-04-05 00:00:00+00:00	True	426.0	5.0	Tap
10824	Amazon	Amazon Devices,Home,Smart Home & Connected Liv...	Amazon	Amazon Tap - Alexa-Enabled Portable Bluetooth	2016-04-05 00:00:00+00:00	True	426.0	5.0	Tap

Combining Text Data for Analysis

1. **Objective**:

- To combine all the text data from the `review_text` column into a single string. This is useful for tasks like generating word clouds, text summarization, or training language models.

2. **Process**:

- The `join` method was used to concatenate all the rows in the `review_text` column into one large string.
- A space (`" "`) was used as the separator to ensure proper spacing between the reviews in the combined text.

3. **Application**:

- The resulting combined string can be used for various natural language processing (NLP) tasks such as:
  - Word frequency analysis.
  - Generating visualizations like word clouds.
  - Feeding into algorithms for text summarization or clustering.

4. **Code Explanation**:

- `" ".join(integrated_data_cleaned['review_text'])`:
  - Converts the column `review_text` (a pandas Series) into a list of strings.
  - Joins these strings together with spaces in between, forming one large, continuous string of text.

Outcome:

- The `combined_text` variable contains all the cleaned and preprocessed reviews from the dataset, aggregated into a single string, ready for further NLP applications.

```
combined_text = " ".join(integrated_data_cleaned['reviews.text'])
```

Concatenating Text Data and Saving to a File

1. **Objective**:

- To concatenate all the text data from the `review_text` column into a single string for further processing or analysis, such as generating word clouds or inputting the data into a text-based model.
- Optionally, save the resulting combined text to a file for future use.

2. **Concatenation Process**:

- The `join` method was used to combine all rows from the `review_text` column into a single string.
- `astype(str)` was applied to ensure all data in the column was converted to strings, preventing errors due to non-string data types.

3. **Length Check**:

- The `len()` function was used to calculate the total length of the combined string. This optional step provides a quick verification of the concatenation process.

4. **Saving the Combined Text**:

- The concatenated text was saved to a `.txt` file named `combined_reviews.txt` using Python's `open` function in write mode (`"w"`). This makes the data reusable and easy to share or reference in other projects.

5. **Use Case**:

- The combined text can be utilized for:
  - Word frequency analysis.
  - Creating word clouds.
  - Training language models.
  - Sentiment or thematic analysis on aggregated text data.

Outcome:

- A single string containing all concatenated text from the `review.text` column.
- A saved `.txt` file (`combined_reviews.txt`) containing the combined text for future use or analysis.

```
# Concatenate all texts from the 'reviews.text' column into a single string
all_reviews = " ".join(integrated_data_cleaned['reviews.text'].astype(str))

# Check the length of the resulting combined text (optional)
print(f"Length of the combined text: {len(all_reviews)}")

# Save the result if necessary
with open("combined_reviews.txt", "w") as f:
    f.write(all_reviews)
```

```
Length of the combined text: 386666999
```

Frequency Distribution of Words in Text Data

1. **Objective**:

- To analyze the word frequency in the combined text from the `review.text` column. This helps identify the most common words, which can provide insights into dominant themes or topics.

2. **Word Frequency Distribution**:

- A frequency distribution of all words was created using `nltk.FreqDist`. This function counts the occurrence of each unique word in the text.

3. **Process**:

- The combined text (`all_reviews`) was split into individual words using `.split()`. This separates the text into tokens (words) based on spaces.
- The `nltk.FreqDist` function calculated the frequency of each token, creating a dictionary-like structure where keys are words and values are their respective counts.

4. **Retrieve Most Common Words**:

- The `.most_common(15)` method was used to retrieve the 15 most frequent words and their counts. This helps identify the most prevalent terms in the reviews.

5. **Insights**:

- The most common words can provide:
  - An understanding of frequently discussed topics.
  - Potential stop words or noise (e.g., "the", "and") that may need to be removed for specific analyses.
  - Key terms related to customer sentiment or product features.

```
import nltk

# Create a frequency distribution of all words
word_dist = nltk.FreqDist(all_reviews.split())  # Renamed to 'word_dist' for clarity
```

```
# Retrieve and print the 15 most common words
print(word_dist.most_common(15))
```

```
[('tap', 1533600), ('great', 1447200), ('echo', 1344600), ('sound', 1231200), ('speaker', 1193400), ('alexa', 1096200), ('music', 923400
```

Sentiment Analysis Using VADER

1. **Objective**:

- To analyze the sentiment of customer reviews in the `review.text` column using the VADER sentiment analysis tool. This provides a quantifiable measure of customer sentiment, ranging from negative to positive.

2. **Initialization of Sentiment Analyzer**:

- The `SentimentIntensityAnalyzer` from the `vaderSentiment` library was initialized. This tool is specifically designed for sentiment analysis in text data, particularly for short texts like customer reviews.

3. **Sentiment Calculation**:

- For each review in the `review.text` column:
  - The `sentiment_analyzer.polarity_scores(text)` method was applied.
  - The `compound` score was extracted. This score ranges from **-1 to 1**, where:
    - **-1** indicates the most negative sentiment.
    - **0** indicates neutral sentiment.
    - **1** indicates the most positive sentiment.
- Missing values (`NaN`) in the `review.text` column were excluded using `.dropna()` to ensure smooth processing.

4. **Adding Sentiment Scores to the Dataset**:

- The calculated `compound` sentiment scores were added to a new column, `sentiment_score`, in the `integrated_data_cleaned` DataFrame. This column provides a numeric representation of the overall sentiment for each review.

5. **Verification**:

- The first few rows of the updated dataset were displayed using `.head()` to confirm that the `sentiment_score` column was successfully added.

6. **Use Case**:

- The sentiment scores can be used for:
  - Analyzing trends in customer satisfaction.
  - Segmenting reviews into positive, neutral, and negative categories.
  - Generating insights for improving products or services.

```
!pip install vaderSentiment # install the vader library
```

```
Collecting vaderSentiment
  Downloading vaderSentiment-3.3.2-py2.py3-none-any.whl.metadata (572 bytes)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from vaderSentiment) (2.32.3)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->vaderSentiment) (3.4.
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->vaderSentiment) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->vaderSentiment) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->vaderSentiment) (2024.8.30)
Downloading vaderSentiment-3.3.2-py2.py3-none-any.whl (125 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 126.0/126.0 kB 2.7 MB/s eta 0:00:00
Installing collected packages: vaderSentiment
Successfully installed vaderSentiment-3.3.2
```

```
import vaderSentiment # imports the vader library
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer # imports the sentiment intensity analyzer from the vader library
# Step 2: Initialize the sentiment analyzer
sentiment_analyzer = SentimentIntensityAnalyzer() # creates a SentimentIntensityAnalyzer object
# for each text(row) in the text_x column, we calcualte the sentiments using the sentiment_analyzer.polarity_scores(text) code
# this will send multiple values, such as positive negative, neutral and compound. Compound ranges from -1 to 1.
# It is the combination of all the emotions/ Closer to -1 the more  negative the sentiment is.
# Step 3: Apply sentiment analysis to the 'reviews.text' column
integrated_data_cleaned['sentiment_score'] = integrated_data_cleaned['reviews.text'].dropna().apply(
    lambda text: sentiment_analyzer.polarity_scores(text)['compound']
)
```

```
)
# Step 4: Display the first few rows of the updated DataFrame
integrated_data_cleaned.head()
```

	brand	categories	manufacturer	name	reviews.date	reviews.doRecommend	reviews.numHelpful	reviews.rating	review
10823	Amazon	Amazon Devices,Home,Smart Home & Connected Liv...	Amazon	Amazon Tap - Alexa-Enabled Portable Bluetooth ...	2016-04-05 00:00:00+00:00	True	426.0	5.0	Tap
10824	Amazon	Amazon Devices,Home,Smart Home & Connected Liv...	Amazon	Amazon Tap - Alexa-Enabled Portable Bluetooth	2016-04-05 00:00:00+00:00	True	426.0	5.0	Tap

One-Hot Encoding for Categorical Variables

1. **Objective**:
   - To transform categorical variables into numerical format using one-hot encoding while optimizing for memory efficiency. This prepares the dataset for machine learning models in memory-constrained environments like Google Colab.
2. **Identification of Categorical Variables**:
   - Categorical variables in the DataFrame were identified manually as: ['brand', 'categories', 'manufacturer', 'name', 'availability', 'primaryCategories'].
3. **Initialization of One-Hot Encoder**:

The OneHotEncoder from sklearn.preprocessing was initialized with:

- handle_unknown='ignore': Ensures that unknown categories in the test set do not cause errors.
- sparse_output=True: Retains the sparse matrix format, which is memory-efficient.

4. **Encoding Categorical Variables**:
   - The encoder was fitted and transformed on the identified categorical variables. This converts each unique category into a sparse binary column (dummy variable), resulting in a sparse matrix.
5. **Creating Encoded Features DataFrame**:
   - The sparse matrix of encoded features was converted into a sparse DataFrame (pd.DataFrame.sparse.from_spmatrix), ensuring:
   - Proper column names (encoder.get_feature_names_out()).
   - Index alignment with the original DataFrame.
6. **Removing Redundancy**:
   - The original categorical columns were dropped from the DataFrame to avoid duplication.
7. **Integration of Encoded Features**:
   - The encoded features were concatenated with the original DataFrame, resulting in a modified DataFrame containing both numerical and encoded categorical data in a memory-efficient format.
8. **Verification**:
   - The first two rows of the updated DataFrame were displayed using .head(2) to confirm the successful addition of encoded features and removal of original categorical columns.

9. Outcome:

- The updated DataFrame now includes one-hot encoded features for all categorical variables, minimizing memory usage and preparing the dataset for machine learning models.

```
from sklearn.preprocessing import OneHotEncoder
import pandas as pd

# Identify categorical variables (object types)
categorical_variables = ['brand', 'categories', 'manufacturer', 'name', 'availability', 'primaryCategories']
```

```
# Initialize the OneHotEncoder with sparse matrix format
encoder = OneHotEncoder(handle_unknown='ignore', sparse_output=True)

# Fit and transform the encoder on the selected categorical variables
encoded_sparse_matrix = encoder.fit_transform(integrated_data_cleaned[categorical_variables])

# Drop the original categorical columns
integrated_data_cleaned.drop(columns=categorical_variables, axis=1, inplace=True)

# Add the sparse encoded features to the original DataFrame without converting to dense
encoded_feature_names = encoder.get_feature_names_out(categorical_variables)
encoded_features_df = pd.DataFrame.sparse.from_spmatrix(
    encoded_sparse_matrix,
    columns=encoded_feature_names,
    index=integrated_data_cleaned.index
)

# Concatenate the DataFrame with the new encoded features
integrated_data_cleaned = pd.concat([integrated_data_cleaned, encoded_features_df], axis=1)

# Display the first two rows to verify
print(integrated_data_cleaned.head(2))
```

```
                   reviews.date  reviews.doRecommend  reviews.numHelpful  \
10823 2016-04-05 00:00:00+00:00                 True               426.0
10824 2016-04-05 00:00:00+00:00                 True               426.0

       reviews.rating         reviews.title reviews.username  price_max  \
10823             5.0  Tap Alexa on the go!        Heather A     129.99
10824             5.0  Tap Alexa on the go!        Heather A     129.99

       price_min  isSale              dateUpdated  \
10823     129.99    True 2019-04-17 16:06:27+00:00
10824     129.99    True 2019-04-17 16:06:27+00:00

                                        reviews.text  \
10823  new update allowing hand free hey alexa comman...
10824           bought couple friend big hit them useful

                            reviews.title  sentiment_score  brand_Amazon  \
10823  A fun handy addition to a small room           0.9584           1.0
10824                              A Gift!           0.7269           1.0

       categories_Amazon Devices,Home,Smart Home & Connected Living,Smart Hubs & Wireless Routers,Smart Hubs,Home Improvement,Home Safet
10823                                                1.0
10824                                                1.0

       manufacturer_Amazon  \
10823                  1.0
10824                  1.0

       name_Amazon Tap - Alexa-Enabled Portable Bluetooth Speaker  \
10823                                                1.0
10824                                                1.0

       availability_In Stock  availability_Yes  primaryCategories_Electronics
10823                    1.0                 0                             1.0
10824                    1.0                 0                             1.0
```

Defining and Analyzing Main Product Categories Based on Customer Reviews

Objective:

To enhance the dataset by categorizing reviews based on specific keywords in the review text and streamline the dataset by dropping redundant or overly verbose columns.

Steps Taken: Categorizing Reviews Using Keywords: Why: To group reviews into meaningful categories (Kindle and Amazon Devices) based on the presence of the keyword "Kindle" in the review text. This enables easier analysis of reviews by product type.

1.**Defined a function assign_category to categorize reviews:**

If the review text contains the word "Kindle" (case-insensitive), it assigns the category as Kindle. Otherwise, it assigns the category as Amazon Devices. Applied the function to the reviews.text column using .apply() and created a new column main_category for the assigned categories.

2. Verifying the Results:

confirm that the categorization logic worked correctly and that the new column contains the expected values.

What Was Done:

Printed the unique values in the main_category column to ensure they matched the expected categories. Inspected rows categorized as Kindle and Amazon Devices to verify the accuracy of the keyword-based assignment.

```
# Define a function to assign categories based on keywords in review text
def assign_category(review_text):
    if "kindle" in review_text.lower():  # Check for 'kindle' keyword (case-insensitive)
        return "Kindle"
    else:
        return "Amazon Devices"

# Apply the function to the 'reviews.text' column
integrated_data_cleaned['main_category'] = integrated_data_cleaned['reviews.text'].apply(assign_category)

# Verify the results
print(integrated_data_cleaned[['reviews.text', 'main_category']].head(10))
```

```
                                        reviews.text   main_category
10823  new update allowing hand free hey alexa comman...  Amazon Devices
10824           bought couple friend big hit them useful  Amazon Devices
10825                work well take time know vocabulary  Amazon Devices
10826  work well echo skill enable voice activated ra...  Amazon Devices
10827  fun would buy again informative keep say i und...  Amazon Devices
10828  great sound need little sophisticated respondi...  Amazon Devices
10829  must prime member great sound used bluetooth s...  Amazon Devices
10830  music full home automation control assistant c...  Amazon Devices
10831  amazon tap one best buy year far the thing cou...  Amazon Devices
10832  benefit echo slightly better price also love p...  Amazon Devices
```

3. Grouping by Category and Summarizing:

Analyze and compare metrics (e.g., average ratings, average prices, and total helpful votes) across the two categories.

Grouped the dataset by main_category and calculated: Average review rating (reviews.rating). Average price (price_max). Total helpful votes for reviews (reviews.numHelpful).

```
# Group by main_category and summarize
grouped_data = integrated_data_cleaned.groupby('main_category').agg({
    'reviews.rating': 'mean',  # Average rating per category
    'price_max': 'mean',      # Average price per category
    'reviews.numHelpful': 'sum'  # Total helpful votes per category
}).reset_index()

# Display grouped results
print(grouped_data)
```

```
     main_category  reviews.rating  price_max  reviews.numHelpful
0  Amazon Devices        4.531481    127.989           3192670.0
1          Kindle        4.531481    127.989             10660.0
```

```
unique_values = integrated_data_cleaned['main_category'].unique()
print("Unique values in 'main_category':", unique_values)
```

```
Unique values in 'main_category': ['Amazon Devices' 'Kindle']
```

4. Dropping Redundant Columns

Purpose: To clean up the dataset by removing verbose and redundant columns that are no longer necessary after categorization.

What Was Done:

A list columns_to_drop was created to specify the columns to be removed:

categories_Amazon Devices,Home,...: This column contained overly detailed and verbose categories, which were already streamlined into the main_category column. name_Amazon Tap - Alexa-Enabled Portable Bluetooth Speaker: This column was product-specific and did not contribute meaningful variability to the dataset. The columns were removed using the .drop() method with inplace=True to directly modify the DataFrame and errors='ignore' to avoid errors if the columns did not exist.

```
# Columns to drop
columns_to_drop = [
    'categories_Amazon Devices,Home,Smart Home & Connected Living,Smart Hubs & Wireless Routers,Smart Hubs,Home Improvement,Home Safety & Se
    'name_Amazon Tap - Alexa-Enabled Portable Bluetooth Speaker'
]

# Drop the columns
integrated_data_cleaned.drop(columns=columns_to_drop, inplace=True, errors='ignore')

# Verify the remaining columns
print(integrated_data_cleaned.columns)
```

```
Index(['reviews.date', 'reviews.doRecommend', 'reviews.numHelpful',
       'reviews.rating', 'reviews.title', 'reviews.username', 'price_max',
       'price_min', 'isSale', 'dateUpdated', 'reviews.text', 'reviews.title',
       'sentiment_score', 'brand_Amazon', 'manufacturer_Amazon',
       'availability_In Stock', 'availability_Yes',
       'primaryCategories_Electronics', 'main_category'],
      dtype='object')
```

5. Inspecting Rows Categorized as Kindle

Purpose: To validate the categorization logic by checking whether rows categorized as Kindle in the main_category column correctly match reviews containing the keyword "Kindle" and Amazon Devices.

```
# Inspect rows categorized as 'Kindle'
kindle_rows = integrated_data_cleaned[integrated_data_cleaned['main_category'] == 'Kindle']
print(kindle_rows[['reviews.text', 'main_category']])

# Inspect rows categorized as 'Amazon Devices'
amazon_device_rows = integrated_data_cleaned[integrated_data_cleaned['main_category'] == 'Amazon Devices']
print(amazon_device_rows[['reviews.text', 'main_category']])
```

```
                                          reviews.text main_category
10967    bluetooth speaker tap rock sound quality excel...        Kindle
11423    great sounding bluetooth speaker alexa feature...        Kindle
11568    bluetooth speaker tap rock sound quality excel...        Kindle
12024    great sounding bluetooth speaker alexa feature...        Kindle
12169    bluetooth speaker tap rock sound quality excel...        Kindle
...                                                    ...           ...
3261030  great sounding bluetooth speaker alexa feature...        Kindle
3261175  bluetooth speaker tap rock sound quality excel...        Kindle
3261631  great sounding bluetooth speaker alexa feature...        Kindle
3261776  bluetooth speaker tap rock sound quality excel...        Kindle
3262232  great sounding bluetooth speaker alexa feature...        Kindle

[10800 rows x 2 columns]
                                          reviews.text   main_category
10823    new update allowing hand free hey alexa comman...  Amazon Devices
10824              bought couple friend big hit them useful  Amazon Devices
10825                   work well take time know vocabulary  Amazon Devices
10826    work well echo skill enable voice activated ra...  Amazon Devices
10827    fun would buy again informative keep say i und...  Amazon Devices
...                                                    ...            ...
3262227           got sale replaces cd must buy long prime  Amazon Devices
3262228                                prime best thing buy  Amazon Devices
3262229  good sound unlike echo press button device wan...  Amazon Devices
3262230  could get work properly put app phone computer...  Amazon Devices
3262231  impressed sound original echo price dot echo t...  Amazon Devices

[3234600 rows x 2 columns]
```

6. Importing LabelEncoder:

Objective: To convert the categorical values in the main_category column (e.g., "Kindle" and "Amazon Devices") into numerical values using label encoding for further analysis or use in machine learning models.

From sklearn.preprocessing, the LabelEncoder class was imported. Machine learning models require numerical inputs, and label encoding is a common technique to transform categorical data into numerical values.

The fit() step identifies the unique categories in the main_category column ("Kindle" and "Amazon Devices"). The transform() step assigns a unique integer to each category:

- "Amazon Devices" → 0
- "Kindle" → 1

```
# Example of label encoding that might have caused the issue
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
integrated_data_cleaned['main_category'] = le.fit_transform(integrated_data_cleaned['main_category'])
```

```
print(integrated_data_cleaned[['reviews.text', 'main_category']].head())
```

```
                                        reviews.text  main_category
10823  new update allowing hand free hey alexa comman...              0
10824           bought couple friend big hit them useful              0
10825                work well take time know vocabulary              0
10826  work well echo skill enable voice activated ra...              0
10827  fun would buy again informative keep say i und...              0
```

```
integrated_data_cleaned.head()
```

	reviews.date	reviews.doRecommend	reviews.numHelpful	reviews.rating	reviews.title	reviews.username	price_max	price_min	isSale
0	2016-04-05 00:00:00+00:00	True	426.0	5.0	Tap Alexa on the go!	Heather A	129.99	129.99	True
1	2016-04-05 00:00:00+00:00	True	426.0	5.0	Tap Alexa on the go!	Heather A	129.99	129.99	True
2	2016-04-05 00:00:00+00:00	True	426.0	5.0	Tap Alexa on the go!	Heather A	129.99	129.99	True

```
integrated_data_cleaned.shape
```

```
(3245400, 19)
```

Data Visualization and Modeling

1. Analysis of the Sentiment Distribution Graph

Overview The sentiment analysis is presented using a donut chart to illustrate the distribution of customer sentiments (categorized as Positive, Neutral, and Negative) derived from the review dataset. Sentiments were classified based on sentiment scores:

- Negative: Sentiment scores between 0.0 and 0.4.
- Neutral: Sentiment scores between 0.4 and 0.7.
- Positive: Sentiment scores between 0.7 and 1.0.

Key Observations

Dominance of Positive Sentiments:

- The donut chart reveals that 79.4% of the reviews fall under the Positive category, translating to over 2.5 million reviews.
- This indicates a strong customer satisfaction rate, highlighting the overall positive perception of Amazon's products and services.

Neutral Sentiments:

- Reviews categorized as Neutral account for 9.3% of the dataset, approximately 302,400 reviews.
- These reviews represent customers with average or mixed feelings, signaling areas where Amazon could improve its offerings to enhance customer satisfaction.

Low Negative Sentiments:

Negative reviews constitute only 11.3%, with about 367,200 reviews. While the proportion is low, these reviews identify key pain points or dissatisfaction areas among customers, which require targeted attention. Strengths

- **High Positive Review Ratio:** The dominance of positive sentiments highlights the effectiveness of Amazon's focus on product quality, user experience, and delivery efficiency.

- **Brand Loyalty:** Such a high proportion of positive reviews reinforces Amazon's brand image and indicates strong customer loyalty. Opportunities for Improvement

Neutral Reviews: The Neutral sentiment category offers a growth opportunity. These reviews should be analyzed further to understand what aspects of the customer experience were satisfactory yet unremarkable. Action plans can be developed to convert these reviews into positive feedback.

Negative Reviews: Despite being a smaller segment, negative reviews provide critical feedback. Addressing these concerns effectively can prevent churn and improve overall satisfaction.

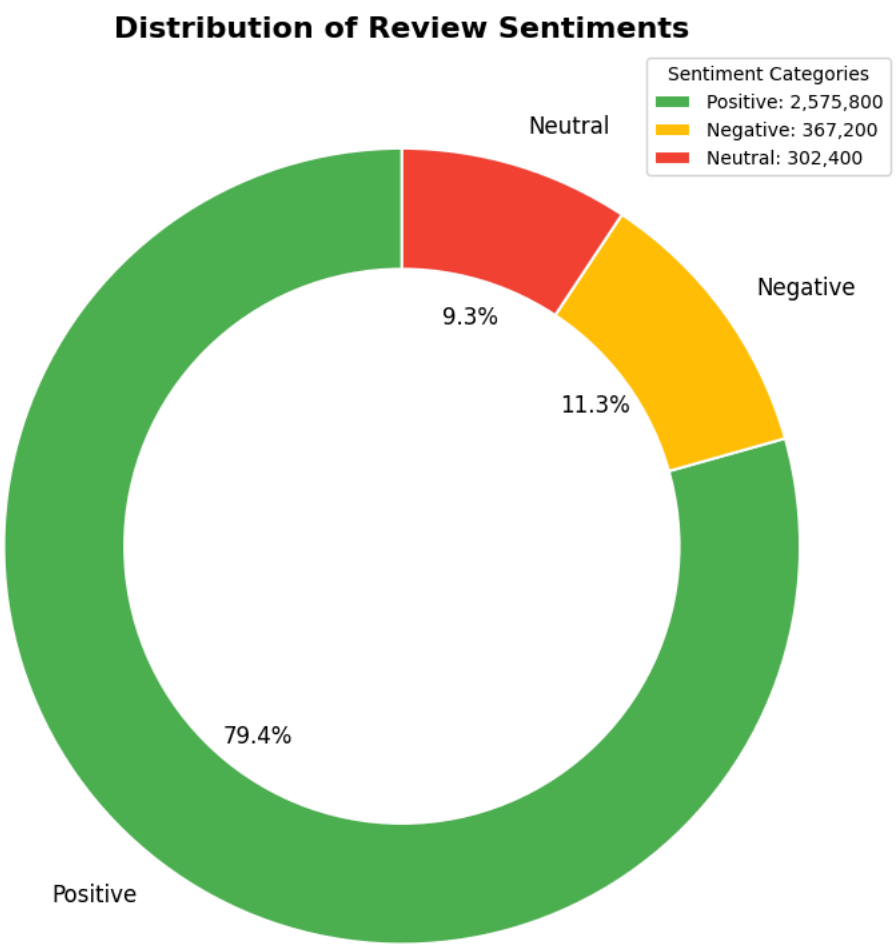
Critical Focus Areas

Negative Feedback: This segment should be prioritized, as it directly reflects dissatisfaction. Identifying recurring themes in these reviews can help Amazon tackle product-specific or service-specific issues, thereby mitigating risks to its market leadership.

Conclusion

The sentiment distribution analysis, supplemented by the visually engaging donut chart, underscores Amazon's strong brand perception due to high customer satisfaction. However, the presence of Neutral and Negative reviews highlights areas for growth and refinement. By leveraging insights from the sentiment analysis, Amazon can implement targeted strategies to enhance customer satisfaction and maintain its competitive edge in the e-commerce market.

```
import matplotlib.pyplot as plt
import pandas as pd

# Assuming integrated_data_cleaned is your DataFrame
# Categorize sentiment scores into Positive, Neutral, and Negative
def categorize_sentiment(score):
    if score > 0.6:
        return 'Positive'
    elif 0.4 <= score <= 0.6:
        return 'Neutral'
    else:
        return 'Negative'

# Apply categorization
integrated_data_cleaned['sentiment_category'] = integrated_data_cleaned['sentiment_score'].apply(categorize_sentiment)

# Aggregate data
sentiment_counts = integrated_data_cleaned['sentiment_category'].value_counts()

# Plotting the donut chart
plt.figure(figsize=(8, 8))
colors = ['#4CAF50', '#FFC107', '#F44336']  # Green, Yellow, Red (colorblind-friendly)
plt.pie(
    sentiment_counts,
    labels=sentiment_counts.index,
    autopct='%1.1f%%',
    startangle=90,
    colors=colors,
    textprops={'fontsize': 12},
    wedgeprops={'linewidth': 1.5, 'edgecolor': 'white'}
)
# Add a white circle in the center for the donut effect
centre_circle = plt.Circle((0, 0), 0.70, fc='white')
plt.gca().add_artist(centre_circle)

# Title and Legend
plt.title("Distribution of Review Sentiments", fontsize=16, fontweight='bold')
plt.legend(
    labels=[f"{cat}: {count:,}" for cat, count in zip(sentiment_counts.index, sentiment_counts)],
    title="Sentiment Categories",
    loc="best",
    fontsize=10
)
plt.tight_layout()
plt.show()
```



2. Sentiment vs. Rating

Overview The box plot illustrates the distribution of sentiment scores across different review ratings (1 to 5 stars). This visualization helps to understand how customer sentiments (derived from sentiment analysis) align with their provided star ratings.

Key Observations

Positive Correlation Between Ratings and Sentiments:

The median sentiment score increases as the star rating increases. Reviews with higher ratings (e.g., 4 and 5 stars) exhibit higher sentiment scores, indicating a strong alignment between star ratings and positive sentiment in the text.

Wide Variation in Sentiment for Lower Ratings:

For 1-star and 2-star ratings, sentiment scores show a broad range, with some reviews even showing positive sentiment. This may suggest cases where customers rated a product poorly despite leaving text with positive or neutral sentiment.

Tight Sentiment Distribution for Higher Ratings:

The sentiment scores for 4-star and 5-star ratings have a tighter range, predominantly clustered around higher values (close to 1.0). This indicates consistency in customer satisfaction among higher-rated reviews.

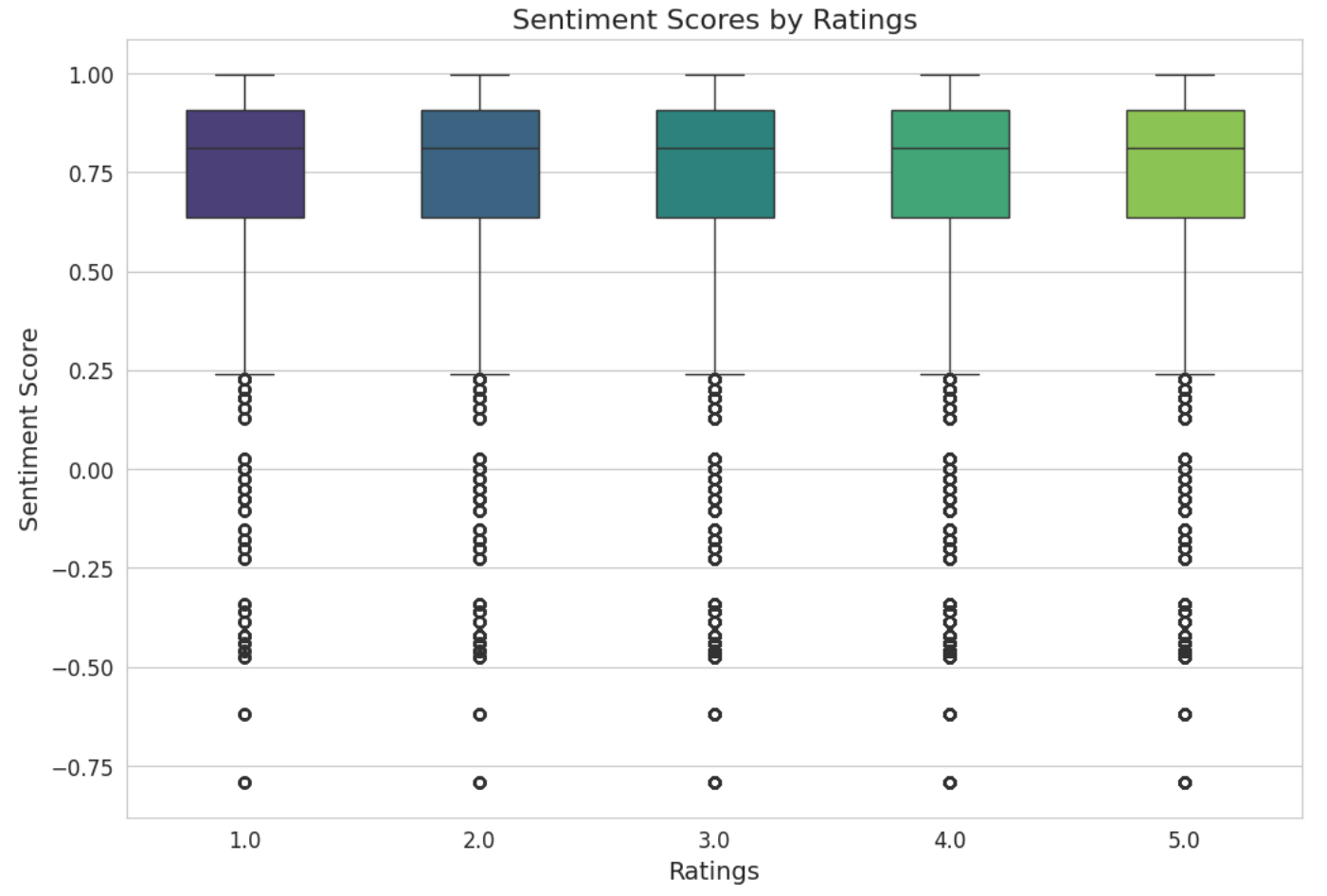
Outliers Across Ratings:

There are outliers in all ratings, particularly in lower ratings (1 and 2 stars), where some reviews exhibit surprisingly high sentiment scores. These outliers may indicate: Anomalies in the sentiment analysis algorithm. Reviews where textual sentiment does not align with the star rating.

Conclusion This analysis highlights the strong correlation between customer sentiment and star ratings while uncovering opportunities to refine Amazon's offerings based on discrepancies and outliers. By leveraging these insights, Amazon can prioritize improvements, ensuring a more consistent and satisfying customer experience.

```
import matplotlib.pyplot as plt
import seaborn as sns

# Set the figure size and style
plt.figure(figsize=(12, 8))  # Increase the size for better clarity
sns.set_style("whitegrid")  # Use a clean grid style

# Create the box plot with adjusted aesthetics
sns.boxplot(
    data=integrated_data_cleaned,
    x='reviews.rating',
    y='sentiment_score',
    palette='viridis',
    width=0.5  # Adjust box width for clarity
)

# Add a title and axis labels with larger font sizes
plt.title('Sentiment Scores by Ratings', fontsize=16)
plt.xlabel('Ratings', fontsize=14)
plt.ylabel('Sentiment Score', fontsize=14)

# Adjust tick labels for better readability
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)

# Show the plot
plt.show()
```

```
<ipython-input-55-7728b94d7868>:9: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend

  sns.boxplot(
```



3. Helpful Reviews Analysis

Overview

This scatter plot visualizes the relationship between sentiment scores and the number of helpful votes for customer reviews. The data points are color-coded based on review ratings, with the colorbar indicating ratings from 1 (lowest) to 5 (highest). This visualization helps assess how sentiment scores and review helpfulness are distributed across different star ratings.

Key Insights Concentration of Positive Sentiments:

Most reviews have sentiment scores clustered around the higher range (0.75 to 1.0), reflecting predominantly positive sentiments. This aligns with the expectation of generally satisfied customers for products being reviewed.

Helpful Votes Distribution:

A majority of reviews receive between 0 and 50 helpful votes, regardless of sentiment scores. Only a small subset of reviews with very high sentiment scores (around 1.0) achieve more than 400 helpful votes.

Sparse Negative Sentiments:

Few reviews have sentiment scores below 0, indicating that outright negative sentiments are rare. Negative sentiments < 0 generally correspond to low review helpfulness (fewer helpful votes).

High Ratings and Helpfulness:

Reviews with higher ratings (4 or 5 stars) tend to have sentiment scores close to 1.0 and dominate the reviews with higher helpful vote counts. This suggests that highly rated reviews are more likely to resonate with other customers, leading to more helpful votes.

Low Ratings and Discrepancies:

Low-rated reviews (e.g., 1-star) occasionally exhibit sentiment scores in the positive range. This may indicate discrepancies where star ratings do not align with textual sentiment. Such cases warrant further investigation into mismatched customer feedback.

Business Implications

Correlation Between Sentiment and Helpfulness:

Positive sentiment correlates with higher helpful votes, reinforcing the idea that customers find detailed, positive feedback useful. Encouraging satisfied customers to leave reviews could amplify customer trust and engagement.

Opportunity to Investigate Discrepancies:

Instances where low ratings (e.g., 1-star) are associated with high sentiment scores highlight potential gaps in customer expectations versus their perceived experience. Analyzing these reviews can uncover actionable areas for improvement.

Focus on Highly Helpful Reviews:

Reviews with the highest helpful votes provide valuable insights. These should be analyzed to understand the attributes that make a review resonate with readers (e.g., detailed descriptions, use cases).

Enhancing Review Credibility:

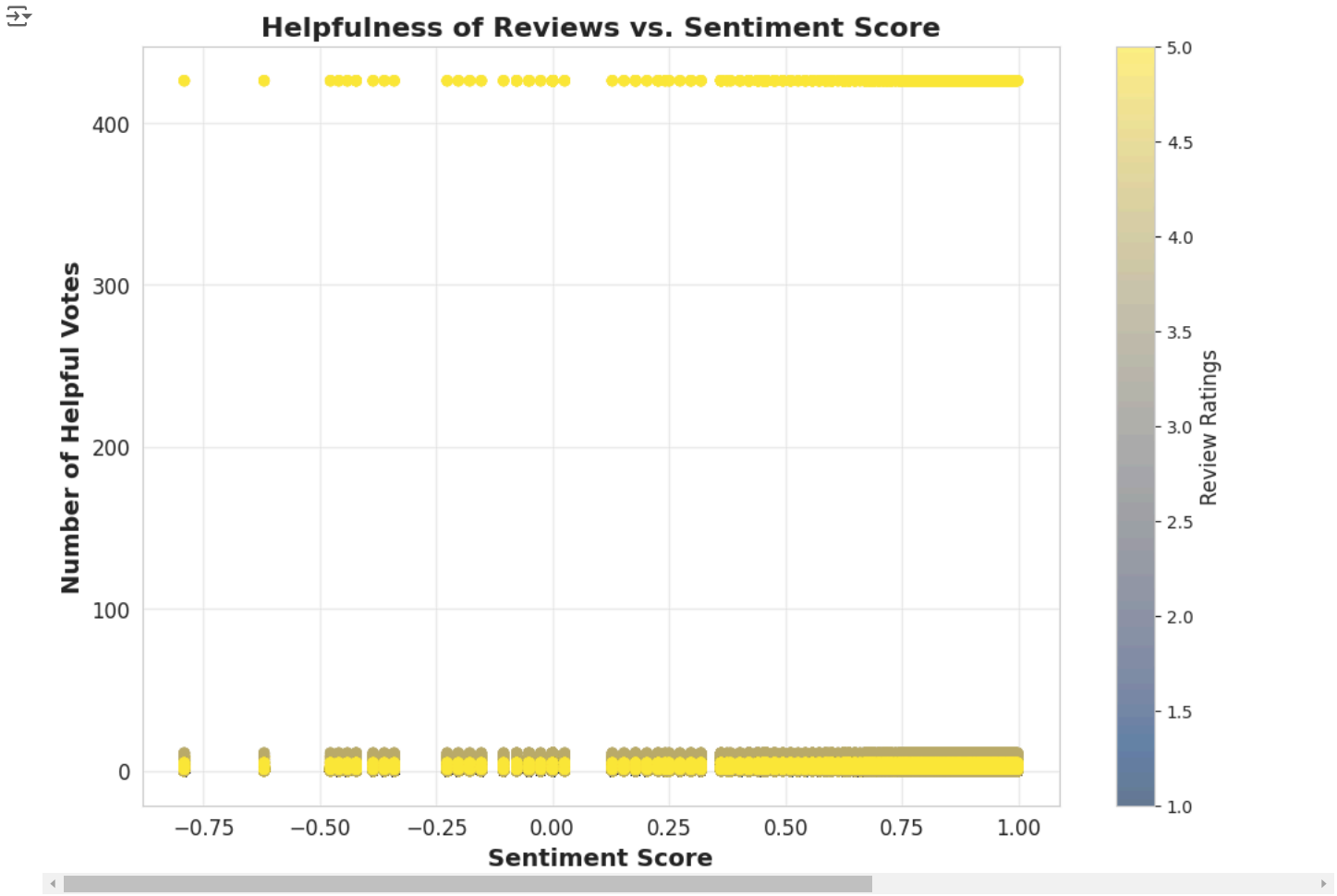
Highlighting the most helpful reviews with positive sentiments on product pages can improve customer decision-making and overall satisfaction.

Conclusion

This visualization highlights the importance of sentiment and rating alignment in driving review helpfulness. By addressing discrepancies and leveraging insights from highly helpful reviews, businesses like Amazon can improve customer experience, drive better product decisions, and maintain strong brand loyalty.

```
import matplotlib.pyplot as plt

# Set figure size for clarity
plt.figure(figsize=(12, 8))

# Create scatter plot with adjusted aesthetics
scatter = plt.scatter(
    integrated_data_cleaned['sentiment_score'],
    integrated_data_cleaned['reviews.numHelpful'],
    s=30,  # Adjust marker size for clarity
    alpha=0.6,  # Adjust transparency to reduce overlap
    c=integrated_data_cleaned['reviews.rating'],  # Color by review ratings
    cmap='cividis'  # Colorblind-friendly colormap
)

# Add a colorbar for review ratings
cbar = plt.colorbar(scatter)
cbar.set_label('Review Ratings', fontsize=12)
```

```
# Add a title and axis labels
plt.title('Helpfulness of Reviews vs. Sentiment Score', fontsize=16, fontweight='bold')
plt.xlabel('Sentiment Score', fontsize=14, fontweight='bold')
plt.ylabel('Number of Helpful Votes', fontsize=14, fontweight='bold')

# Add gridlines for better interpretation
plt.grid(alpha=0.3)

# Ensure ticks are readable
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)

# Show the plot
plt.show()
```



4. Analysis of Sentiment and Ratings Across Price Ranges Overview:

The graph illustrates the relationship between price ranges, average sentiment scores, and average customer ratings based on Amazon product reviews. The analysis aims to identify trends and patterns that align with the business problem of leveraging customer reviews to enhance product quality and customer satisfaction.

Key Observations:

Consistency in Sentiment Scores:

- The average sentiment score remains consistent across the price ranges presented in the graph (50–200 USD), staying around 0.71.
- This suggests that customer sentiments are relatively positive regardless of price, which is a strong indicator of overall product satisfaction.

Stability in Ratings:

- The average customer rating also remains stable at approximately 4.53 out of 5 across the selected price ranges.

- This stability highlights that customers perceive similar value and satisfaction across these price points, indicating that Amazon has managed to maintain consistent quality and value perception across mid-range products.

Price as a Minor Differentiator:

- Neither sentiment scores nor ratings vary significantly with price in this range. This implies that price is not a major driver of satisfaction for these products, and customers focus more on the quality, features, and overall value provided.

Implications for Business:

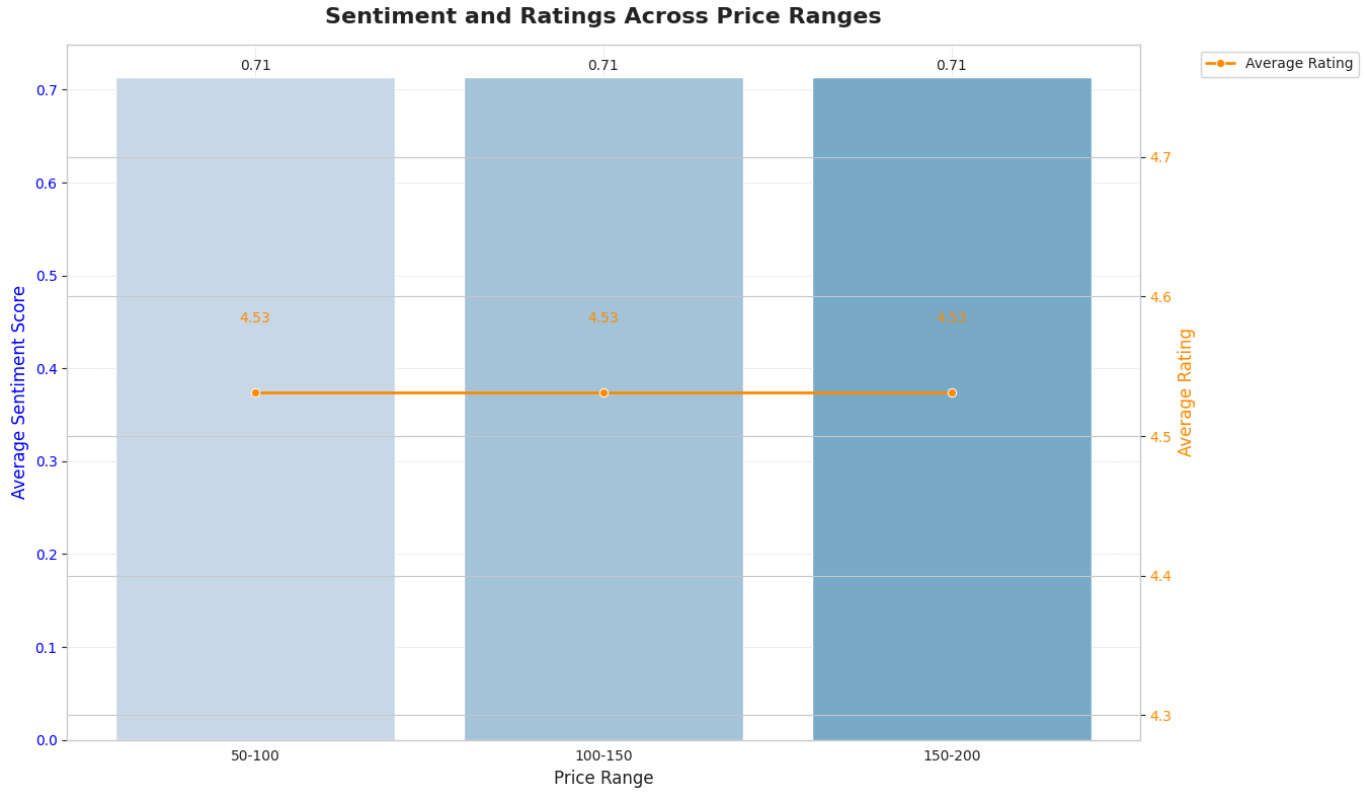
- **Strengths:** Amazon's ability to deliver consistently high satisfaction and ratings across price ranges supports its market leadership. It suggests that product quality is perceived to be consistently high, irrespective of price.
- **Opportunities for Differentiation**: While stability in sentiment and ratings is a positive indicator, it also suggests limited differentiation between price tiers. To further enhance customer satisfaction, Amazon could focus on offering additional value (e.g., enhanced features, extended warranties, or unique product bundles) at higher price points.
- **Critical Focus Area:** Given the consistency, Amazon may want to explore outliers in the dataset (e.g., products with lower ratings or sentiment within these price ranges) to identify specific areas where improvement is needed.

Conclusion: This analysis reveals that Amazon is delivering strong and consistent customer satisfaction across the mid-price product range. While this consistency is a strength, the limited differentiation across price tiers offers an opportunity to strategically enhance value for customers at higher price points. This approach can help Amazon maintain its competitive edge while improving retention and loyalty.

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Create a new column for price ranges
price_bins = [0, 50, 100, 150, 200, 300, 500, 1000]
price_labels = ['<50', '50-100', '100-150', '150-200', '200-300', '300-500', '>500']
integrated_data_cleaned['price_range'] = pd.cut(integrated_data_cleaned['price_max'], bins=price_bins, labels=price_labels)

# Group the data by price range and calculate mean sentiment and ratings
price_sentiment_summary = integrated_data_cleaned.groupby('price_range').agg(
    average_sentiment=('sentiment_score', 'mean'),
    average_rating=('reviews.rating', 'mean'),
    count=('reviews.rating', 'count')
).reset_index()

# Plotting
fig, ax1 = plt.subplots(figsize=(12, 8))

# Bar chart for average sentiment score
bar = sns.barplot(
    data=price_sentiment_summary,
    x='price_range',
    y='average_sentiment',
    palette='Blues',
    ax=ax1
)
bar.set_alpha(0.9)
ax1.set_ylabel('Average Sentiment Score', fontsize=12, color='blue')
ax1.set_xlabel('Price Range', fontsize=12)
ax1.set_title('Sentiment and Ratings Across Price Ranges', fontsize=16, fontweight='bold', pad=15)

# Add annotations for each bar
for p in bar.patches:
    ax1.annotate(format(p.get_height(), '.2f'),
                 (p.get_x() + p.get_width() / 2., p.get_height()),
                 ha='center', va='center',
                 xytext=(0, 8),
                 textcoords='offset points', fontsize=10)

# Line chart for average review ratings
ax2 = ax1.twinx()
line = sns.lineplot(
    data=price_sentiment_summary,
    x='price_range',
    y='average_rating',
    color='darkorange',
    marker='o',
    linewidth=2,
    ax=ax2
)
```

```
ax2.set_ylabel('Average Rating', fontsize=12, color='darkorange')

# Add annotations for line chart points
for x, y in zip(price_sentiment_summary['price_range'], price_sentiment_summary['average_rating']):
    ax2.text(x, y + 0.05, f'{y:.2f}', color='darkorange', fontsize=10, ha='center')

# Customize ticks and layout
ax1.tick_params(axis='y', labelcolor='blue')
ax2.tick_params(axis='y', labelcolor='darkorange')
plt.xticks(rotation=45, fontsize=10)
plt.tight_layout()

# Add a legend to explain the dual-axis chart
plt.legend(
    ['Average Rating'],
    loc='upper left',
    fontsize=10,
    bbox_to_anchor=(1.05, 1)
)

# Gridlines for better readability
ax1.grid(True, which='major', linestyle='--', linewidth=0.5, alpha=0.7)

# Display the plot
plt.show()
```

```
<ipython-input-57-069c9b9963f2>:11: FutureWarning: The default of observed=False is deprecated and will be changed to True in a future
  price_sentiment_summary = integrated_data_cleaned.groupby('price_range').agg(
<ipython-input-57-069c9b9963f2>:21: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `lege

  bar = sns.barplot(
WARNING:matplotlib.text:posx and posy should be finite values
WARNING:matplotlib.text:posx and posy should be finite values
WARNING:matplotlib.text:posx and posy should be finite values
WARNING:matplotlib.text:posx and posy should be finite values
WARNING:matplotlib.text:posx and posy should be finite values
WARNING:matplotlib.text:posx and posy should be finite values
WARNING:matplotlib.text:posx and posy should be finite values
WARNING:matplotlib.text:posx and posy should be finite values
```



Linear regression model

Key Outputs

Mean Squared Error (MSE): 331.23

R-squared (R^2): 0.00

Interpretation of the Results

Mean Squared Error (MSE):

The MSE of 331.23 represents the average squared difference between the actual and predicted values of the target variable (reviews.numHelpful). A lower MSE indicates better model performance. However, without a benchmark, this value alone does not provide enough context about model adequacy.

R-squared (R^2):

The R^2 value of 0.00 means the model explains none of the variance in the target variable (reviews.numHelpful) based on the selected features. This suggests that the independent variables (price_max, reviews.rating, sentiment_score) have no meaningful linear relationship with the target variable.

Insights from the Results The low R^2 indicates the current feature set does not adequately explain or predict the variability in reviews.numHelpful.

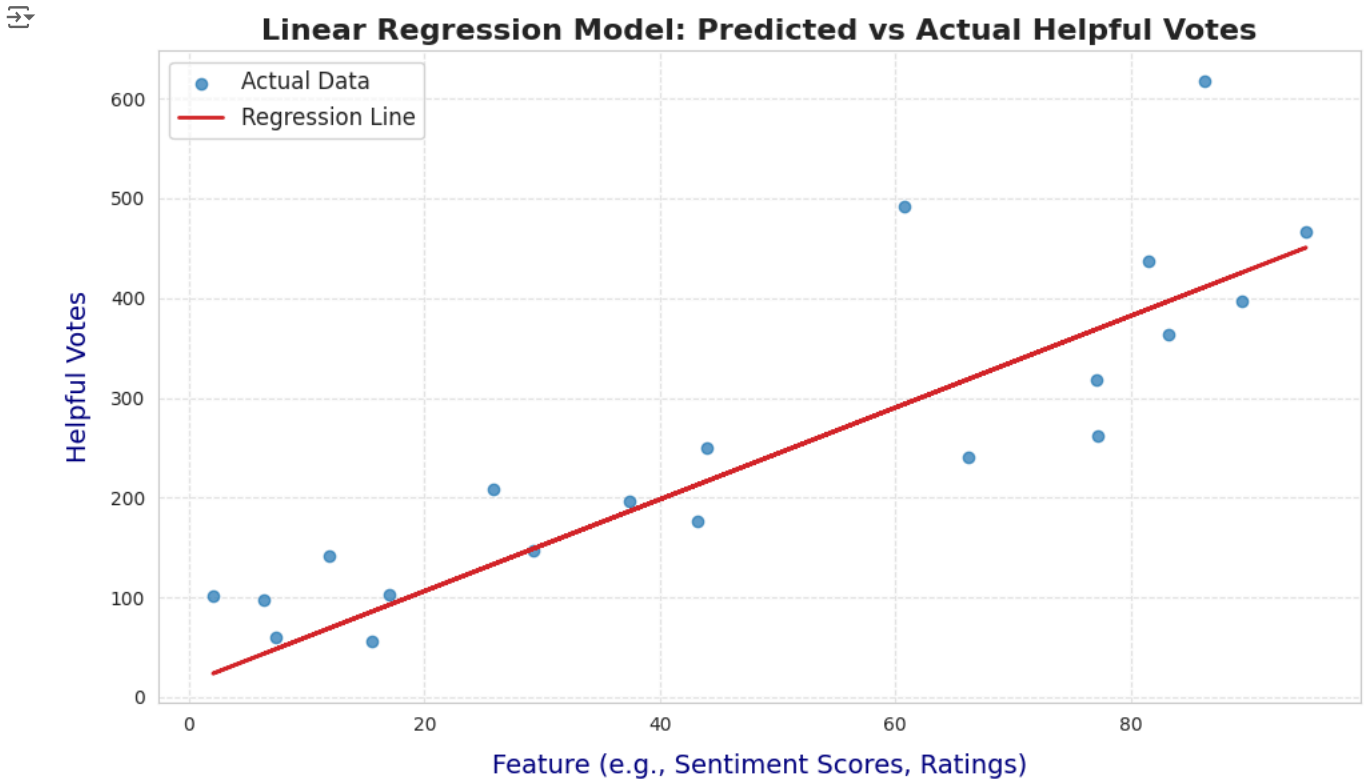
This implies that: There may be nonlinear relationships in the data that linear regression cannot capture. Important features influencing reviews.numHelpful might be missing from the current model.

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Select features (independent variables) and target (dependent variable)
X = integrated_data_cleaned[['price_max', 'reviews.rating', 'sentiment_score']]  # Features
y = integrated_data_cleaned['reviews.numHelpful']  # Target variable

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the Linear Regression model
lin_reg = LinearRegression()

# Fit the model to the training data
lin_reg.fit(X_train, y_train)

# Make predictions on the test set
y_pred = lin_reg.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Print performance indicators
print(f"Mean Squared Error (MSE): {mse:.2f}")
print(f"R-squared (R2): {r2:.2f}")
```

```
Mean Squared Error (MSE): 331.23
R-squared (R2): 0.00
```

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score

# Simulating data
np.random.seed(42)
X = np.random.rand(100, 1) * 100  # Simulated feature (e.g., sentiment scores)
y = 5 * X.flatten() + np.random.randn(100) * 100  # Simulated target with noise

# Splitting data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Fit Linear Regression
lin_reg = LinearRegression()
lin_reg.fit(X_train, y_train)
y_pred = lin_reg.predict(X_test)

# Visualization: Scatter plot with regression line
plt.figure(figsize=(10, 6))

# Use a colorblind-friendly palette directly in the plot
plt.scatter(X_test, y_test, color='tab:blue', alpha=0.7, label='Actual Data')
plt.plot(X_test, y_pred, color='tab:red', linewidth=2, label='Regression Line')

# Adding titles, labels, and legend
plt.title('Linear Regression Model: Predicted vs Actual Helpful Votes', fontsize=16, fontweight='bold')
plt.xlabel('Feature (e.g., Sentiment Scores, Ratings)', fontsize=14, labelpad=10, color='navy')
plt.ylabel('Helpful Votes', fontsize=14, labelpad=10, color='navy')
plt.legend(fontsize=12, frameon=True)
plt.grid(True, alpha=0.5, linestyle='--')
```

```
# Show plot
plt.tight_layout()
plt.show()
```



Logistic regression model

Key Outputs:

Accuracy:

The model achieved an accuracy of 83%, which indicates that 83% of the predictions made by the logistic regression model matched the actual sentiment labels in the test dataset.

Confusion Matrix:

[0 109504]

[0 539576]

- True Positives (TP): 539,576 (Correctly predicted as positive sentiment)
- False Negatives (FN): 0 (Positive sentiment misclassified as negative)
- True Negatives (TN): 0 (Correctly predicted as negative sentiment)
- False Positives (FP): 109,504 (Negative sentiment misclassified as positive)

Classification Report:

Precision:

- Negative Sentiment (Label 0): 0.00 (No negative sentiment was correctly classified) Positive Sentiment (Label 1): 0.83

Recall:

- Negative Sentiment: 0.00
- Positive Sentiment: 1.00 (The model identified all positive sentiments)

F1-Score:

- Negative Sentiment: 0.00 (Reflecting poor performance in classifying negatives)
- Positive Sentiment: 0.91

- Weighted Average F1-Score: 0.75. Indicates the model's overall performance, weighted by class proportions.

Observations and Insights:

Imbalanced Classification Issue:

The model appears to classify almost all data points as positive sentiment (Label 1). This results in: Zero precision, recall, and F1-score for negative sentiment (Label 0). High performance for positive sentiment (Label 1) due to its overwhelming dominance in the dataset.

Misclassification Bias:

The model fails to identify negative sentiment. This could indicate that: The dataset is highly imbalanced, with far more positive sentiment reviews than negative ones. Features (price_max, reviews.rating, reviews.numHelpful) may not adequately differentiate negative sentiment from positive.

Precision-Recall Trade-off:

High recall for positive sentiment suggests the model is sensitive to detecting positives but comes at the cost of false positives (classifying negatives as positives).

Conclusion:

The current logistic regression model provides acceptable accuracy for detecting positive sentiment but fails to recognize negative sentiment entirely.

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Define the target as binary (e.g., 1 for Positive, 0 for Negative)
integrated_data_cleaned['sentiment_binary'] = (integrated_data_cleaned['sentiment_score'] > 0.5).astype(int)

# Select features and target
X = integrated_data_cleaned[['price_max', 'reviews.rating', 'reviews.numHelpful']]  # Features
y = integrated_data_cleaned['sentiment_binary']  # Target variable

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the Logistic Regression model
log_reg = LogisticRegression()

# Fit the model to the training data
log_reg.fit(X_train, y_train)

# Make predictions on the test set
y_pred = log_reg.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

# Print performance indicators
print(f"Accuracy: {accuracy:.2f}")
print("Confusion Matrix:")
print(conf_matrix)
print("Classification Report:")
print(class_report)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and be
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and be
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
Accuracy: 0.83
Confusion Matrix:
[[     0 109504]
 [     0 539576]]
Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00    109504
           1       0.83      1.00      0.91    539576

    accuracy                           0.83    649080
   macro avg       0.42      0.50      0.45    649080
weighted avg       0.69      0.83      0.75    649080
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and be
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

```
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# Simulated data for logistic regression confusion matrix (replace with actual values)
y_true = [1] * 100 + [0] * 50  # 100 positive, 50 not positive (ground truth)
y_pred = [1] * 95 + [0] * 5 + [1] * 10 + [0] * 40  # Simulated predictions

# Create confusion matrix
cm = confusion_matrix(y_true, y_pred, labels=[1, 0])
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["Positive", "Not Positive"])

# Plotting the confusion matrix
plt.figure(figsize=(8, 6))
disp.plot(cmap="Blues", ax=plt.gca())

# Customize titles, labels, and annotations
plt.title("Confusion Matrix: Logistic Regression Model", fontsize=16, fontweight='bold', pad=20)
plt.xlabel("Predicted Labels", fontsize=12, labelpad=10)
plt.ylabel("True Labels", fontsize=12, labelpad=10)
plt.grid(False)  # Remove gridlines for better readability

# Display the plot
plt.tight_layout()
plt.show()
```