MA336: Artificial intelligence and Machine Learning with applications_Final Project Report

APRIL 2,2024

University of Essex, UK
Authored by: Mantosh Nandy
Registration no: 2315740
Msc Data Science
Year 2023-24

Sentiment Analysis by NLP

NLP: bridges the Gap between Computers & Human languages.

NLP focuses on enabling computers to interpret text and spoken words like humans do, allowing for well-informed decisions to be made without the help of thoroughly reviewing every single comment. This technology also helps in analyzing the sentiment of the content.

Uses of NLP:

- Speech recognition
- Named entity recognition.
- Sentiment analysis (in this report we will understand this use case)
- Natural language generation (structured speech to text)
- Word sense disambiguation
- Text classification
- Chatbots & Virtual Assistant

> ***"Natural Language processing is the next frontier in Artificial intelligence"* (2017)**
>
> ---------- *Quoted by Fei-Fei Li, a Leading AI Researcher*

In this report, I have explained about the work on EDA & Sentiment Analysis of Amazon products review dataset with the help of Naïve Bayes & Support Vector classifier algorithm.

Introduction

Online reviews have become a must-have for consumers looking to make informed buying decisions (influencing an estimated **84% of consumers** before they make a purchase). And nowhere is this more apparent than in e-commerce platforms such as Amazon, Flipkart, Temu where customer reviews have the power to influence product perception and ultimately sales success. However, with millions of reviews generated every day, manually analyzing sentiment can be overwhelming or we can say it is impossible. That's where AI and, in particular, Sentiment Analysis techniques (SA) come in.

Background & Motivation

Automatically assessing customer sentiment based on reviews offers valuable insights to any business. Recognizing customer satisfaction levels, pinpointing improvement opportunities, and evaluating product reception are essential for enhancing product development, refining marketing strategies, and enhancing overall customer satisfaction which helps to grow to business for more profitable & sustainable.

Why AI?

Traditional approaches to sentiment analysis typically depend on keyword matching or lexicon-based techniques, which may find it struggling to handle the intricacies of human language. However, AI-driven sentiment analysis techniques employ advanced algorithms that have been trained on extensive collections of classified reviews. These sophisticated algorithms can detect intricate connections among words, phrases, and sentiment, leading to more accurate and insightful analyses.

Originality and Difficulty:

Although sentiment analysis of online reviews is a widely recognized area, there is still space for innovation. Here are some ways to make your project stand out:

- **Focus on a specific product category**: Analysis of reviews for a particular product type like electronics, clothing and consider incorporating domain-specific knowledge.

- **Extend beyond sentiment analysis:** Explore aspects like emotional outcomes (intensity of sentiment) or identify specific topics or concerns raised in reviews.
- **Multilingual analysis**: If we have the opportunity to utilize datasets in multiple languages, consider investigating sentiment analysis across various linguistic contexts.
- **Comparative analysis**: Contrast sentiment analysis outcomes for rival products or examine how sentiment evolves over time for a particular product.

Difficulty:

- The difficulty level of a project depends on the chosen approach for analysis purpose.
- For a good entry point we should start with pre-trained models and established libraries.
- For higher accuracy we should focus on building custom deep learning models or incorporating advanced NLP techniques.

The beauty of the power of Artificial intelligence and Sentiment analysis, trying to achieve the goal of this project which is to uncover valuable insights from Amazon's vast data of product reviews, allowing companies to make data-informed decisions for enhance the organization`s stability, helps to make data-driven decisions regarding product development, marketing strategies and improve customer experience.

Analysis of DataSet:

- **Data Source:**
  This dataset comprises real-world data collected from customers who are using e-commerce platform Amazon for purchasing their products spanning across the world. It encompasses all the significant customer reviews, customer ratings including customers recommendations. The dataset consists of samples from Amazon user`s Ratings for selected products. The reviews are picked casually, and the corpus has nearly 1.6k reviews of different Amazon customers.
- **Acknowledgement:**
  The dataset sourced from the website:
  https://www.kaggle.com/datasets/yasserh/amazon-product-reviews-dataset-dataset

This dataset consists of 1597 rows which is considered as data entry and has 27 columns. Here has 1597 non-null values for 10 columns (id, asins, brand, categories, dateAdded, dateUpdated, name, keys, reviews.sourceURLs, and reviews.text). In this dataset there are 1597 non-null values for 7 columns (asins, categories, names, prices, reviews.date, reviews.rating, and reviews.username) that are of '*object'* data type. However, it is not clear whether they are non-null or if they contain empty strings. It is observed that there are several columns (colors, dimension, reviews.userCity, reviews.userProvince, and sizes) that have significant numbers of null values, which could indicate missing or incomplete data.

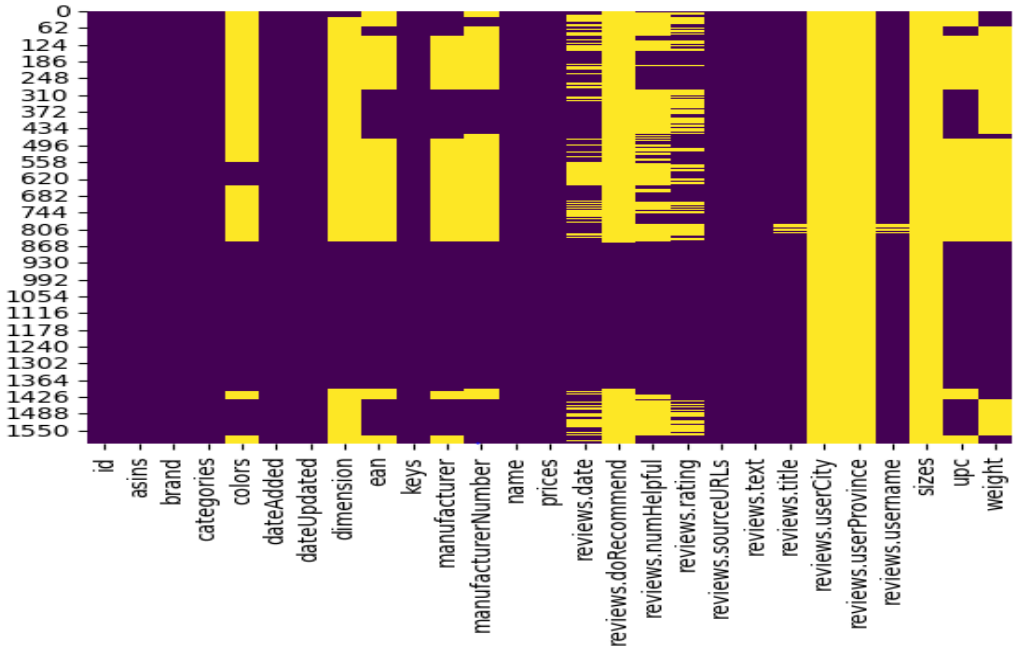
Before diving deeper, to improve the overall quality of the data for analysis, it's beneficial to explore the distribution and quality of each variable. We might need to perform some cleaning tasks**.** Additionally, we might need to address inconsistencies, outliers, missing values, and data format variations through cleaning and preprocessing steps.

PRELIMINARY ANALYSIS –

At first, I have done some data structure`s analysis of this dataset using ‘describe ()’ which helps us get a quick overview of the dataset’s distribution, central tendency, and variability. Also helps in understanding the data's structure, identifying potential errors like missing values or outliers, and gaining insights into the dataset's characteristics before proceeding with further analysis or preprocessing.

HeatMap Analysis:

Heatmaps are a powerful visualization tool for data exploration. They use color intensity to visually represent values, allowing you to quickly identify areas of high or low activity, correlations between variables, and overall data distribution.
Here I am applying heatmap analysis to uncover insights, patterns, and relationships that may not be apparent from raw data alone.

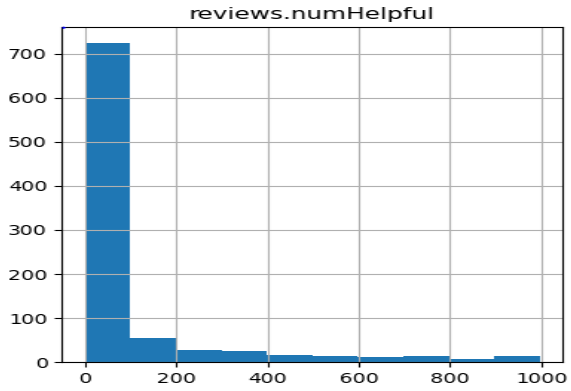


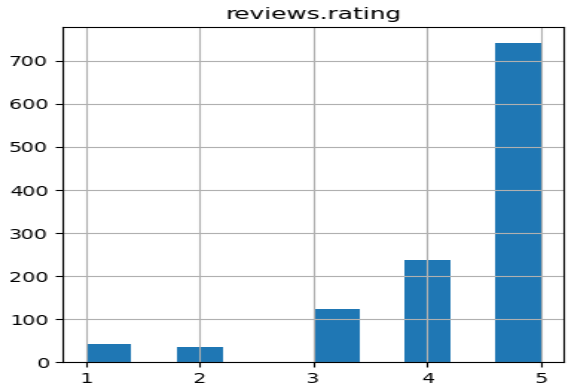
From this heatmap analysis we can decide that should be remove reviews.userCity, reviews.userProvince, sizes.

Data Cleansing:

For the data cleaning process, we are applying ‘drop ()’ which helps us to remove the columns which hold null value. It is a powerful tool for data manipulation and cleaning in Pandas before starting data visualization.

As a preprocessing part – I am trying to do a small data visualization part for trying to understand the co relation in between reviews.numhelpful & reviews.ratings which helps us in future for sentimental analysis as it hold the customer`s sentimental revies.





The x-axis is labeled "reviews.numHelpful" and likely represents the number of times a review was found helpful by other users. The y-axis is labeled "reviews.rating" and represents the rating of the review on a scale of 1 to 5.

Based on the available data, there is a slight tendency for reviews with higher ratings to receive more helpful votes, but this trend is not consistent or strong. There are several instances where lower-rated reviews have more helpful votes than higher-rated ones, indicating that there are many exceptions to this trend.
A considerable portion of reviews appear in the bottom left corner, indicating that numerous reviews received low ratings and were not deemed helpful by users. Additionally, there are reviews located in the top right corner, suggesting that despite having a low number of "helpful" ratings, some reviews received high overall ratings.

Now, for robust model performance, prevent overfitting, and enable fair evaluation and comparison of different models or algorithms we divide the dataset into training & testing dataset which is essential for NLP analysis.

So, before splitting the dataset, I have done some pre processing on 'reviews.rating' column for remove the null values as well as convert the data into integer format. We should keep in mind that it might be lead to data loss if the missing values were a significant portion of the original data.

Split the Dataset into Test & Train:

For splitting the dataset, chosen 'Stratified Shuffle Method' because to prevent from the heavy bias in our dataset towards one particular section, we implement shuffling after each new data point is inserted. This ensures that no single type of data dominates the training dataset, resulting in a more balanced model that is well-equipped to handle diverse datasets.

```
# By using StratifiedShuffleSplit, start test & train the datasets
split=StratifiedShuffleSplit(n_splits=5,test_size=0.3, random_state=1)
for train_index,test_index in split.split(mn_data_after,mn_data_after["reviews.rating"]):
    strshf_train=mn_data_after.iloc[train_index]
    strshf_test=mn_data_after.iloc[test_index]
strshf_train.head()
```

The provided code is conducting a 5-fold stratified shuffle split on the dataset **mn_data_after**, utilizing the **reviews.rating** column as the target variable. This process ensures that the dataset is split into five subsets while preserving the distribution of ratings in each subset. Split the dataset on 70:30 portion of training & testing purpose respectively.

The resulting **strshf_train** and **strshf_test** dataframes are reindexed versions of the original dataset, containing the rows corresponding to the training and testing indices, respectively. The **strshf_train.head()** command displays the first 5 rows of the **strshf_train** DataFrame, providing a glimpse into the training dataset.

EDA Analysis:

Exploratory Data Analysis (EDA) is an essential phase in the data science workflow as it helps in comprehending the dataset under examination. This entails scrutinizing and graphically representing the data, recognizing patterns, outliers, and irregularities, and extracting insights to steer subsequent data analysis and modeling process.

```
reviews=strshf_train.copy()
reviews.head(2)
```

To prevent accidentally modifying the original data, the first line creates a copy of the strshf_train dataframe and stores it in a new variable named reviews. This ensures the original data remains untouched.
We can take a quick look at the data in reviews using the second line. It shows the first 2 rows with reviews.head(2), giving us a basic idea of what's there.

```
reviews.groupby('asins')['name'].unique()
```

This code segment offers a method to investigate the distinct product names linked with each ASIN in the reviews column. Essentially, it reveals the count of unique product names within each product category identified by the ASIN. From the output , I observed that the ASINs in the dataset correspond to one or more product names, and the mapping between these ASINs and their respective product names is provided in the list.

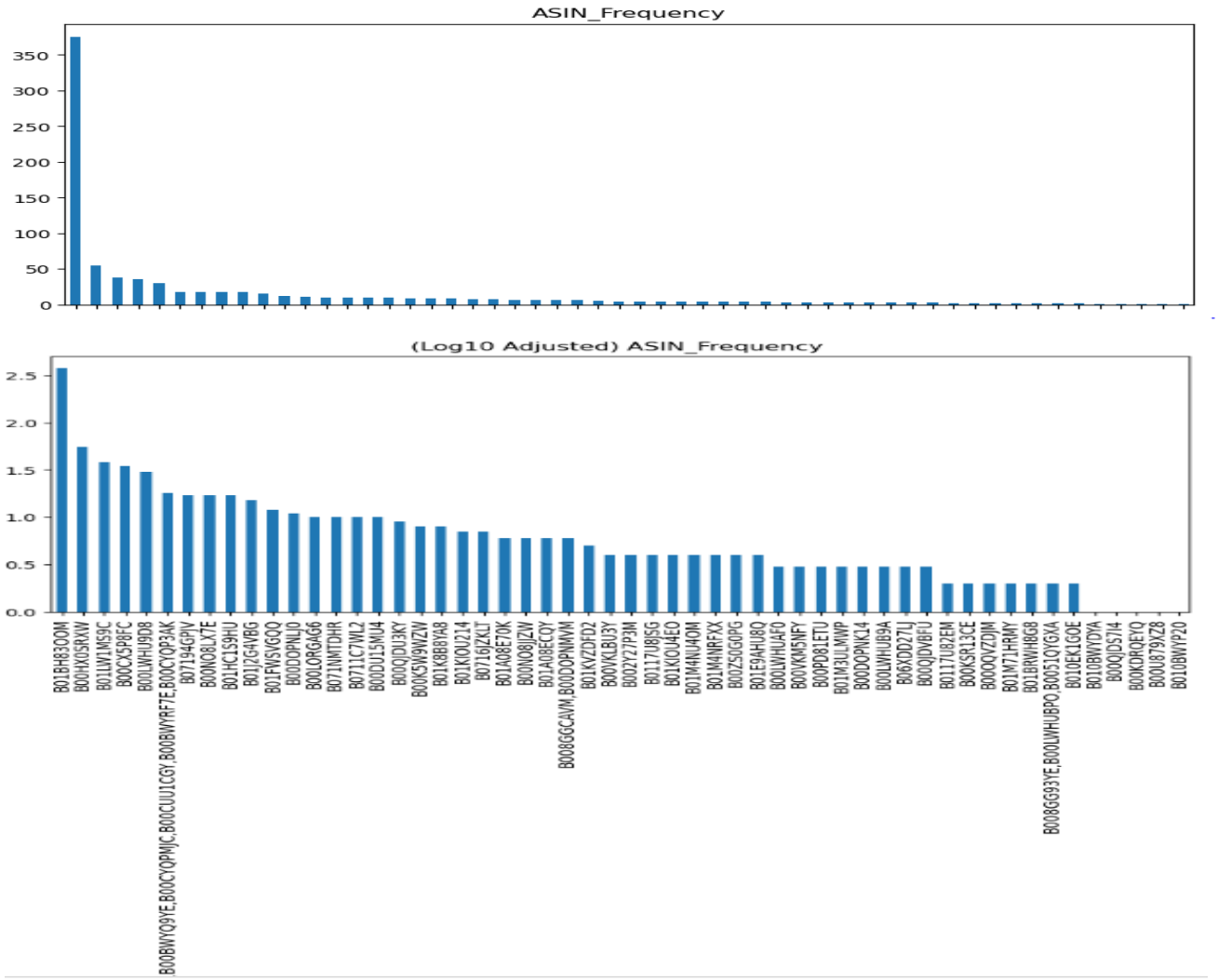
```
reviews.info()
```

This code segment refers how many columns & rows are present in this dataframe. Found 823 entries including 23 columns. From the output, *specially* found that 'reviews.rating' column is of type of int32 and contains numerical ratings provided by users. The 'reviews.numHelpful' column contains the number of helpful votes received for each review.

```
fig = plt.figure(figsize=(10,10))
# we use subplot when we have to see interrelationship btw two graphs
axis1 = plt.subplot(2,1,1)
axis2 = plt.subplot(2,1,2, sharex = axis1)
reviews["asins"].value_counts().plot(kind="bar", ax=axis1,
                                     title="ASIN_Frequency")

np.log10(reviews["asins"].value_counts()).plot(kind="bar", ax=axis2,
                                     title="(Log10 Adjusted) ASIN_Frequency ")
# np.log10 normalises our data to visualise the graph and difference much better
plt.show()
```

This code offers an informative visualization of the occurrence frequency of ASINs within the reviews dataframe. It effectively showcases the distribution of ASINs by employing a chart adjusted with log10 scaling. From this code found two side-by-side bar plots showing the frequency distribution of ASINs in the reviews DataFrame, with the second plot using a logarithmic scale for better visualization of the data distribution.

The Charts are as follows:



From above first chart observations like : a bar plot of the frequency counts of the "asins" values in the "reviews" dataset and displays it in the first subplot. The title of the plot is "ASIN Frequency".

From the second subplot barchart found of the logarithmic (base 10) frequency counts of the "asins" values in the "reviews" dataset and displays it in the second subplot. The title of the plot is "ASIN Frequency (Log10 Adjusted)" which makes it easier to see the distribution of higher frequency "asins" and helps to distinguish between the frequencies of less popular products. So, now we can understand that what asins have max number of people rating. And the answers of which asins have most sales.

```
# mean of rating from trianing data set
reviews['reviews.rating'].mean()
```

```
4.359659781287971
```

This line computes the mean value of the 'reviews.rating' column within the reviews DataFrame. It's probable that the 'reviews.rating' column consists of numeric data representing the star ratings (ranging from 1 to 5) assigned to individual reviews. Here we got the ratings output ~ 4.36.

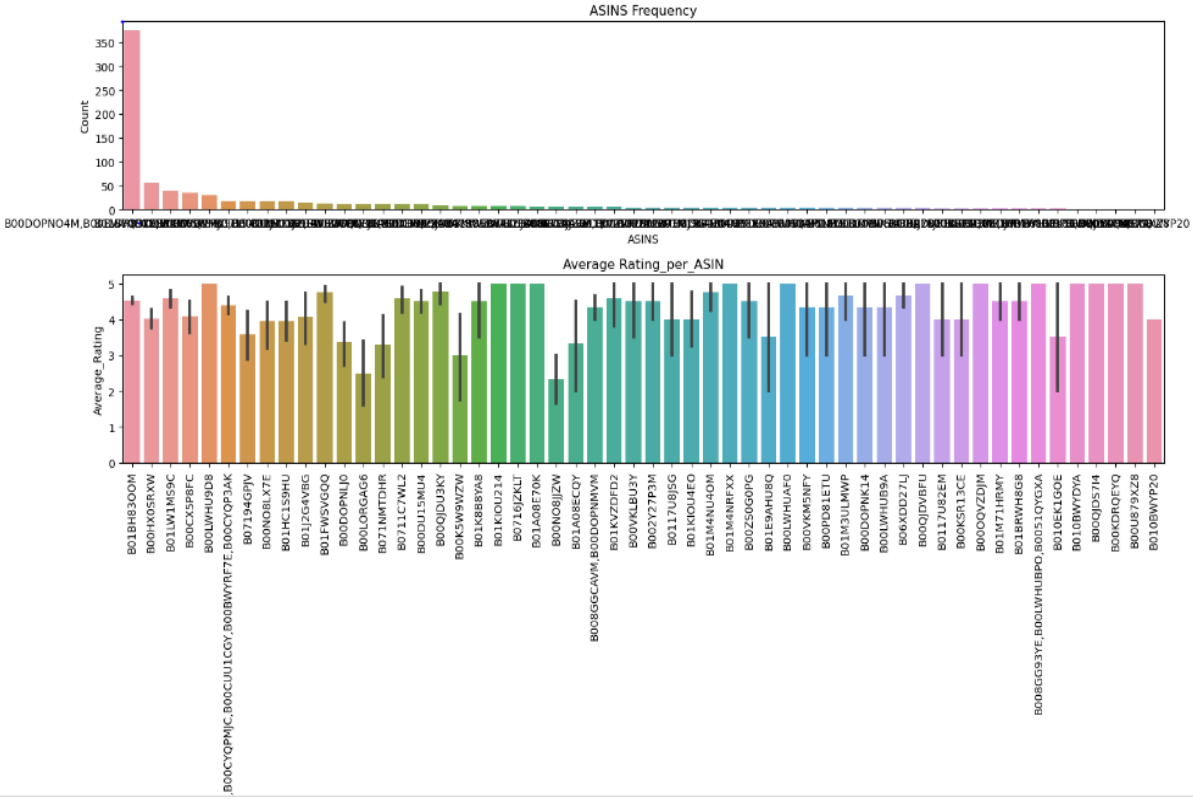
```
# Using barplot to display number of people giving ratings
count_asins=reviews['asins'].value_counts().index

# Settng up subplots
fig, axes = plt.subplots(2, 1, figsize=(16,12))

# Plotting ASINS frequency
sns.countplot(x='asins', data=reviews, order=count_asins, ax=axes[0])
axes[0].set_title('ASINS Frequency')
axes[0].set_xlabel('ASINS')
axes[0].set_ylabel('Count')

# Plotting average rating per ASIN
sns.barplot(x='asins', y='reviews.rating', data=reviews, order=count_asins, ax=axes[1])
axes[1].set_title('Average Rating_per_ASIN')
axes[1].set_xlabel('ASINS')
axes[1].set_ylabel('Average_Rating')
axes[1].tick_params(axis='x', rotation=90)

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

From this above code trying to understand how many people giving ratings. Here for better understanding use bar plots graphs. This code creates a subplot that contains two charts. The first chart is a bar plot that displays the frequency of ASINs. The second chart is also a bar plot that shows the connection between ASINs and average ratings. Overall, this subplot offers a detailed visualization of the distribution of ASINs and their corresponding ratings.

Specialty of this code for visualization, used the **tick_params()** function in the matplotlib library is used to modification the appearance of the ticks and tick labels in a plot. It can be used to change the color, size, and style of the ticks and tick labels, as well as the label rotation and font size.

Details of output on the next page.

Output:

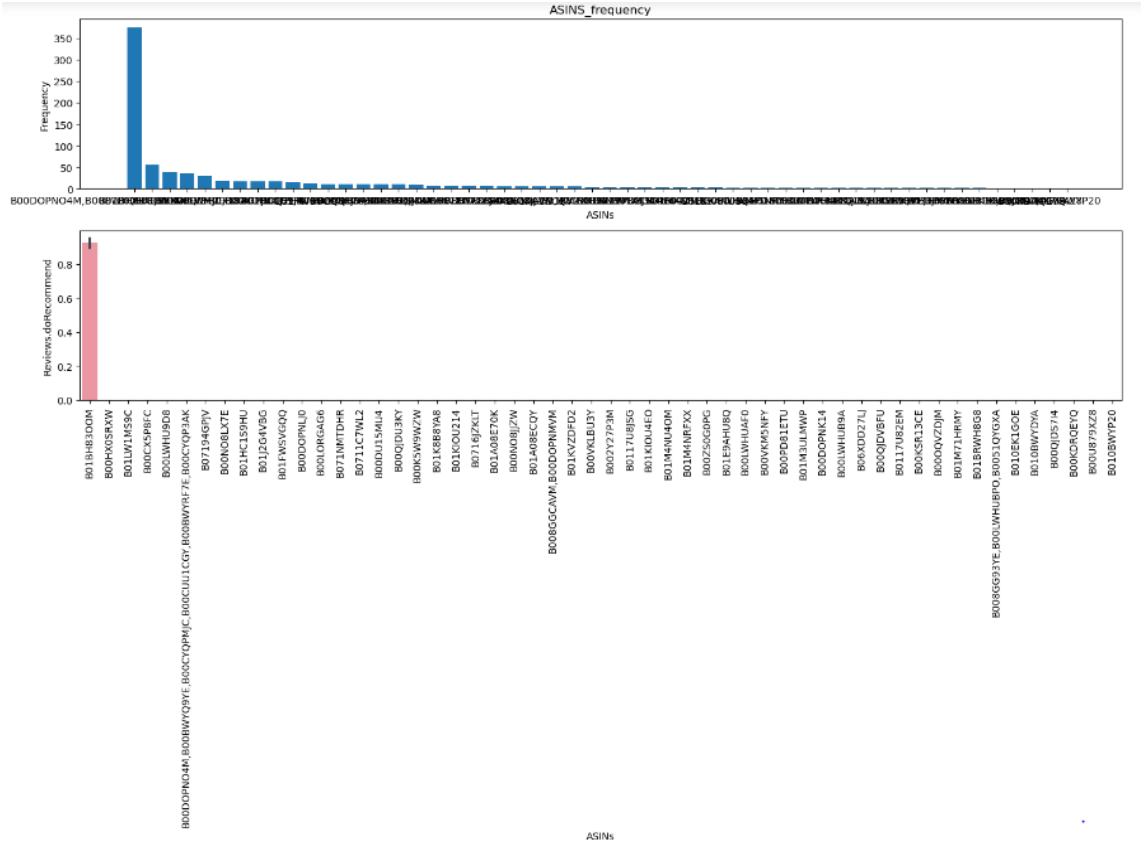


- This first subplot illustrates a bar chart that displays the number of occurrences for each ASIN in the dataset.
- Specifically, each bar represents the total reviews linked to a specific ASIN.
- The main title of this subplot is "ASINs frequency".
- The second subplot presents a point plot that illustrates the relationship between ASINs and the average ratings given in reviews.
- Each ASIN is represented by a point on the plot, with the y-coordinate representing the average rating and the x-coordinate representing the ASIN.
- The vertical lines indicate the confidence interval for each ASIN's average rating.
- The ASINs are ordered based on their frequency count (order=count_asins).
- The x-axis labels are rotated 90 degrees to improve readability.

```
asins_count = reviews['asins'].value_counts()

fig, ax1 = plt.subplots(2, 1, figsize=(16,12))

ax1[0].bar(asins_count.index, asins_count.values)
ax1[0].set_title('ASINS_frequency')
ax1[0].set_xlabel('ASINs')
ax1[0].set_ylabel('Frequency')

sns.barplot(x="asins", y="reviews.doRecommend", data=reviews, order=asins_count.index, ax=ax1[1])
ax1[1].set_xlabel('ASINs')
ax1[1].set_ylabel('Reviews.doRecommend')
ax1[1].tick_params(axis='x', rotation=90)

plt.tight_layout()
plt.show()
```

This next code helps us to create a visualization with two subplots to analyze the reviews dataset, likely related in between product reviews and recommendations on Amazon. It helps analyze the frequency of ASINs (product identifiers) in the reviews dataframe. And also, analyze the relationship between ASINs and the "reviews.doRecommend" variable in the reviews dataframe.

Output:



The 1st subplot displays a bar chart where the x-axis denotes unique ASINs, and the y-axis indicates the count of reviews for each ASIN. By examining this plot, can

discern ASINs with the highest number of reviews, represented by the tallest bars, as well as those with fewer reviews, indicated by shorter bars. Such analysis aids in identifying potentially popular products, denoted by a high review count, or less common products within the dataset.

The 2nd subplot illustrates a bar plot where each point corresponds to a review for a specific ASIN. The x-axis denotes the ASINs, likely arranged according to their frequency as shown in subplot 1 (assuming count is sorted). The y-axis represents the values in the "reviews.doRecommend" variable, typically indicating recommendations with 0 or 1, signifying not recommended or recommended, respectively.

By examining this plot, can discuss patterns in recommendations across various products (ASINs). Are there ASINs with consistently high or low recommendation ratings, as depicted by the y-axis position of the points?

For the information from subplot 1 (frequency), may potentially identify popular products (exhibited by a high review count in subplot 1) that also have high or low recommendation ratings (depicted in subplot 2).

```
reviews_counts = reviews['asins'].value_counts().reset_index()
reviews_counts.columns = ['asins', 'counts']

avg_revs_rating = reviews.groupby("asins")['reviews.rating'].mean().reset_index()
avg_revs_rating.columns = ['asins', 'avg_revs_rating']

result = pd.merge(reviews_counts, avg_revs_rating, on='asins')

print(result)
```

From those code, found that creates a dataframe summarizing the frequency of ASINs in the original ‘reviews’ data. It tells how many reviews exist for each unique product. It calculates the average rating for all reviews associated with each unique product (ASIN). It provides a sense of the average customer sentiment towards each product based on their ratings. The merge happens on the asins column, which is present in both dataframes. This ensures corresponding rows from each dataframe are merged based on the same product identifier (ASIN).

Sentiment Analysis:

```
# 'mapping' dictionary helps to indicate the keys are the rating values and the values are the corresponding sentiments.
def sentiments(rating):                    # declare sentiments function
    mapping = {5: "Positive", 4: "Positive", 3: "Neutral", 2: "Negative", 1: "Negative"}
    return mapping.get(rating, "Unknown")     #  get() method to retrieve the sentiment corresponding to the given rating.
                                              #  absence of rating in the dictionary, it returns "Unknown".
```

Now, after some EDA analysis, start to focus on 'sentimental analysis' for this dataset. First, define the 'sentiments' function where assigns a dictionary called mapping which helps to comment either "positive" or "negative" or "neutral" based on ratings (5,4,….,1). In this code, the use of get () is to retrieve the sentiment corresponding to given rating by the customers.

```
# Call sentiments function to training and testing dataset
strshf_train['Sentiments']=strshf_train['reviews.rating'].apply(sentiments)
strshf_test['Sentiments']=strshf_test['reviews.rating'].apply(sentiments)

strshf_train["Sentiments"][:20]
```

Now, after declared the sentiment function – its time to applying this function on training & testing dataset. From the above code observed that To analyze the sentiment of review ratings, applying a function called sentiments to the reviews.rating column in both (test & train) datasets using apply(). The outcome is stored in a new column named Sentiments. Also, it is observed that it displays the first 20 sentiment analysis results as this code declares [:20] in the last line.

Output:

```
1014    Positive
192     Positive
464     Positive
982     Positive
208     Positive
1424     Neutral
1489    Positive
1546     Neutral
1510     Neutral
649     Positive
1573     Neutral
1420    Positive
478      Neutral
1377    Positive
1389    Positive
493     Negative
108     Negative
1086    Positive
961     Positive
1260     Neutral
Name: Sentiments, dtype: object
```

```
# Define variables
x_trn, x_trn_targetsentiment = strshf_train['reviews.text'], strshf_train['Sentiments']
x_tst, x_tst_targetsentiment = strshf_test['reviews.text'], strshf_test['Sentiments']

# Print lengths
print(len(x_trn), len(x_tst))
```

```
823 354
```

This code segment is responsible for dividing the text data and the associated sentiment labels into sets for training and testing in the context of sentiment analysis.
x_trn_targetsentiment: This variable holds the target sentiments related to the training data. It is archived from the 'Sentiments' column of the strshf_train dataframe.
x_tst_targetsentiment: This variable holds the target sentiments associated with the testing data. It is derived from the 'Sentiments' column of the strshf_test dataframe.
The lengths of x_trn and x_tst are printed (823, 354), providing insights into the size of the training and testing datasets.

```
# tokenisation involves breaks sentences into individual words
# stopwords: filtering unwanted words like the ,are , of ,is etc.

x_trn=x_trn.fillna(' ')               # filling na with space
x_tst=x_tst.fillna(' ')
x_trn_targetsentiment=x_trn_targetsentiment.fillna(' ')
x_tst_targetsentiment=x_tst_targetsentiment.fillna(' ')
```

The code segments you've provided are replacing any NaN values in x_trn, x_tst, x_trn_targetsentiment, and x_tst_targetsentiment with an empty space (' '). This is likely being done to make sure that there are no missing values in the data before using it for sentiment analysis. This can help prevent issues / errors that missing values might cause in the analysis.

```
# Using count vectorizer counting Text preprocessing and occurence

from sklearn.feature_extraction.text import CountVectorizer

# Fit and transform the x_trn data
count_vector = CountVectorizer()
x_trn_counts = count_vector.fit_transform(x_trn)

# Print the shape of the matrix
print(x_trn_counts.shape)          # it dislays how many samples and how many distinct words has
```

```
(823, 4763)
```

Utilizing CountVectorizer on x_trn after handling NA values by filling them with spaces will generate a sparse matrix depiction of the text information. This matrix

will be sized based on the number of documents in the training set and the count of unique words (inclusive of those substituted for the missing values) in the dataset. The matrix's non-zero elements indicate the frequency of each word within its respective document. Such vectorization provides as a preliminary phase for tasks like text classification, such as sentiment analysis. The vectorized form can subsequently provide input for different machine learning algorithms. Here, observed that has 823 samples & 4763 distinct words.

```
# Using tfidf transformer for reduces less meaning words which have higher occurence

from sklearn.feature_extraction.text import TfidfVectorizer      # Downscales stop words like "the" "are" etc.

# Create a TfidfVectorizer instance
tfidf_vectorizer = TfidfVectorizer(use_idf=False)

# Fit-transform the training data
x_trn_tfidf = tfidf_vectorizer.fit_transform(x_trn)

# Get the shape of the transformed training data
x_trn_tfidf.shape
```

```
(823, 4763)
```

TF-IDF transformers reduces the significance of words that appear often across documents (like "the" and "are"). It achieves this by dividing the number of times a word appears in a document by the total number of words in that document.

Application of Model Used for Sentimental Analysis:

Here uses of Naïve Bayes Model & Support Vector Classifier help to find out the accuracy of the results.

Why Naïve Bayes?

Naive Bayes is a classification technique rooted in Bayes` theorem. Its operation involves determining of a specific event (e.g., a sentiment category) occurring given a collection of features (e.g., words within a review). Due to its simplicity and effectiveness in handling text data we used it often. It's a probabilistic classification algorithm that is rooted on the Bayes' theorem, which calculates the probability of a class given some input features.

Why Support Vector Classifier?

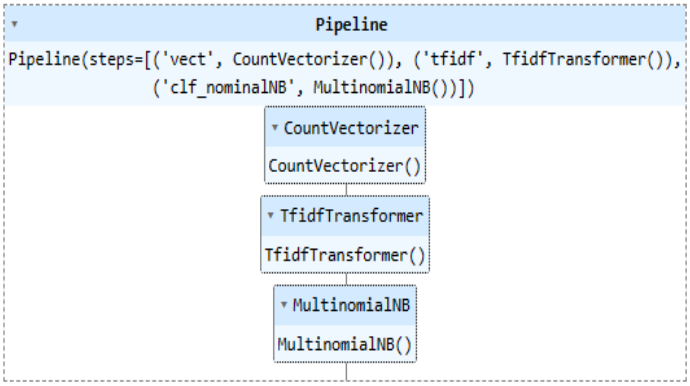
SVC, or Support Vector Classification, belongs to the category of supervised machine learning algorithms utilized for classification purposes. Within NLP, it finds application in various text classification tasks like sentiment analysis, spam detection,

and topic classification. The fundamental concept behind SVC involves identifying the most suitable hyperplane that segregates data points into distinct classes, ensuring maximum separation margin. SVC is a powerful and dynamic machine learning algorithm for classification tasks, including text classification in NLP. It can manage both linear and non-linear classification problems, and it can be customized using various parameters.

Uses of Naïve Bayes:

In [31]:
```
from sklearn.naive_bayes import MultinomialNB            # Using Naive Bayes Algorithm
from sklearn.pipeline import Pipeline                    # for multiple preprocessing steps before feeding data into a fina
from sklearn.feature_extraction.text import TfidfTransformer   # helping to represent text data in a format that is suitable

# Declare the pipeline
data_multiNB_pipe = Pipeline([
    ('vect', CountVectorizer()),
    ('tfidf', TfidfTransformer()),
    ('clf_nominalNB', MultinomialNB())
]) # using pipeline from sklearn helps a number of tasks to be implemented on every datapoint.
   # also be used when to implement multiple models on dataset in sequential manner

# Fit the pipeline on the training data
data_multiNB_pipe.fit(x_trn, x_trn_targetsentiment)
```

Out[31]:



The provided code establishes a machine learning pipeline employing the Pipeline class sourced from the sklearn.pipeline module. The pipeline consists of three sequential stages:

1. vect: Utilizing the CountVectorizer class, this step converts the textual data into a matrix reflecting token counts.
2. tfidf: Employing the TfidfTransformer class, this step converts the token count matrix into a matrix featuring TF-IDF (Term Frequency-Inverse Document Frequency) attributes.
3. clf_nominalNB: This stage employs the MultinomialNB class from the sklearn.naive_bayes module to train a Naive Bayes classifier utilizing the TF-IDF attributes.

The pipeline undergoes training on the training dataset via the fit method, enabling the classifier to learn from the provided data. The training data comprises the x_trn variable, containing the textual information, and the x_trn_targetsentiment variable, holding the associated sentiment labels.
At the end of completion of training, the Naive Bayes classifier becomes capable of predicting sentiment labels for new textual data. This prediction task is facilitated through the predict method of the pipeline. It can be used anywhere as per the requirement of the model`s dataset in sequential manner.

```
1 prediction_multiNB=data_multiNB_pipe.predict(x_tst)
2 prediction_multiNB
```

```
array(['Positive', 'Positive', 'Positive', 'Positive',
       'Positive', 'Positive', 'Positive', 'Positive',
       'Positive', 'Positive', 'Positive', 'Positive',
       'Positive', 'Positive', 'Positive', 'Positive',
       'Positive', 'Positive', 'Positive', 'Positive',
       'Positive', 'Positive', 'Positive', 'Positive',
       'Positive', 'Positive', 'Positive', 'Positive', .....
'Positive', 'Positive', 'Positive', 'Positive', 'Positive',
'Positive', 'Positive', 'Positive', 'Positive', 'Positive',
'Positive', 'Neutral', 'Positive', 'Positive', 'Positive',
'Positive', 'Positive', 'Positive', 'Positive', 'Positive',
'Positive', 'Positive', 'Positive', 'Positive', 'Positive',
'Positive', 'Neutral', 'Positive', 'Neutral', 'Positive',

                                                    .....
Positive', 'Positive', 'Positive', 'Positive', 'Positive',
Positive', 'Positive', 'Positive', 'Neutral', 'Positive',
Positive', 'Positive', 'Positive', 'Positive', 'Positive',
Positive', 'Positive', 'Positive', 'Positive', 'Positive',
Positive', 'Positive', 'Positive', 'Positive', 'Positive',
Positive', 'Positive', 'Positive', 'Positive', 'Positive',
Positive', 'Positive', 'Positive', 'Positive', 'Neutral',
Positive', 'Positive', 'Positive', 'Positive'], dtype='<U8')
```

This code is performing sentiment prediction using a trained Naive Bayes classifier. prediction_multiNB=data_multiNB_pipe.predict(x_tst): it utilizes the trained pipeline data_multiNB_pipe to predict sentiment labels for the test data (x_tst). This code segment generates predictions for sentiment labels based on the test data using a Naive Bayes classifier trained within a pipeline structure. The variable prediction_multiNB holds these predicted labels for further analysis or evaluation.

```
# Perform accuracy using MultinomialNB model

from sklearn.metrics import accuracy_score
print("Accuracy: {}".format(accuracy_score(x_tst_targetsentiment,prediction_multiNB)))
```

```
Accuracy: 0.8615819209039548
```

This part of code indicates the accuracy score of the NB model. It calculates the accuracy by comparing the actual sentiment labels in the test data with the labels the model predicted. A higher accuracy score ~ 86% (closer to 100%) indicates the model performed better at classifying sentiment in the test data.

```
 # Perform accuracy based on Linear Support Vector Classifier

from sklearn.svm import LinearSVC
text_clf = Pipeline([("vect", CountVectorizer()),
                                ("tfidf", TfidfTransformer()),
                                ("clf_linearSVC", LinearSVC())])
text_clf.fit(x_trn, x_trn_targetsentiment)

predicted_LinearSVC = text_clf.predict(x_tst)
print('Accuracy: {}'. format(accuracy_score(x_tst_targetsentiment, predicted_LinearSVC)))
```

```
Accuracy: 0.884180790960452
```

This code segment indicates to uses of Support Vector Classifier for trained the dataset. It constructs a pipeline for text classification using LinearSVC as the classifier. It helps to trains the pipeline on the training data (x_trn and x_trn_targetsentiment). Then, it predicts sentiment labels for the test data (x_tst) and evaluates the accuracy of the predictions. The accuracy score is ~ 88% which indicates this model also performed better.

```
test_text=["it works well it takes time for it to know you",
            "Built on Android does not mean you can",
              "I will not recomend this porduct",
          ]
text_clf.predict(test_text)
```

```
array(['Positive', 'Positive', 'Negative'], dtype=object)
```

Observed that this code performs sentiment analysis on a list of new text samples using the trained pipeline (text_clf). The predict () method takes the new text samples (test_text) as input and returns the predicted sentiment labels for each sample.

Conclusions:

So, here concluded our report about Sentimental Analysis of Amazon Products Review with the help of Naive Bayes & Support Vector Classifier algorithm of NLP. Uses these both algorithm & EDA analysis we discovered some possibles conclusions:

- The high overall rating suggests a high level of customer satisfaction with the product.
- The large number of reviews in the electronics category suggests its popularity among consumers.
- Reviewers who provide higher ratings generally tend to compose lengthier reviews, implying a deeper level of engagement with the product.
- Using Naive Bayes & Support Vector Classifier algorithm helps to classify reviews “positive”,” negative”, “neutral” sentiment.
- If the sentiment is predominantly positive, highlight the product's strengths.
- If negative sentiment is identified, suggest areas for improvement based on recurring themes in the reviews.
- This analysis also predicts the reviewer’s behavior to identify the patterns in the reviewer’s behavior, such as the reviewer's rating, review length, or review frequency.
- The reviewers who give high ratings are prone to write longer reviews, suggesting that they are more engaged with the product.
- For sentimental analysis – we found that Support Vector Classifier accuracy (88%) analysis was better than Naïve Bayes analysis (86%).

In conclusion, our NLP analysis indicates valuable insights into customer sentiment and product perception gleaned from Amazon reviews. This understanding empowers businesses to refine their products and marketing approaches, ensuring better alignment with customer preferences and needs.

Also, have some **limitations** like:

- Potential data bias, such as more negative reviews reflecting customer dissatisfaction.
- Limitations of NLP methods, such as difficulties in interpreting sarcasm or complex language nuances.

For Future purpose we can go for deeper analysis for more specific outcomes.

Reference:

1. Bing Liu, Lei Zhang, Sooyun Wang, and Junsheng Lin, 2009, A Sentiment Lexicon for Analyzing Social Media Text, https://aclanthology.org/W13-1617.pdf
2. Pang, B., Lee, L., & Vaithyanathan, S., 2004, Sentiment Analysis: Finding the Opinions on People, Places, and Events. https://aclanthology.org/N18-1171
3. Yoav Goldberg,2014, Natural Language Processing with Deep Learning. https://aclanthology.org/people/y/yoav-goldberg/
4. Jurafsky, Daniel & Martin, James H.,2000, https://web.stanford.edu/~jurafsky/slp3/
5. Jacob Devlin et al., "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding" https://arxiv.org/abs/1810.04805
6. M.R. Berry and A.S. Linoff.,"Text Mining and Analysis: Decision-making and Discovery".

MA336_AI & ML Project - Amazon Product Reviews (Sentimental Analysis)

In [1]:
```
# import libraries
import pandas as pd              # for data cleaning, reshaping, merging, and more.
import numpy as np               # for array operations, mathematical functions, linea

import nltk                      # import the Natural Language Toolkit (NLTK) for text
import re, random, os            # 're' module, for search, match, and manipulate stri

import matplotlib.pyplot as plt  # for MATLAB-like plotting framework including line p
import seaborn as sns            # for creating visually appealing statistical graphic
import math

!pip install spacy
import spacy                                # for basic preprocessing (optional)
!python -m spacy download en_core_web_sm    # for downloads the English language model,

from sklearn.model_selection import StratifiedShuffleSplit  # for test & training the

import matplotlib.pyplot as plt
%matplotlib inline

import warnings
warnings.filterwarnings("ignore",category=DeprecationWarning)   # for managing warning
```

```
Requirement already satisfied: spacy in c:\users\mantosh.nandy\anaconda3\lib\site-pac
kages (3.7.4)
Requirement already satisfied: spacy-legacy<3.1.0,>=3.0.11 in c:\users\mantosh.nandy
\anaconda3\lib\site-packages (from spacy) (3.0.12)
Requirement already satisfied: spacy-loggers<2.0.0,>=1.0.0 in c:\users\mantosh.nandy
\anaconda3\lib\site-packages (from spacy) (1.0.5)
Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in c:\users\mantosh.nandy\an
aconda3\lib\site-packages (from spacy) (1.0.10)
Requirement already satisfied: cymem<2.1.0,>=2.0.2 in c:\users\mantosh.nandy\anaconda
3\lib\site-packages (from spacy) (2.0.8)
Requirement already satisfied: preshed<3.1.0,>=3.0.2 in c:\users\mantosh.nandy\anacon
da3\lib\site-packages (from spacy) (3.0.9)
Requirement already satisfied: thinc<8.3.0,>=8.2.2 in c:\users\mantosh.nandy\anaconda
3\lib\site-packages (from spacy) (8.2.3)
Requirement already satisfied: wasabi<1.2.0,>=0.9.1 in c:\users\mantosh.nandy\anacond
a3\lib\site-packages (from spacy) (1.1.2)
Requirement already satisfied: srsly<3.0.0,>=2.4.3 in c:\users\mantosh.nandy\anaconda
3\lib\site-packages (from spacy) (2.4.8)
Requirement already satisfied: catalogue<2.1.0,>=2.0.6 in c:\users\mantosh.nandy\anac
onda3\lib\site-packages (from spacy) (2.0.10)
Requirement already satisfied: weasel<0.4.0,>=0.1.0 in c:\users\mantosh.nandy\anacond
a3\lib\site-packages (from spacy) (0.3.4)
Requirement already satisfied: typer<0.10.0,>=0.3.0 in c:\users\mantosh.nandy\anacond
a3\lib\site-packages (from spacy) (0.9.4)
Requirement already satisfied: smart-open<7.0.0,>=5.2.1 in c:\users\mantosh.nandy\ana
conda3\lib\site-packages (from spacy) (5.2.1)
Requirement already satisfied: tqdm<5.0.0,>=4.38.0 in c:\users\mantosh.nandy\anaconda
3\lib\site-packages (from spacy) (4.65.0)
Requirement already satisfied: requests<3.0.0,>=2.13.0 in c:\users\mantosh.nandy\anac
onda3\lib\site-packages (from spacy) (2.31.0)
Requirement already satisfied: pydantic!=1.8,!=1.8.1,<3.0.0,>=1.7.4 in c:\users\manto
sh.nandy\anaconda3\lib\site-packages (from spacy) (1.10.8)
Requirement already satisfied: jinja2 in c:\users\mantosh.nandy\anaconda3\lib\site-pa
ckages (from spacy) (3.1.2)
Requirement already satisfied: setuptools in c:\users\mantosh.nandy\anaconda3\lib\sit
e-packages (from spacy) (68.0.0)
Requirement already satisfied: packaging>=20.0 in c:\users\mantosh.nandy\anaconda3\li
b\site-packages (from spacy) (23.1)
Requirement already satisfied: langcodes<4.0.0,>=3.2.0 in c:\users\mantosh.nandy\anac
onda3\lib\site-packages (from spacy) (3.3.0)
Requirement already satisfied: numpy>=1.19.0 in c:\users\mantosh.nandy\anaconda3\lib
\site-packages (from spacy) (1.24.3)
Requirement already satisfied: typing-extensions>=4.2.0 in c:\users\mantosh.nandy\ana
conda3\lib\site-packages (from pydantic!=1.8,!=1.8.1,<3.0.0,>=1.7.4->spacy) (4.7.1)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\mantosh.nandy\ana
conda3\lib\site-packages (from requests<3.0.0,>=2.13.0->spacy) (2.0.4)
Requirement already satisfied: idna<4,>=2.5 in c:\users\mantosh.nandy\anaconda3\lib\s
ite-packages (from requests<3.0.0,>=2.13.0->spacy) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\mantosh.nandy\anaconda3
\lib\site-packages (from requests<3.0.0,>=2.13.0->spacy) (1.26.16)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\mantosh.nandy\anaconda3
\lib\site-packages (from requests<3.0.0,>=2.13.0->spacy) (2023.7.22)
Requirement already satisfied: blis<0.8.0,>=0.7.8 in c:\users\mantosh.nandy\anaconda3
\lib\site-packages (from thinc<8.3.0,>=8.2.2->spacy) (0.7.11)
Requirement already satisfied: confection<1.0.0,>=0.0.1 in c:\users\mantosh.nandy\ana
conda3\lib\site-packages (from thinc<8.3.0,>=8.2.2->spacy) (0.1.4)
Requirement already satisfied: colorama in c:\users\mantosh.nandy\anaconda3\lib\site-
packages (from tqdm<5.0.0,>=4.38.0->spacy) (0.4.6)
Requirement already satisfied: click<9.0.0,>=7.1.1 in c:\users\mantosh.nandy\anaconda
3\lib\site-packages (from typer<0.10.0,>=0.3.0->spacy) (8.0.4)
```

```
Requirement already satisfied: cloudpathlib<0.17.0,>=0.7.0 in c:\users\mantosh.nandy
\anaconda3\lib\site-packages (from weasel<0.4.0,>=0.1.0->spacy) (0.16.0)
Requirement already satisfied: MarkupSafe>=2.0 in c:\users\mantosh.nandy\anaconda3\li
b\site-packages (from jinja2->spacy) (2.1.1)
ERROR: Invalid requirement: '#'
```

In [2]:
```
mn_data = pd.read_csv('C:/Users/Mantosh.Nandy/OneDrive/Documents/MA 336 AI_ML/archive/
print(mn_data.shape)           # Printing the shape (number of rows & columns) of the
```

```
(1597, 27)
```

In [3]:
```
mn_data.info()       # overview of the DataFrame's structure and characteristics for a
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1597 entries, 0 to 1596
Data columns (total 27 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   id                    1597 non-null   object
 1   asins                 1597 non-null   object
 2   brand                 1597 non-null   object
 3   categories            1597 non-null   object
 4   colors                774 non-null    object
 5   dateAdded             1597 non-null   object
 6   dateUpdated           1597 non-null   object
 7   dimension             565 non-null    object
 8   ean                   898 non-null    float64
 9   keys                  1597 non-null   object
 10  manufacturer          965 non-null    object
 11  manufacturerNumber    902 non-null    object
 12  name                  1597 non-null   object
 13  prices                1597 non-null   object
 14  reviews.date          1217 non-null   object
 15  reviews.doRecommend   539 non-null    object
 16  reviews.numHelpful    900 non-null    float64
 17  reviews.rating        1177 non-null   float64
 18  reviews.sourceURLs    1597 non-null   object
 19  reviews.text          1597 non-null   object
 20  reviews.title         1580 non-null   object
 21  reviews.userCity      0 non-null      float64
 22  reviews.userProvince  0 non-null      float64
 23  reviews.username      1580 non-null   object
 24  sizes                 0 non-null      float64
 25  upc                   898 non-null    float64
 26  weight                686 non-null    object
dtypes: float64(7), object(20)
memory usage: 337.0+ KB
```

In [4]:
```
des_data=mn_data.copy()
des_data.describe()
```

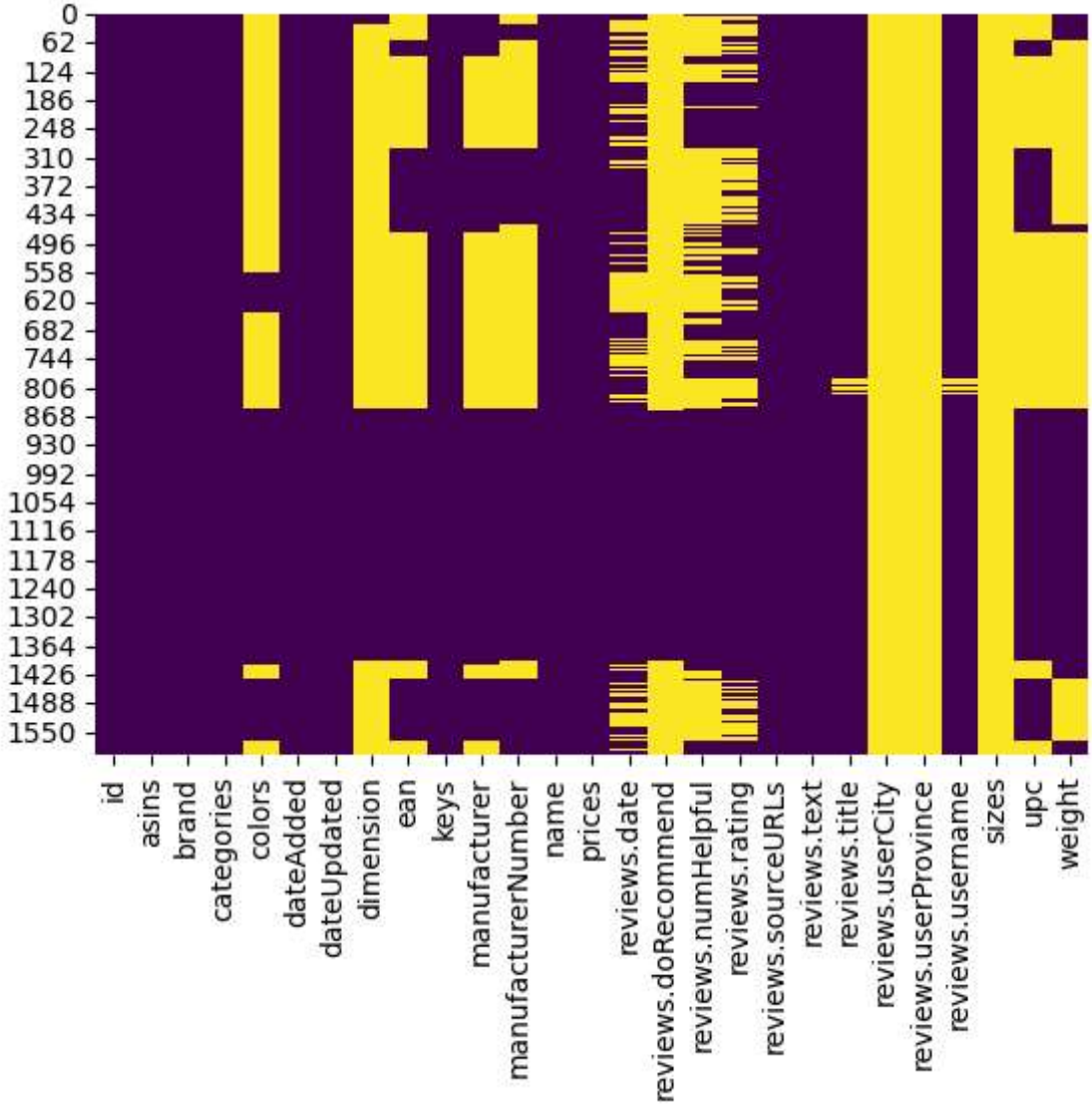
Out[4]:

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

In [5]:
```
# From Upper analysis we found mean reviews rating comes out to be 4.35.
# Also rating info and review helpful rating is important to analyse data later.
```

In [6]:
```
# Heatmap analysis
sns.heatmap(des_data.isnull(),cbar=False,cmap="viridis")
```

Out[6]:
```
<Axes: >
```



In [7]:
```
mn_data.drop(['reviews.userCity','reviews.userProvince','sizes','ean'],inplace=True,ax
mn_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1597 entries, 0 to 1596
Data columns (total 23 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   id                    1597 non-null   object 
 1   asins                 1597 non-null   object 
 2   brand                 1597 non-null   object 
 3   categories            1597 non-null   object 
 4   colors                774 non-null    object 
 5   dateAdded             1597 non-null   object 
 6   dateUpdated           1597 non-null   object 
 7   dimension             565 non-null    object 
 8   keys                  1597 non-null   object 
 9   manufacturer          965 non-null    object 
 10  manufacturerNumber    902 non-null    object 
 11  name                  1597 non-null   object 
 12  prices                1597 non-null   object 
 13  reviews.date          1217 non-null   object 
 14  reviews.doRecommend   539 non-null    object 
 15  reviews.numHelpful    900 non-null    float64
 16  reviews.rating        1177 non-null   float64
 17  reviews.sourceURLs    1597 non-null   object 
 18  reviews.text          1597 non-null   object 
 19  reviews.title         1580 non-null   object 
 20  reviews.username      1580 non-null   object 
 21  upc                   898 non-null    float64
 22  weight                686 non-null    object 
dtypes: float64(3), object(20)
memory usage: 287.1+ KB
```

In [8]:
```
mn_data.describe()
```

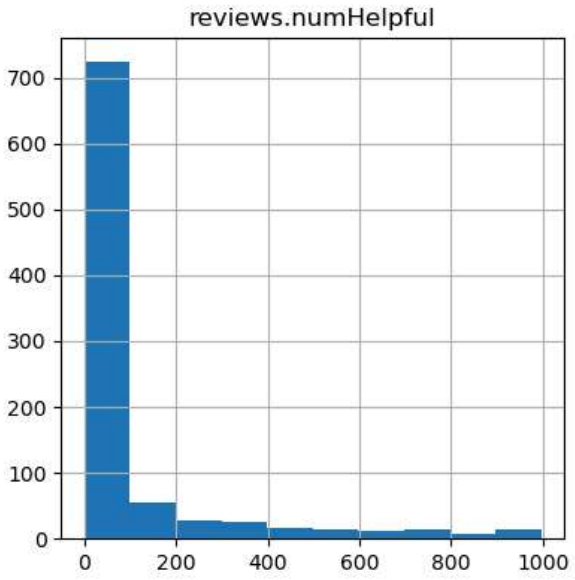
Out[8]:

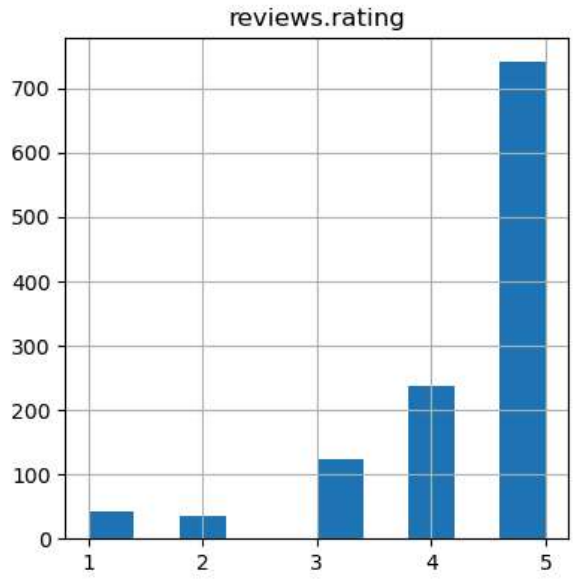
	reviews.numHelpful	reviews.rating	upc
count	900.000000	1177.000000	8.980000e+02
mean	83.584444	4.359388	8.443135e+11
std	197.150238	1.021445	3.416444e+09
min	0.000000	1.000000	8.416670e+11
25%	0.000000	4.000000	8.416670e+11
50%	0.000000	5.000000	8.416670e+11
75%	34.000000	5.000000	8.487190e+11
max	997.000000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

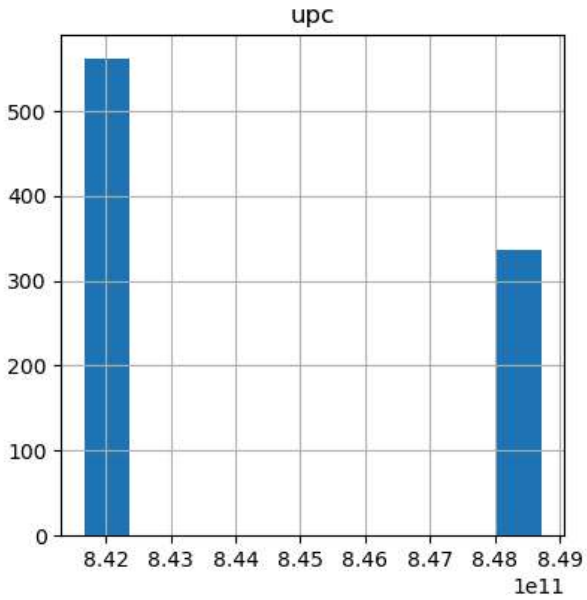
In [9]:
```
# # Calculate the number of unique asins using .unique() function
unique_asins = len(mn_data['asins'].unique())

# # Print the number of unique asins help of format method
print("Number of Unique asins: {}".format(unique_asins))
print(mn_data['asins'].unique())
```

```
Number of Unique asins: 54
['B00QJDU3KY' 'B002Y27P3M' 'B00DU15MU4' 'B01LW1MS9C' 'B01FWSVGQQ'
 'B00DOPNLJ0' 'B00NO8LX7E' 'B00LWHUAF0' 'B00KDRQEYQ' 'B00OQVZDJM'
 'B00QJDVBFU' 'B00VKLBU3Y' 'B010BWYP20' 'B00VKM5NFY' 'B01KIOU4EO'
 'B0117U8JSG' 'B01A08ECQY' 'B01J2G4VBG' 'B00QJDS7I4' 'B00U879XZ8'
 'B010EK1GOE' 'B00LWHUB9A' 'B00ZS0G0PG' 'B01A08E70K' 'B00PD81ETU'
 'B06XDD27LJ' 'B00CX5P8FC' 'B07194GPJV' 'B008GG93YE,B00LWHUBPO,B0051QYGXA'
 'B071NMTDHR' 'B0117U82EM' 'B0711C7WL2' 'B01M3ULMWP' 'B00LWHU9D8'
 'B0716JZKLT' 'B01BRWH8G8' 'B01M4NRFXX' 'B01KVZDFD2' 'B01M4NU4OM'
 'B008GGCAVM,B00DOPNMVM' 'B00KSR13CE' 'B01M71HRMY' 'B01K8B8YA8'
 'B010BWYDYA' 'B01KIOU214' 'B00DOPNK14'
 'B00DOPNO4M,B00BWYQ9YE,B00CYQPMJC,B00CUU1CGY,B00BWYRF7E,B00CYQP3AK'
 'B01BH83OOM' 'B00K5W9WZW' 'B01E9AHU8Q' 'B01HC1S9HU' 'B00HX0SRXW'
 'B00LORGAG6' 'B00NO8JJZW']
```

In [10]:
```
mn_data.hist(figsize=(10,10))
plt.show()
```







Train & Test Split with Stratified Shuffle Method

In [11]:
```
# # Drop the NULL values from the reviews.rating
mn_data_after = mn_data.dropna(subset=['reviews.rating'])

# # Cast the 'reviews.rating' column as integers
mn_data_after['reviews.rating'] = mn_data_after['reviews.rating'].astype(int)

# # Before and after pre-processing - Calculate and print the number of rows
print(f"Before: {len(mn_data)} \nAfter: {len(mn_data_after)}")
```

```
Before: 1597
After: 1177
C:\Users\Mantosh.Nandy\AppData\Local\Temp\ipykernel_10524\3309294776.py:5: SettingWit
hCopyWarning:
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/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  mn_data_after['reviews.rating'] = mn_data_after['reviews.rating'].astype(int)
```

In [12]:
```
# By using StratifiedShuffleSplit, start test & train the datasets
split=StratifiedShuffleSplit(n_splits=5,test_size=0.3, random_state=1)
for train_index,test_index in split.split(mn_data_after,mn_data_after["reviews.rating"
    strshf_train=mn_data_after.iloc[train_index]
    strshf_test=mn_data_after.iloc[test_index]
strshf_train.head()
```

Out[12]:

	id	asins	brand	categories	colors	dateAdded	dateU
1014	AVpfpK8KLJeJML43BCuD	B01BH83OOM	Amazon	Amazon Devices,Home,Smart Home & Connected Liv...	Black	2017-01-04T03:51:17Z	20 13T08
192	AV1T0-J7vKc47QAVgf2T	B01FWSVGQQ	Amazon	Amazon Devices	NaN	2017-07-18T03:53:01Z	20 13T08
464	AVsRjfwAU2_QcyX9PHqe	B01J2G4VBG	Amazon	Amazon Devices & Accessories,Amazon Device Acc...	NaN	2017-03-27T20:56:09Z	20 13T08
982	AVpfpK8KLJeJML43BCuD	B01BH83OOM	Amazon	Amazon Devices,Home,Smart Home & Connected Liv...	Black	2017-01-04T03:51:17Z	20 13T08
208	AV1T19jBvKc47QAVgf3S	B00OQVZDJM	Amazon	Amazon Devices	NaN	2017-07-18T03:57:20Z	20 18T23

5 rows × 23 columns

In [13]:
```
len(strshf_train) , len(strshf_test)
```

Out[13]:
```
(823, 354)
```

In [14]:
```
# Searching for get percentage of each of the ratings of training dataset

strshf_train["reviews.rating"].value_counts(normalize=True)    # normalized to represer
```

Out[14]:
```
reviews.rating
5    0.629405
4    0.200486
3    0.105711
1    0.035237
2    0.029162
Name: proportion, dtype: float64
```

In [15]:
```
# Searching for get percentage of each of the ratings of test dataset

strshf_test["reviews.rating"].value_counts(normalize=True)    # normalized to represer
```

Out[15]:
```
reviews.rating
5    0.629944
4    0.200565
3    0.104520
1    0.036723
2    0.028249
Name: proportion, dtype: float64
```

Exploratory Data Exploration on Training Data Set

In [16]:
```
reviews=strshf_train.copy()
reviews.head(2)
```

Out[16]:

	id	asins	brand	categories	colors	dateAdded	dateU
1014	AVpfpK8KLJeJML43BCuD	B01BH83OOM	Amazon	Amazon Devices,Home,Smart Home & Connected Liv...	Black	2017-01-04T03:51:17Z	20 13T08
192	AV1T0-J7vKc47QAVgf2T	B01FWSVGQQ	Amazon	Amazon Devices	NaN	2017-07-18T03:53:01Z	20 13T08

2 rows × 23 columns

In [17]:
```
len(reviews['name'].unique()),len(reviews['asins'].unique())
```

Out[17]:
```
(62, 54)
```

In [18]:
```
reviews.groupby('asins')['name'].unique()
```

Out[18]:

asins	
B002Y27P3M	
[Kindle Keyboard]	
B008GG93YE,B00LWHUBPO,B0051QYGXA	
[Kindle]	
B008GGCAVM,B00DOPNMVM	
[Kindle Fire HD 7"]	
B00CX5P8FC	
[Amazon Fire TV]	
B00DOPNK14	
[Kindle Paperwhite]	
B00DOPNLJ0	
[Kindle Fire HDX 8.9"]	
B00DOPNO4M,B00BWYQ9YE,B00CYQPMJC,B00CUU1CGY,B00BWYRF7E,B00CYQP3AK	
[Kindle Fire HDX 7"]	
B00DU15MU4	[Certified Refur
bished Amazon Fire TV (Previou...	
B00HX0SRXW	
[Amazon Premium Headphones]	
B00K5W9WZW	[Moshi Anti-Glar
e No Bubble Screen Protector f...	
B00KDRQEYQ	[Certi
fied Refurbished Kindle E-reader]	
B00KSR13CE	
[Kindle Paperwhite]	
B00LORGAG6	[Replacement
Remote for Amazon Fire TV Stick]	
B00LWHU9D8	
[Fire HD 6 Tablet]	
B00LWHUAF0	
[Fire HD 7 Tablet]	
B00LWHUB9A	
[Fire HDX 8.9 Tablet]	
B00NO8JJZW	[Alexa Voice Rem
ote for Amazon Fire TV and Fir...	
B00NO8LX7E	[Amazon Fire TV
Game Controller, All-New Amazo...	
B00OQVZDJM	[Ki
ndle Paperwhite E-reader - Black]	
B00PD81ETU	[Certified Refur
bished Kindle Voyage E-reader ...	
B00QJDS7I4	[Ki
ndle Paperwhite E-reader - Black]	
B00QJDU3KY	
[Kindle Paperwhite]	
B00QJDVBFU	
[Kindle Paperwhite 3G]	
B00U879XZ8	[Certified Refur
bished Kindle Paperwhite E-rea...	
B00VKLBU3Y	
[Fire HD 10 Tablet with Alexa]	
B00VKM5NFY	
[Fire HD 10 Tablet with Alexa]	
B00ZS0G0PG	[Kindle Oasis E-
reader with Leather Charging C...	
B010BWYDYA	
[Fire Tablet with Alexa]	
B010BWYP20	
[Fire Kids Edition Tablet]	
B010EK1GOE	[Kindle Oasis wi

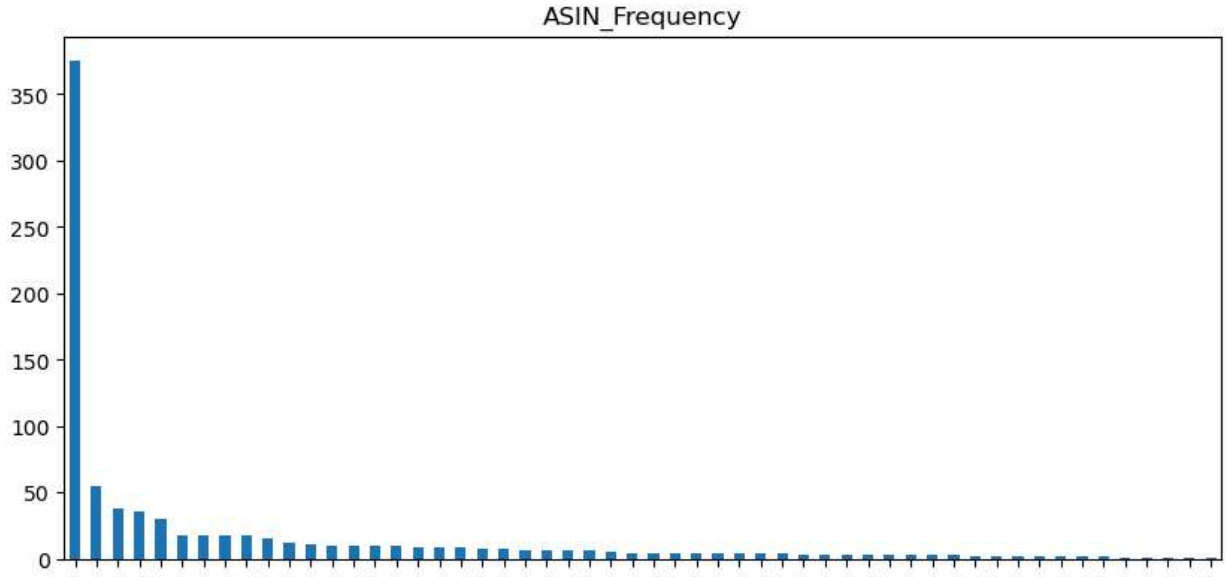
```
th Leather Charging Cover - Bl...
B0117U82EM
[Fire HD 8 Tablet]
B0117U8JSG                                                    [Certifi
ed Refurbished Fire HD 10 Tablet]
B01A08E70K
[Kindle E-reader - Black]
B01A08ECQY                                                  [Certified Ref
urbished Kindle E-reader - Black]
B01BH83OOM                                                [Amazon Tap - Al
exa-Enabled Portable Bluetooth...
B01BRWH8G8
[Fire HD 8 Tablet with Alexa]
B01E9AHU8Q                                                [Alexa Voice Rem
ote for Amazon Echo and Echo Dot]
B01FWSVGQQ                                                [Amazon Tap Slin
g Cover - White, Amazon Tap Sl...
B01HC1S9HU                                                [Amazon Kindle O
asis Premium Leather Battery C...
B01J2G4VBG                                                [Amazon 5W USB O
fficial OEM Charger and Power ...
B01K8B8YA8                                                              [E
cho Dot (2nd Generation) - Black]
B01KIOU214
[Amazon Echo - Black]
B01KIOU4EO
[Echo Show - Black]
B01KVZDFD2                                                [Kindle for Kids
Bundle with the latest Kindle...
B01LW1MS9C                                                [Amazon Echo Dot
Case (fits Echo Dot 2nd Gener...
B01M3ULMWP                                                           [All
-New Fire HD 8 Tablet with Alexa]
B01M4NRFXX                                                        [All-N
ew Fire HD 8 Kids Edition Tablet]
B01M4NU4OM                                                          [Al
l-New Fire 7 Kids Edition Tablet]
B01M71HRMY
[All-New Fire 7 Tablet with Alexa]
B06XDD27LJ                                                [Certified Refur
bished Echo Dot (2nd Generatio...
B0711C7WL2                                                [All-New Amazon
Kid-Proof Case for Amazon Fire...
B0716JZKLT                                                [All-New Amazon
Kid-Proof Case for Amazon Fire...
B07194GPJV                                                [All-New Amazon
Fire 7 Tablet Case (7th Genera...
B071NMTDHR                                                [All-New Amazon
Fire HD 8 Tablet Case (7th Gen...
Name: name, dtype: object
```

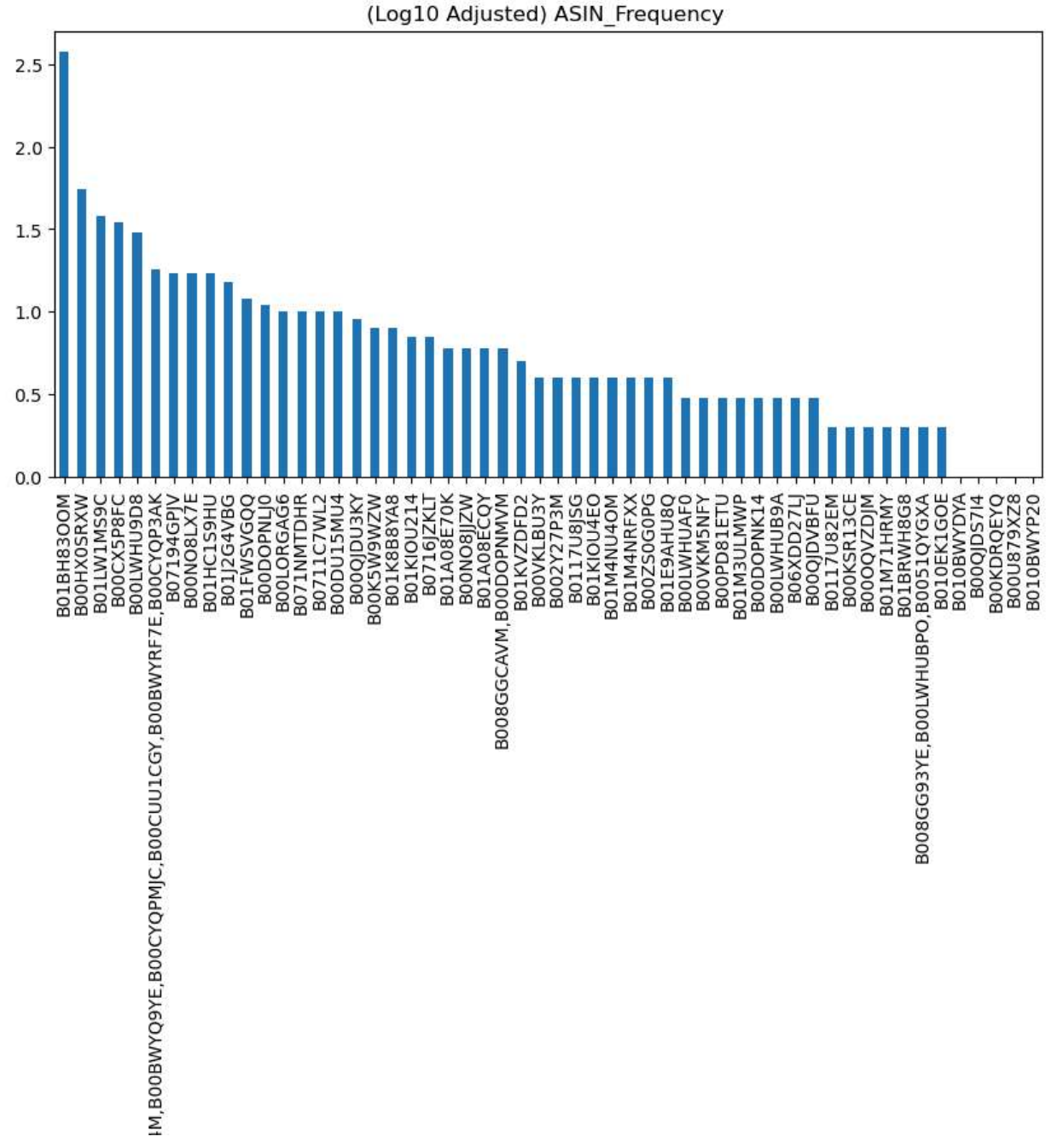
In [19]:
```
reviews.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 823 entries, 1014 to 931
Data columns (total 23 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   id                   823 non-null    object 
 1   asins                823 non-null    object 
 2   brand                823 non-null    object 
 3   categories           823 non-null    object 
 4   colors               468 non-null    object 
 5   dateAdded            823 non-null    object 
 6   dateUpdated          823 non-null    object 
 7   dimension            384 non-null    object 
 8   keys                 823 non-null    object 
 9   manufacturer         508 non-null    object 
 10  manufacturerNumber   485 non-null    object 
 11  name                 823 non-null    object 
 12  prices               823 non-null    object 
 13  reviews.date         666 non-null    object 
 14  reviews.doRecommend  373 non-null    object 
 15  reviews.numHelpful   615 non-null    float64
 16  reviews.rating       823 non-null    int32  
 17  reviews.sourceURLs   823 non-null    object 
 18  reviews.text         823 non-null    object 
 19  reviews.title        823 non-null    object 
 20  reviews.username     823 non-null    object 
 21  upc                  487 non-null    float64
 22  weight               445 non-null    object 
dtypes: float64(2), int32(1), object(20)
memory usage: 151.1+ KB
```

In [20]:

```
fig = plt.figure(figsize=(10,10))
# we use subplot when we have to see interrelationship btw two graphs
axis1 = plt.subplot(2,1,1)
axis2 = plt.subplot(2,1,2, sharex = axis1)
reviews["asins"].value_counts().plot(kind="bar", ax=axis1,
                                     title="ASIN_Frequency")

np.log10(reviews["asins"].value_counts()).plot(kind="bar", ax=axis2,
                                     title="(Log10 Adjusted) ASIN_Frequency ")
# np.log10 normalises our data to visualise the graph and difference much better
plt.show()
```

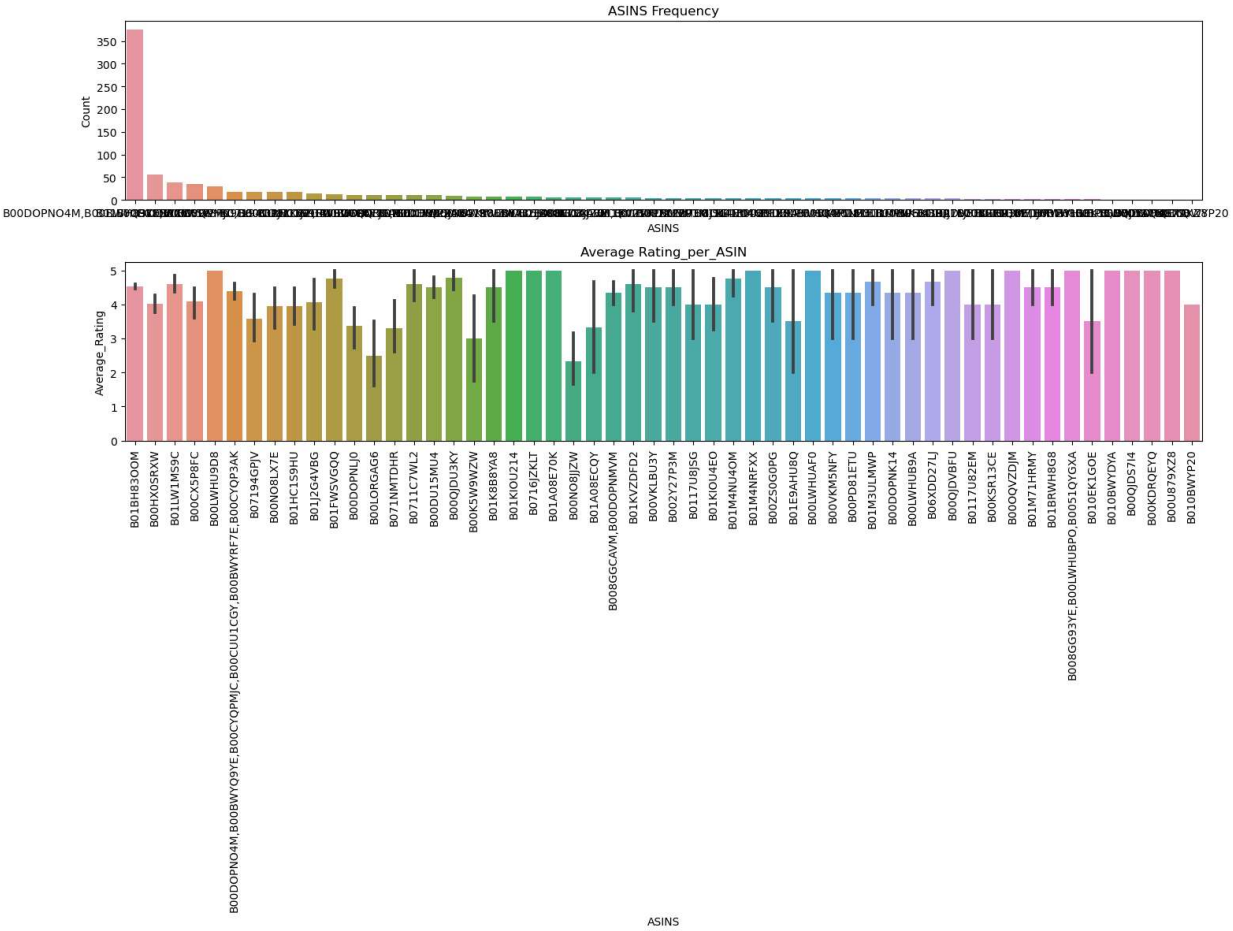

ASIN_Frequency
350
300
250
200
150
100
50
0


(Log10 Adjusted) ASIN_Frequency
2.5
2.0
1.5
1.0
0.5
0.0
B01BH83OOM
B00HX0SRXW
B01LW1MS9C
B00CX5P8FC
B00LWHU9D8
IM,B00BWYQ9YE,B00CYQPMJC,B00CUU1CGY,B00BWYRF7E,B00CYQP3AK
B07194GPJV
B00NO8LX7E
B01HC1S9HU
B01J2G4VBG
B01FWSVGQQ
B00DOPNLJ0
B00LORGAG6
B071NMTDHR
B0711C7WL2
B00DU15MU4
B00QJDU3KY
B00K5W9WZW
B01K8B8YA8
B01KIOU214
B0716JZKLT
B01A08E70K
B00NO8JJZW
B01A08ECQY
B008GGCAVM,B00DOPNMVM
B01KVZDFD2
B00VKLBU3Y
B002Y27P3M
B0117U8JSG
B01KIOU4EO
B01M4NU4OM
B01M4NRFXX
B00ZS0G0PG
B01E9AHU8Q
B00LWHUAF0
B00VKM5NFY
B00PD81ETU
B01M3ULMWP
B00DOPNK14
B00LWHUB9A
B06XDD27LJ
B00QJDVBFU
B0117U82EM
B00KSR13CE
B00OQVZDJM
B01M71HRMY
B01BRWH8G8
B008GG93YE,B00LWHUBPO,B0051QYGXA
B010EK1GOE
B010BWYDYA
B00QJDS7I4
B00KDRQEYQ
B00U879XZ8
B010BWYP20

In [21]:
```
# mean of rating from trianing data set
reviews['reviews.rating'].mean()
```

Out[21]:
```
4.359659781287971
```

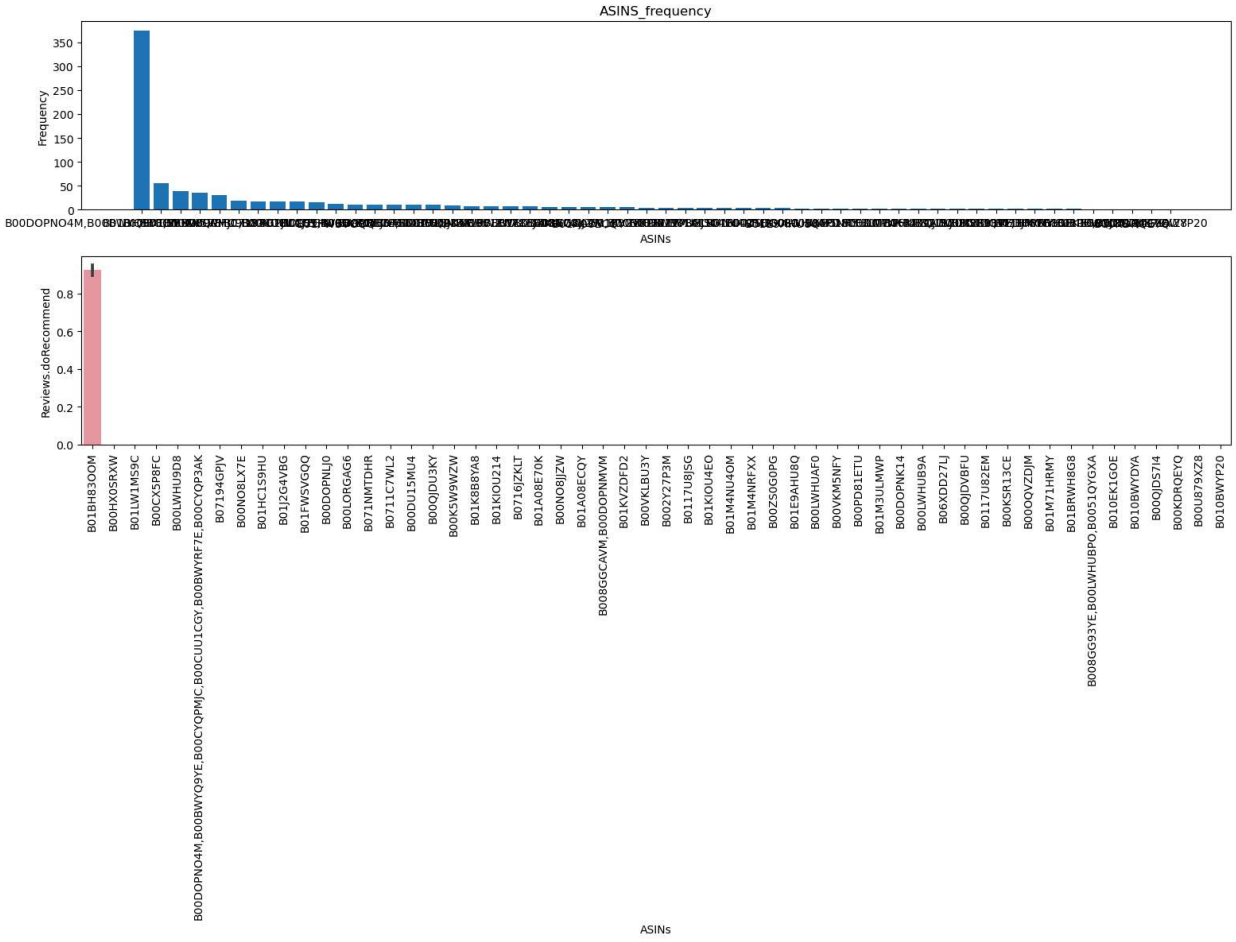
In [22]:
```
# Using pointplot to display number of those people giving ratings
count_asins=reviews['asins'].value_counts().index

# Settng up subplots
fig, axes = plt.subplots(2, 1, figsize=(16,12))

# Plotting ASINS frequency
sns.countplot(x='asins', data=reviews, order=count_asins, ax=axes[0])
axes[0].set_title('ASINS Frequency')
axes[0].set_xlabel('ASINS')
axes[0].set_ylabel('Count')

# Plotting average rating per ASIN
sns.barplot(x='asins', y='reviews.rating', data=reviews, order=count_asins, ax=axes[1]
axes[1].set_title('Average Rating_per_ASIN')
axes[1].set_xlabel('ASINS')
axes[1].set_ylabel('Average_Rating')
axes[1].tick_params(axis='x', rotation=90)

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



Reviews or Based on Recommendation Products Selling Strategy

In [23]:
```
asins_count = reviews['asins'].value_counts()

fig, ax1 = plt.subplots(2, 1, figsize=(16,12))

ax1[0].bar(asins_count.index, asins_count.values)
ax1[0].set_title('ASINS_frequency')
ax1[0].set_xlabel('ASINs')
ax1[0].set_ylabel('Frequency')

sns.barplot(x="asins", y="reviews.doRecommend", data=reviews, order=asins_count.index,
ax1[1].set_xlabel('ASINs')
ax1[1].set_ylabel('Reviews.doRecommend')
ax1[1].tick_params(axis='x', rotation=90)

plt.tight_layout()
plt.show()
```

```
C:\Users\Mantosh.Nandy\anaconda3\Lib\site-packages\seaborn\algorithms.py:98: RuntimeW
arning: Mean of empty slice
  boot_dist.append(f(*sample, **func_kwargs))
C:\Users\Mantosh.Nandy\anaconda3\Lib\site-packages\numpy\lib\nanfunctions.py:1556: Ru
ntimeWarning: All-NaN slice encountered
  return function_base._ureduce(a,
```



In [24]:
```
reviews_counts = reviews['asins'].value_counts().reset_index()
reviews_counts.columns = ['asins', 'counts']

avg_revs_rating = reviews.groupby("asins")['reviews.rating'].mean().reset_index()
avg_revs_rating.columns = ['asins', 'avg_revs_rating']

result = pd.merge(reviews_counts, avg_revs_rating, on='asins')

print(result)
```

	asins	counts	avg_revs_rating
0	B01BH83OOM	375	4.536000
1	B00HX0SRXW	55	4.018182
2	B01LW1MS9C	38	4.605263
3	B00CX5P8FC	35	4.085714
4	B00LWHU9D8	30	5.000000
5	B00DOPNO4M,B00BWYQ9YE,B00CYQPMJC,B00CUU1CGY,B0...	18	4.388889
6	B07194GPJV	17	3.588235
7	B00NO8LX7E	17	3.941176
8	B01HC1S9HU	17	3.941176
9	B01J2G4VBG	15	4.066667
10	B01FWSVGQQ	12	4.750000
11	B00DOPNLJ0	11	3.363636
12	B00LORGAG6	10	2.500000
13	B071NMTDHR	10	3.300000
14	B0711C7WL2	10	4.600000
15	B00DU15MU4	10	4.500000
16	B00QJDU3KY	9	4.777778
17	B00K5W9WZW	8	3.000000
18	B01K8B8YA8	8	4.500000
19	B01KIOU214	7	5.000000
20	B0716JZKLT	7	5.000000
21	B01A08E70K	6	5.000000
22	B00NO8JJZW	6	2.333333
23	B01A08ECQY	6	3.333333
24	B008GGCAVM,B00DOPNMVM	6	4.333333
25	B01KVZDFD2	5	4.600000
26	B00VKLBU3Y	4	4.500000
27	B002Y27P3M	4	4.500000
28	B0117U8JSG	4	4.000000
29	B01KIOU4EO	4	4.000000
30	B01M4NU4OM	4	4.750000
31	B01M4NRFXX	4	5.000000
32	B00ZS0G0PG	4	4.500000
33	B01E9AHU8Q	4	3.500000
34	B00LWHUAF0	3	5.000000
35	B00VKM5NFY	3	4.333333
36	B00PD81ETU	3	4.333333
37	B01M3ULMWP	3	4.666667
38	B00DOPNK14	3	4.333333
39	B00LWHUB9A	3	4.333333
40	B06XDD27LJ	3	4.666667
41	B00QJDVBFU	3	5.000000
42	B0117U82EM	2	4.000000
43	B00KSR13CE	2	4.000000
44	B00OQVZDJM	2	5.000000
45	B01M71HRMY	2	4.500000
46	B01BRWH8G8	2	4.500000
47	B008GG93YE,B00LWHUBPO,B0051QYGXA	2	5.000000
48	B010EK1GOE	2	3.500000
49	B010BWYDYA	1	5.000000
50	B00QJDS7I4	1	5.000000
51	B00KDRQEYQ	1	5.000000
52	B00U879XZ8	1	5.000000
53	B010BWYP20	1	4.000000

Sentiment Analysis

In [25]:
```
# 'mapping' dictionary helps to indictae the keys are the rating values and the values
def sentiments(rating):                          # declare sentiments function
    mapping = {5: "Positive", 4: "Positive", 3: "Neutral", 2: "Negative", 1: "Negative
    return mapping.get(rating, "Unknown")        #  get() method to retrieve the sentimen
                                                 #  absence of rating in the dictionary,
```

In [26]:
```
# Call sentiments function to training and testing dataset
strshf_train['Sentiments']=strshf_train['reviews.rating'].apply(sentiments)
strshf_test['Sentiments']=strshf_test['reviews.rating'].apply(sentiments)

strshf_train["Sentiments"][:20]
```

```
C:\Users\Mantosh.Nandy\AppData\Local\Temp\ipykernel_10524\2614189534.py:2: SettingWit
hCopyWarning:
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/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  strshf_train['Sentiments']=strshf_train['reviews.rating'].apply(sentiments)
C:\Users\Mantosh.Nandy\AppData\Local\Temp\ipykernel_10524\2614189534.py:3: SettingWit
hCopyWarning:
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/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  strshf_test['Sentiments']=strshf_test['reviews.rating'].apply(sentiments)
```

Out[26]:
```
1014    Positive
192     Positive
464     Positive
982     Positive
208     Positive
1424     Neutral
1489    Positive
1546     Neutral
1510     Neutral
649     Positive
1573     Neutral
1420    Positive
478      Neutral
1377    Positive
1389    Positive
493     Negative
108     Negative
1086    Positive
961     Positive
1260     Neutral
Name: Sentiments, dtype: object
```

Preparing text data

In [27]:
```
# Define variables
x_trn, x_trn_targetsentiment = strshf_train['reviews.text'], strshf_train['Sentiments'
x_tst, x_tst_targetsentiment = strshf_test['reviews.text'], strshf_test['Sentiments']
```

```
# Print Lengths
print(len(x_trn), len(x_tst))
```

```
823 354
```

Extracting features i.e., tokenizations , stopwords

In [28]:
```
# tokenisation involves breaks sentences into individual words
# stopwords: filtering unwanted words like the ,are , of ,is etc.

x_trn=x_trn.fillna(' ')              # filling na with space
x_tst=x_tst.fillna(' ')
x_trn_targetsentiment=x_trn_targetsentiment.fillna(' ')
x_tst_targetsentiment=x_tst_targetsentiment.fillna(' ')
```

In [29]:
```
# Using count vectorizer counting Text preprocessing and occurence

from sklearn.feature_extraction.text import CountVectorizer

# Fit and transform the x_train data
count_vector = CountVectorizer()
x_trn_counts = count_vector.fit_transform(x_trn)

# Print the shape of the matrix
print(x_trn_counts.shape)          # it dislays how many samples and how many distinct
```

```
(823, 4763)
```

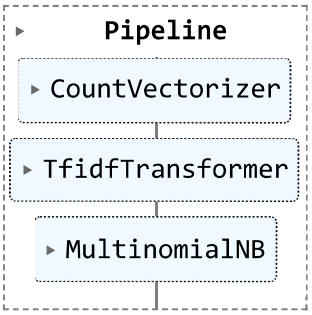
In [30]:
```
# Using tfidf transformer for reduces less meaning words which have higher occurence

from sklearn.feature_extraction.text import TfidfVectorizer      # Downscales stop wor

# Create a TfidfVectorizer instance
tfidf_vectorizer = TfidfVectorizer(use_idf=False)

# Fit-transform the training data
x_trn_tfidf = tfidf_vectorizer.fit_transform(x_trn)

# Get the shape of the transformed training data
x_trn_tfidf.shape
```

Out[30]:
```
(823, 4763)
```

In [31]:
```
from sklearn.naive_bayes import MultinomialNB             # Using Naive Bayes Algorith
from sklearn.pipeline import Pipeline                     # for multiple preprocessing
from sklearn.feature_extraction.text import TfidfTransformer   # helping to represent

# Declare the pipeline
data_multiNB_pipe = Pipeline([
    ('vect', CountVectorizer()),
    ('tfidf', TfidfTransformer()),
    ('clf_nominalNB', MultinomialNB())
]) # using pipeline from sklearn helps a number of tasks to be implemented on every da
   # also be used when to implement multiple models on dataset in sequential manner
```

```
# Fit the pipeline on the training data
data_multiNB_pipe.fit(x_trn, x_trn_targetsentiment)
```

Out[31]:



In [32]:

```
prediction_multiNB=data_multiNB_pipe.predict(x_tst)
prediction_multiNB
```

Out[32]:
```
array(['Positive', 'Positive', 'Positive', 'Positive', 'Positive',
       'Positive', 'Positive', 'Positive', 'Positive', 'Positive',
       'Positive', 'Positive', 'Positive', 'Positive', 'Positive',
       'Positive', 'Positive', 'Positive', 'Positive', 'Positive',
       'Positive', 'Positive', 'Positive', 'Positive', 'Neutral',
       'Positive', 'Positive', 'Positive', 'Positive', 'Positive',
       'Positive', 'Positive', 'Positive', 'Positive', 'Positive',
       'Positive', 'Positive', 'Positive', 'Positive', 'Positive',
       'Positive', 'Positive', 'Positive', 'Positive', 'Positive',
       'Positive', 'Positive', 'Positive', 'Positive', 'Positive',
       'Positive', 'Positive', 'Positive', 'Positive', 'Positive',
       'Positive', 'Positive', 'Positive', 'Positive', 'Positive',
       'Positive', 'Positive', 'Positive', 'Positive', 'Positive',
       'Positive', 'Positive', 'Positive', 'Positive', 'Positive',
       'Positive', 'Positive', 'Positive', 'Positive', 'Positive',
       'Positive', 'Positive', 'Positive', 'Positive', 'Positive',
       'Positive', 'Positive', 'Positive', 'Positive', 'Positive',
       'Positive', 'Positive', 'Positive', 'Positive', 'Positive',
       'Positive', 'Positive', 'Positive', 'Positive', 'Positive',
       'Positive', 'Positive', 'Positive', 'Positive', 'Positive',
       'Positive', 'Positive', 'Positive', 'Positive', 'Positive',
       'Positive', 'Positive', 'Positive', 'Positive', 'Positive',
       'Positive', 'Neutral', 'Positive', 'Positive', 'Positive',
       'Positive', 'Positive', 'Positive', 'Positive', 'Positive',
       'Positive', 'Positive', 'Positive', 'Positive', 'Positive',
       'Positive', 'Neutral', 'Positive', 'Neutral', 'Positive',
       'Positive', 'Positive', 'Positive', 'Positive', 'Positive',
       'Positive', 'Positive', 'Positive', 'Positive', 'Positive',
       'Positive', 'Positive', 'Positive', 'Positive', 'Positive',
       'Positive', 'Positive', 'Positive', 'Positive', 'Positive',
       'Positive', 'Positive', 'Positive', 'Positive', 'Positive',
       'Positive', 'Positive', 'Positive', 'Positive', 'Positive',
       'Positive', 'Positive', 'Positive', 'Neutral', 'Positive',
       'Positive', 'Positive', 'Positive', 'Positive', 'Positive',
       'Positive', 'Positive', 'Positive', 'Positive', 'Positive',
       'Positive', 'Positive', 'Positive', 'Positive', 'Positive',
       'Positive', 'Positive', 'Positive', 'Positive', 'Positive',
       'Positive', 'Positive', 'Positive', 'Positive', 'Positive',
       'Positive', 'Positive', 'Positive', 'Positive', 'Positive',
       'Positive', 'Positive', 'Positive', 'Positive', 'Positive',
       'Positive', 'Positive', 'Positive', 'Positive', 'Positive',
       'Neutral', 'Positive', 'Positive', 'Positive', 'Positive',
       'Positive', 'Positive', 'Positive', 'Positive', 'Positive',
       'Positive', 'Positive', 'Positive', 'Positive', 'Positive',
       'Positive', 'Positive', 'Positive', 'Positive', 'Positive',
       'Positive', 'Positive', 'Positive', 'Neutral', 'Positive',
       'Positive', 'Positive', 'Positive', 'Positive', 'Positive',
       'Positive', 'Positive', 'Positive', 'Positive', 'Positive',
       'Positive', 'Positive', 'Positive', 'Positive', 'Positive',
       'Positive', 'Positive', 'Positive', 'Positive', 'Positive',
       'Positive', 'Positive', 'Positive', 'Positive', 'Positive',
       'Positive', 'Positive', 'Positive', 'Positive', 'Positive',
       'Positive', 'Positive', 'Positive', 'Positive', 'Positive',
       'Positive', 'Positive', 'Positive', 'Positive', 'Positive',
       'Positive', 'Positive', 'Positive', 'Positive', 'Positive',
       'Positive', 'Positive', 'Positive', 'Positive', 'Positive',
       'Positive', 'Positive', 'Positive', 'Positive', 'Positive',
       'Positive', 'Positive', 'Positive', 'Positive', 'Positive',
       'Neutral', 'Positive', 'Positive', 'Positive', 'Positive',
       'Neutral', 'Positive', 'Positive', 'Positive', 'Positive',
```

```
       'Positive', 'Positive', 'Positive', 'Positive', 'Positive',
       'Positive', 'Positive', 'Positive', 'Positive', 'Positive',
       'Positive', 'Positive', 'Positive', 'Positive', 'Positive',
       'Positive', 'Positive', 'Positive', 'Positive', 'Positive',
       'Positive', 'Positive', 'Positive', 'Neutral', 'Positive',
       'Positive', 'Positive', 'Positive', 'Positive', 'Positive',
       'Positive', 'Positive', 'Positive', 'Positive', 'Positive',
       'Positive', 'Positive', 'Positive', 'Positive', 'Positive',
       'Positive', 'Positive', 'Positive', 'Positive', 'Positive',
       'Positive', 'Positive', 'Positive', 'Positive', 'Neutral',
       'Positive', 'Positive', 'Positive', 'Positive'], dtype='<U8')
```

In [33]:
```
# Perform accuracy using MultinomialNB model

from sklearn.metrics import accuracy_score
print("Accuracy: {}".format(accuracy_score(x_tst_targetsentiment,prediction_multiNB)))
```

```
Accuracy: 0.8615819209039548
```

In [34]:
```
x_tst
```

Out[34]:
```
3       I bought one of the first Paperwhites and have...
80      I am not a casual user of on-demand content an...
500     I got my first Kindle. This is a nice ebook re...
534     In this day and age of rectangles with screens...
101     This is perfect. A lot of people were commenti...
                              ...
1559    While I've purchased items from Amazon for yea...
454     According to the info, the Paperwhite charges ...
381     First I would like to say that I am coming fro...
1287    I have the Echo and I just love it... I bought...
698     Love it! I purchased this as I have had audibl...
Name: reviews.text, Length: 354, dtype: object
```

Applying model on input text

In [35]:
```
 # Perform accuracy based on Linear Support Vector Classifier

from sklearn.svm import LinearSVC
text_clf = Pipeline([("vect", CountVectorizer()),
                                ("tfidf", TfidfTransformer()),
                                ("clf_linearSVC", LinearSVC())])
text_clf.fit(x_trn, x_trn_targetsentiment)

predicted_LinearSVC = text_clf.predict(x_tst)
print('Accuracy: {}'. format(accuracy_score(x_tst_targetsentiment, predicted_LinearSVC
```

```
Accuracy: 0.884180790960452
```

```
C:\Users\Mantosh.Nandy\anaconda3\Lib\site-packages\sklearn\svm\_classes.py:32: Future
Warning: The default value of `dual` will change from `True` to ``'auto'`` in 1.5. Set
the value of `dual` explicitly to suppress the warning.
  warnings.warn(
```

In [36]:
```
test_text=["it works well it takes time for it to know you",
            "Built on Android does not mean you can",
              "I will not recomend this porduct",
          ]
text_clf.predict(test_text)
```

Out[36]:
```
array(['Positive', 'Positive', 'Negative'], dtype=object)
```

In [ ]:

In [ ]: