**Code Explanation**

**Data loading**

raw\_reviews = pd.read\_csv('Musical\_instruments\_reviews.csv')

## print shape of dataset with rows and columns and information

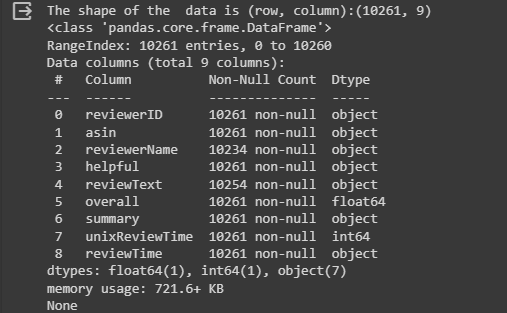
print ("The shape of the  data is (row, column):"+ str(raw\_reviews.shape))

print (raw\_reviews.info())

Explanation-

this code is loading a CSV file into a pandas DataFrame, and then printing out the shape and information of that DataFrame. This is typically one of the first steps in exploring a new dataset. It helps to understand how large the dataset is and what kind of data it contains.

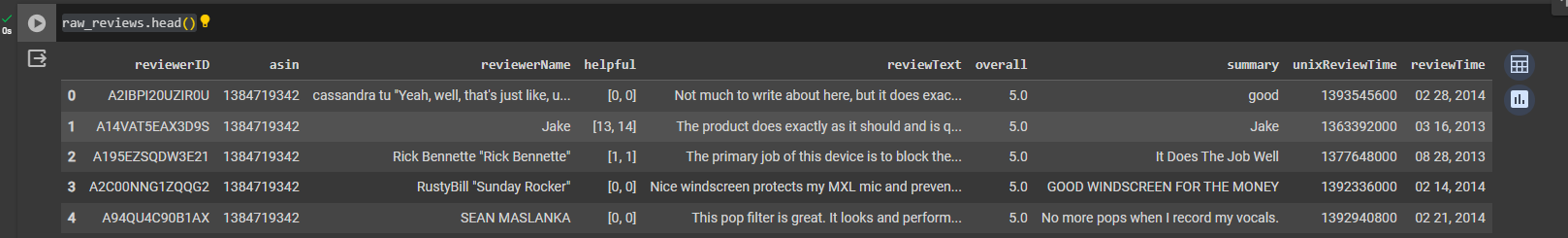
* raw\_reviews = pd.read\_csv('Musical\_instruments\_reviews.csv'): This line is using the read\_csv function from the pandas library to read the CSV (Comma Separated Values) file named ‘Musical\_instruments\_reviews.csv’. The data from this file is being stored in the raw\_reviews DataFrame.
* print ("The shape of the data is (row, column):"+ str(raw\_reviews.shape)): This line is printing the shape of the DataFrame raw\_reviews. The shape attribute of a DataFrame returns a tuple representing the dimensionality of the DataFrame. The output will be in the format of (rows, columns).
* print (raw\_reviews.info()): This line is printing information about the DataFrame raw\_reviews using the info() function. This function is used to print a concise summary of a DataFrame including the index dtype and column dtypes, non-null values and memory usage. This can be used to quickly understand the structure and data types of the DataFrame.



raw\_reviews.head()

Explanation-

The head() function is used to get the first n rows of a DataFrame. By default, it returns the first 5 rows. So, raw\_reviews.head() will display the first 5 rows of the raw\_reviews DataFrame.



**Data Preprocessing and cleaning**

We got to do lot of preprocessing before sending the reviews to the model. Let's go step by step.

Handling NaN values

Let's check for null values

#Creating a copy

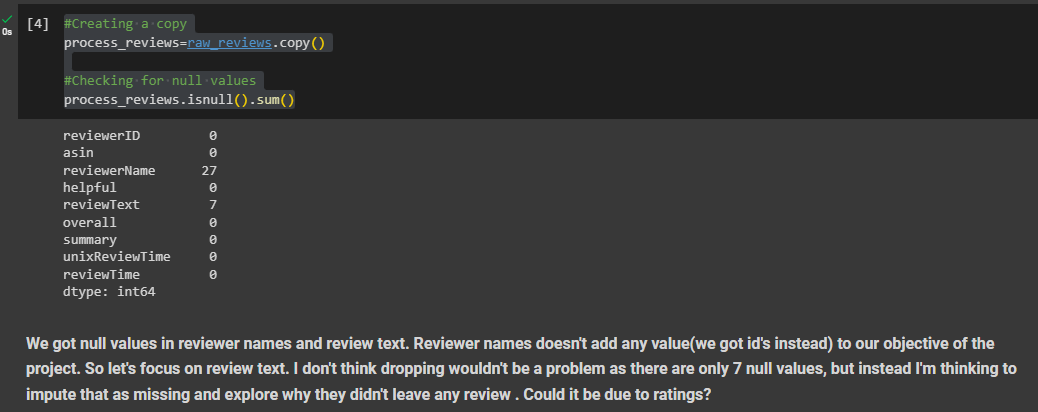
process\_reviews=raw\_reviews.copy()

#Checking for null values

process\_reviews.isnull().sum()

Explanation-

* process\_reviews=raw\_reviews.copy(): This line is creating a copy of the raw\_reviews DataFrame and storing it in process\_reviews. This is often done to avoid modifying the original data. Any changes made to the process\_reviews DataFrame will not affect the raw\_reviews DataFrame.
* process\_reviews.isnull().sum(): This line is checking for null (or missing) values in the process\_reviews DataFrame. The isnull() function returns a DataFrame where each cell is either True (if the original cell contained a null/missing value) or False. By chaining the sum() function, you’re effectively counting the number of True (i.e., null) values in each column.



process\_reviews['reviewText']=process\_reviews['reviewText'].fillna('Missing')

Explanation-

This line of code is filling the missing or null values in the ‘reviewText’ column of the process\_reviews DataFrame with the string ‘Missing’.

* process\_reviews['reviewText']: This is selecting the ‘reviewText’ column from the process\_reviews DataFrame.
* .fillna('Missing'): The fillna() function is used to fill NA/NaN values using the specified method. In this case, it’s replacing the NaN values with the string ‘Missing’.

After this line of code is executed, there will be no null values in the ‘reviewText’ column of the process\_reviews DataFrame. Any previously missing review text will now be represented as ‘Missing’. This can be a useful step in data preprocessing, especially before performing operations that do not handle null values well.

**Concatenating review and text and summary**

Let's combine review text and summary column. The sentiments won't be contradicting in nature.

process\_reviews['reviews']=process\_reviews['reviewText']+process\_reviews['summary']

process\_reviews=process\_reviews.drop(['reviewText', 'summary'], axis=1)

process\_reviews.head()

Explanation-

* process\_reviews['reviews']=process\_reviews['reviewText']+process\_reviews['summary']: This line is creating a new column in the process\_reviews DataFrame called ‘reviews’. This column is the result of concatenating (adding together) the ‘reviewText’ and ‘summary’ columns. After this line, each entry in the ‘reviews’ column will be a string that starts with the review text and ends with the summary.
* process\_reviews=process\_reviews.drop(['reviewText', 'summary'], axis=1): This line is dropping the ‘reviewText’ and ‘summary’ columns from the process\_reviews DataFrame. The drop function removes specified labels from rows or columns. Here, axis=1 denotes that column labels should be dropped. After this line, the ‘reviewText’ and ‘summary’ columns will no longer exist in the process\_reviews DataFrame.
* process\_reviews.head(): This line will display the first 5 rows of the process\_reviews DataFrame. It’s a way to quickly inspect the changes you’ve made.



**Creating ‘Sentiment’ column**

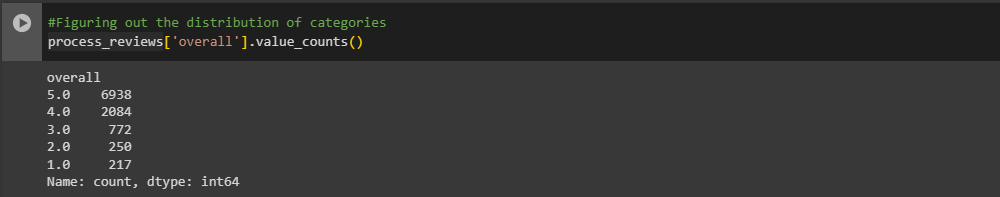
This is an important preprocessing phase, we are deciding the outcome column (sentiment of review) based on the overall score. If the score is greater than 3, we take that as positive and if the value is less than 3 it is negative If it is equal to 3, we take that as neutral sentiment

process\_reviews['overall'].value\_counts()

Explanation-

The line process\_reviews['overall'].value\_counts() is used to get a Series containing counts of unique values in descending order so that the first element is the most frequently-occurring element. It excludes NA values by default.

Here, it’s applied to the ‘overall’ column of the process\_reviews DataFrame. So, this line will return the count of each unique value in the ‘overall’ column. This can be useful to understand the distribution of values in a column.



def f(row):

    '''This function returns sentiment value based on the overall ratings from the user'''

    if row['overall'] == 3.0:

        val = 'Neutral'

    elif row['overall'] == 1.0 or row['overall'] == 2.0:

        val = 'Negative'

    elif row['overall'] == 4.0 or row['overall'] == 5.0:

        val = 'Positive'

    else:

        val = -1

    return val

Explanation-

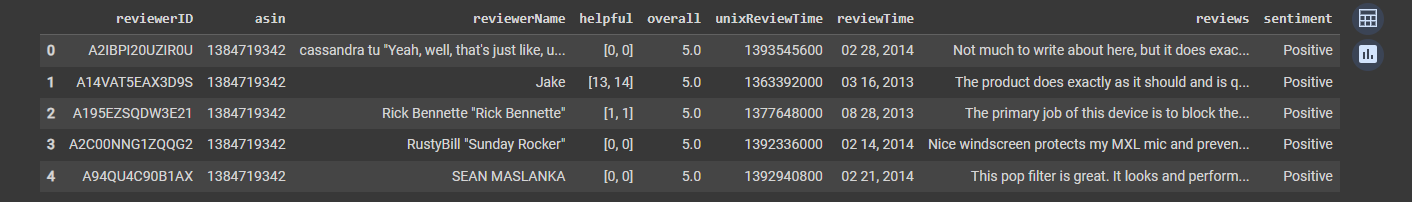
This function, f(row), is designed to categorize the ‘overall’ ratings into sentiment values. It takes a row of a DataFrame as input and checks the ‘overall’ field of that row. Here’s what it does:

* If the ‘overall’ rating is 3.0, it considers the sentiment as ‘Neutral’.
* If the ‘overall’ rating is 1.0 or 2.0, it considers the sentiment as ‘Negative’.
* If the ‘overall’ rating is 4.0 or 5.0, it considers the sentiment as ‘Positive’.
* If the ‘overall’ rating is anything else, it returns -1.
* #Applying the function in our new column
* process\_reviews['sentiment'] = process\_reviews.apply(f, axis=1)
* process\_reviews.head()

Explanation-

This code is applying the function f(row) to each row in the process\_reviews DataFrame and storing the result in a new column called ‘sentiment’.

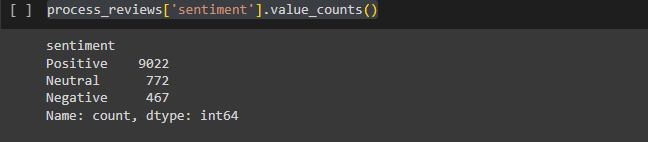
* process\_reviews['sentiment'] = process\_reviews.apply(f, axis=1): This line is using the apply() function to apply the function f(row) to each row in the DataFrame (since axis=1). The result, which will be either ‘Neutral’, ‘Negative’, ‘Positive’, or -1 depending on the ‘overall’ rating of the review, is stored in a new ‘sentiment’ column.
* process\_reviews.head(): This line will display the first 5 rows of the process\_reviews DataFrame. It’s a way to quickly inspect the changes you’ve made.



process\_reviews['sentiment'].value\_counts()

The value\_counts() function is used to get a Series containing counts of unique values in descending order so that the first element is the most frequently-occurring element. It excludes NA values by default.

Here, it’s applied to the ‘sentiment’ column of the process\_reviews DataFrame. So, this line will return the count of each unique value in the ‘sentiment’ column. This can be useful to understand the distribution of sentiments in your reviews.



**Handling time column**

Here we have an unusual review time column which has date and year, once we split both we will split the date further into month and date.

# new data frame which has date and year

new = process\_reviews["reviewTime"].str.split(",", n = 1, expand = True)

# making separate date column from new data frame

process\_reviews["date"]= new[0]

# making separate year column from new data frame

process\_reviews["year"]= new[1]

process\_reviews=process\_reviews.drop(['reviewTime'], axis=1)

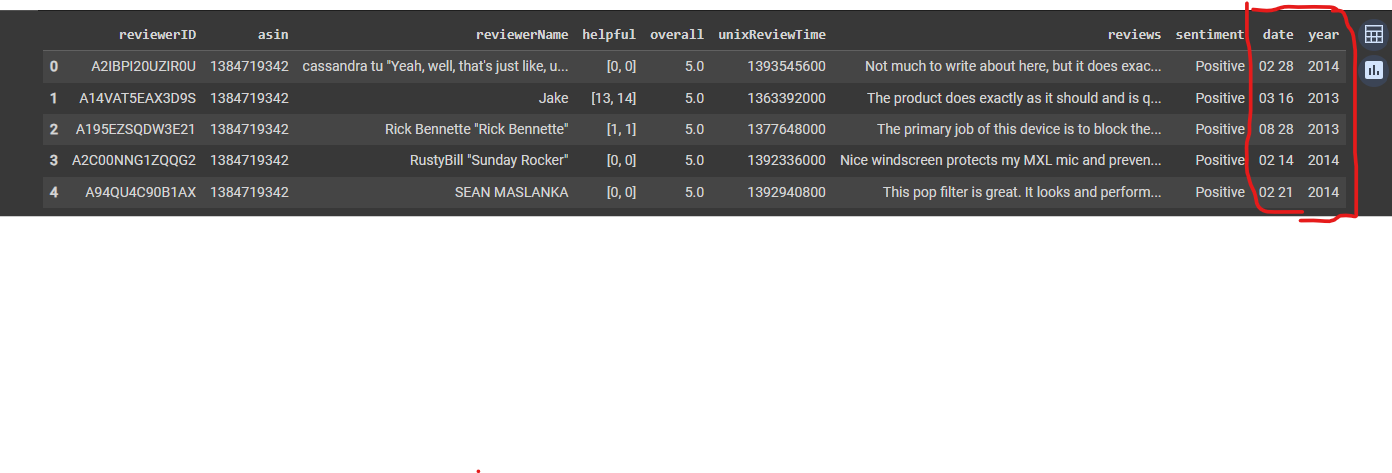
process\_reviews.head()

Explanation-

This code is splitting the ‘reviewTime’ column into two separate columns: ‘date’ and ‘year’.

* new = process\_reviews["reviewTime"].str.split(",", n = 1, expand = True): This line is splitting the ‘reviewTime’ column on the comma character. The str.split() function is used to split a string into a list where each word is a separate element. The expand=True argument means that the split strings should be expanded into separate columns. The result is stored in a new DataFrame called ‘new’.
* process\_reviews["date"]= new[0]: This line is creating a new ‘date’ column in the process\_reviews DataFrame. The values for this column are taken from the first column of the ‘new’ DataFrame.
* process\_reviews["year"]= new[1]: Similarly, this line is creating a new ‘year’ column in the process\_reviews DataFrame. The values for this column are taken from the second column of the ‘new’ DataFrame.
* process\_reviews=process\_reviews.drop(['reviewTime'], axis=1): This line is dropping the original ‘reviewTime’ column from the process\_reviews DataFrame, as it’s no longer needed.
* process\_reviews.head(): This line will display the first 5 rows of the process\_reviews DataFrame. It’s a way to quickly inspect the changes you’ve made.

After running this code, the process\_reviews DataFrame will have two new columns (‘date’ and ‘year’) and one less column (‘reviewTime’).



# Splitting the date

new1 = process\_reviews["date"].str.split(" ", n = 1, expand = True)

# adding month to the main dataset

process\_reviews["month"]= new1[0]

# adding day to the main dataset

process\_reviews["day"]= new1[1]

process\_reviews=process\_reviews.drop(['date'], axis=1)

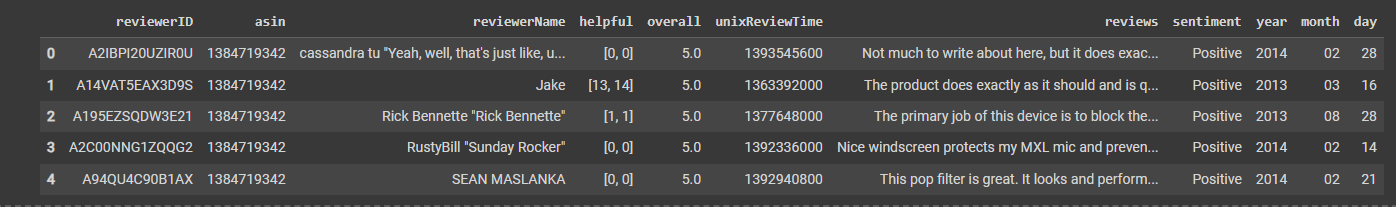
process\_reviews.head()

Explanation-

This code is further splitting the ‘date’ column into two separate columns: ‘month’ and ‘day’. Here’s what each line is doing:

* new1 = process\_reviews["date"].str.split(" ", n = 1, expand = True): This line is splitting the ‘date’ column on the space character. The str.split() function is used to split a string into a list where each word is a separate element. The expand=True argument means that the split strings should be expanded into separate columns. The result is stored in a new DataFrame called ‘new1’.
* process\_reviews["month"]= new1[0]: This line is creating a new ‘month’ column in the process\_reviews DataFrame. The values for this column are taken from the first column of the ‘new1’ DataFrame.
* process\_reviews["day"]= new1[1]: Similarly, this line is creating a new ‘day’ column in the process\_reviews DataFrame. The values for this column are taken from the second column of the ‘new1’ DataFrame.
* process\_reviews=process\_reviews.drop(['date'], axis=1): This line is dropping the original ‘date’ column from the process\_reviews DataFrame, as it’s no longer needed.
* process\_reviews.head(): This line will display the first 5 rows of the process\_reviews DataFrame. It’s a way to quickly inspect the changes you’ve made.

After running this code, the process\_reviews DataFrame will have two new columns (‘month’ and ‘day’) and one less column (‘date’).



**Finding the helpfulness of the review**

From the main dataframe we can see the helpful feature with values in list [a,b] format. It says that a out of b people found that review helpful. But with that format, it could not add value to the machine learning model and it will be difficult to decrypt the meaning for the machine. So I have planned to create helpful\_rate feature which returns a/b value from [a,b]. The following codeblock contains the complete processing step. I have added comments on what's happenening in each code. Unhide to see the code.

# Splitting the dataset based on comma and square bracket

new1 = process\_reviews["helpful"].str.split(",", n = 1, expand = True)

new2 = new1[0].str.split("[", n = 1, expand = True)

new3 = new1[1].str.split("]", n = 1, expand = True)

#Resetting the index

new2.reset\_index(drop=True, inplace=True)

new3.reset\_index(drop=True, inplace=True)

#Dropping empty columns due to splitting

new2=new2.drop([0], axis=1)

new3=new3.drop([1], axis=1)

#Concatenating the splitted columns

helpful=pd.concat([new2, new3], axis=1)

# I found few spaces in new3, so it is better to strip all the values to find the rate

def trim\_all\_columns(df):

    """

    Trim whitespace from ends of each value across all series in dataframe

    """

    trim\_strings = lambda x: x.strip() if isinstance(x, str) else x

    return df.applymap(trim\_strings)

#Applying the function

helpful= trim\_all\_columns(helpful)

#Converting into integer types

helpful[0]=helpful[0].astype(str).astype(int)

helpful[1]=helpful[1].astype(str).astype(int)

#Dividing the two columns, we have 0 in the second columns when dvided gives error, so I'm ignoring those errors

try:

  helpful['result'] = helpful[1]/helpful[0]

except ZeroDivisionError:

  helpful['result']=0

#Filling the NaN values(created due to dividing) with 0

helpful['result'] = helpful['result'].fillna(0)

#Rounding of the results to two decimal places

helpful['result']=helpful['result'].round(2)

#Attaching the results to a new column of the main dataframe

process\_reviews['helpful\_rate']=helpful['result']

#dropping the helpful column from main dataframe

process\_reviews=process\_reviews.drop(['helpful'], axis=1)

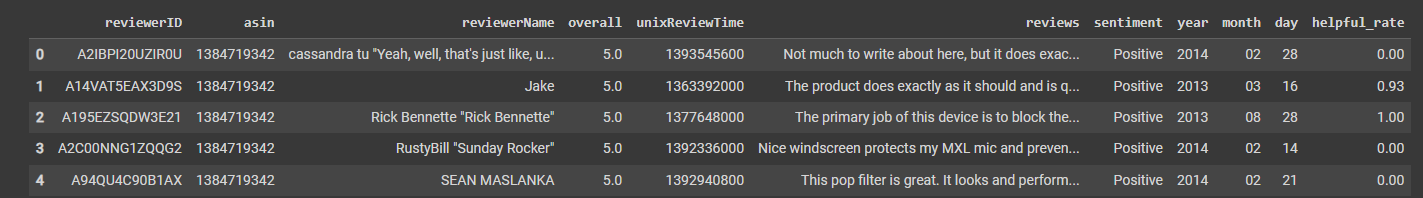
process\_reviews.head()

Explanation-

This code is processing the ‘helpful’ column of the process\_reviews DataFrame and creating a new ‘helpful\_rate’ column. Here’s what each part is doing:

1. **Splitting the ‘helpful’ column**: The ‘helpful’ column is split into two parts based on the comma and square bracket characters. This results in a new DataFrame ‘new1’ with two columns.
2. **Creating ‘new2’ and ‘new3’ DataFrames**: The first column of ‘new1’ is further split based on the ‘[’ character, resulting in the ‘new2’ DataFrame. The second column of ‘new1’ is split based on the ‘]’ character, resulting in the ‘new3’ DataFrame.
3. **Resetting the index**: The index of ‘new2’ and ‘new3’ DataFrames is reset to make concatenation easier later on.
4. **Dropping empty columns**: The first column of ‘new2’ and the second column of ‘new3’, which are empty due to the split operation, are dropped.
5. **Concatenating the ‘new2’ and ‘new3’ DataFrames**: The ‘new2’ and ‘new3’ DataFrames are concatenated along the column axis (axis=1) to form a new DataFrame ‘helpful’.
6. **Trimming all columns**: A function trim\_all\_columns(df) is defined and applied to the ‘helpful’ DataFrame to remove any leading or trailing whitespace from the values.
7. **Converting columns to integer types**: The columns of the ‘helpful’ DataFrame are converted to integer data type.
8. **Calculating the helpful rate**: A new column ‘result’ is created in the ‘helpful’ DataFrame which is the result of dividing the second column by the first column. Any ZeroDivisionError is caught and those values are set to 0.
9. **Filling NaN values and rounding**: Any NaN values in the ‘result’ column, which might have been created due to the division operation, are filled with 0. The ‘result’ values are then rounded to two decimal places.
10. **Creating the ‘helpful\_rate’ column**: The ‘result’ column from the ‘helpful’ DataFrame is added to the process\_reviews DataFrame as a new column ‘helpful\_rate’.
11. **Dropping the ‘helpful’ column**: Finally, the original ‘helpful’ column is dropped from the process\_reviews DataFrame as it’s no longer needed.

After running this code, the process\_reviews DataFrame will have a new ‘helpful\_rate’ column which is a processed version of the original ‘helpful’ column.



We have successfully created the helpful\_rate column through processing steps. Let's look at the values

process\_reviews['helpful\_rate'].value\_counts()



**Text Processing**

**Review text-Punctuation Cleaning**

Let's begin our text processing by removing the punctuations

#Removing unnecessary columns

process\_reviews=process\_reviews.drop(['reviewerName','unixReviewTime'], axis=1)

#Creating a copy

clean\_reviews=process\_reviews.copy()

def review\_cleaning(text):

    '''Make text lowercase, remove text in square brackets,remove links,remove punctuation

    and remove words containing numbers.'''

    text = str(text).lower()

    text = re.sub('\[.\*?\]', '', text)

    text = re.sub('https?://\S+|www\.\S+', '', text)

    text = re.sub('<.\*?>+', '', text)

    text = re.sub('[%s]' % re.escape(string.punctuation), '', text)

    text = re.sub('\n', '', text)

    text = re.sub('\w\*\d\w\*', '', text)

    return text

process\_reviews['reviews']=process\_reviews['reviews'].apply(lambda x:review\_cleaning(x))

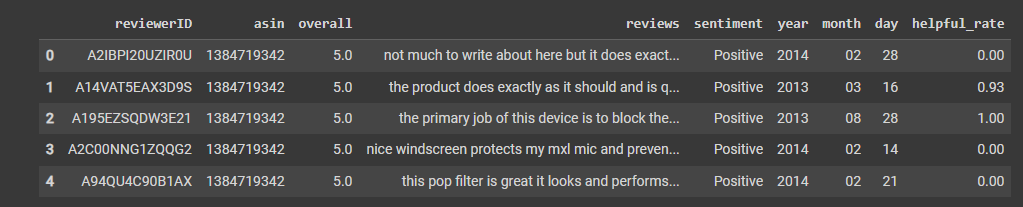
process\_reviews.head()

Explanation-

This code is performing several cleaning operations on the ‘reviews’ column of the process\_reviews DataFrame. Here’s what each part is doing:

1. **Removing unnecessary columns**: The ‘reviewerName’ and ‘unixReviewTime’ columns are dropped from the process\_reviews DataFrame as they are not needed for further analysis.
2. **Creating a copy of the DataFrame**: A copy of the process\_reviews DataFrame is created and stored in clean\_reviews. This is often done to keep an original copy of the data before making modifications.
3. **Defining the**review\_cleaning(text)**function**: This function takes a string as input and performs several cleaning operations:
   * Converts the text to lowercase.
   * Removes any text within square brackets.
   * Removes any URLs.
   * Removes any HTML tags.
   * Removes any punctuation.
   * Removes any newline characters.
   * Removes any words containing numbers.
4. **Applying the cleaning function to the ‘reviews’ column**: The review\_cleaning(text) function is applied to each entry in the ‘reviews’ column of the process\_reviews DataFrame. The cleaned reviews replace the original reviews in the ‘reviews’ column.
5. **Displaying the first 5 rows of the DataFrame**: The head() function is used to display the first 5 rows of the process\_reviews DataFrame. This allows you to quickly check the changes made to the DataFrame.

After running this code, the ‘reviews’ column of the process\_reviews DataFrame will contain cleaned versions of the reviews, and the ‘reviewerName’ and ‘unixReviewTime’ columns will be removed from the DataFrame.



**Review text-Stop Words**

Coming to stop words, general nltk stop words contains words like not, hasn’t, wouldn’t which actually convey a negative sentiment. If we remove that it will end up contradicting the target variable(sentiment). So, I have curated the stop words which doesn't have any negative sentiment or any negative alternatives**.**

stop\_words= ['yourselves', 'between', 'whom', 'itself', 'is', "she's", 'up', 'herself', 'here', 'your', 'each',

             'we', 'he', 'my', "you've", 'having', 'in', 'both', 'for', 'themselves', 'are', 'them', 'other',

             'and', 'an', 'during', 'their', 'can', 'yourself', 'she', 'until', 'so', 'these', 'ours', 'above',

             'what', 'while', 'have', 're', 'more', 'only', "needn't", 'when', 'just', 'that', 'were', "don't",

             'very', 'should', 'any', 'y', 'isn', 'who',  'a', 'they', 'to', 'too', "should've", 'has', 'before',

             'into', 'yours', "it's", 'do', 'against', 'on',  'now', 'her', 've', 'd', 'by', 'am', 'from',

             'about', 'further', "that'll", "you'd", 'you', 'as', 'how', 'been', 'the', 'or', 'doing', 'such',

             'his', 'himself', 'ourselves',  'was', 'through', 'out', 'below', 'own', 'myself', 'theirs',

             'me', 'why', 'once',  'him', 'than', 'be', 'most', "you'll", 'same', 'some', 'with', 'few', 'it',

             'at', 'after', 'its', 'which', 'there','our', 'this', 'hers', 'being', 'did', 'of', 'had', 'under',

             'over','again', 'where', 'those', 'then', "you're", 'i', 'because', 'does', 'all']

process\_reviews['reviews'] = process\_reviews['reviews'].apply(lambda x: ' '.join([word for word in x.split() if word not in (stop\_words)]))

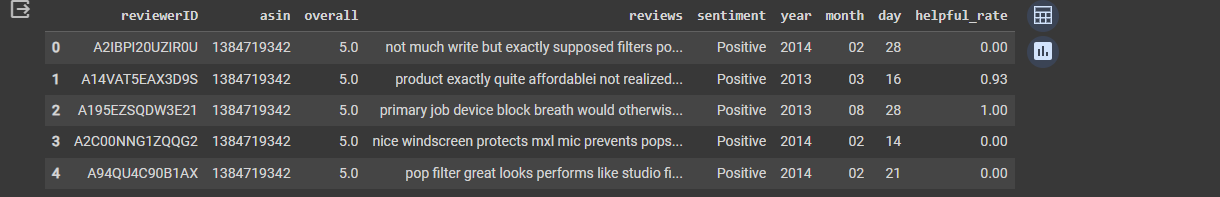
process\_reviews.head()

Explanation-

This code is removing stop words from the ‘reviews’ column of the process\_reviews DataFrame. Here’s what each part is doing:

* stop\_words= [...]: This line is defining a list of stop words. Stop words are words that you want to ignore, so you filter them out when processing text. In this case, stop words include common words like ‘is’, ‘at’, ‘which’, and ‘on’.
* process\_reviews['reviews'] = process\_reviews['reviews'].apply(lambda x: ' '.join([word for word in x.split() if word not in (stop\_words)])): This line is applying a function to the ‘reviews’ column of the process\_reviews DataFrame. This function takes a review as input, splits it into words, filters out the stop words, and then joins the words back together into a single string.

After running this code, the ‘reviews’ column of the process\_reviews DataFrame will contain the reviews with stop words removed.



**Story Generation and Visualization from reviews.**

In this section we will complete do exploratory data analysis on texts as well as other factors to understand what are all features which contributes to the sentiment.

Prior analysis assumptions:

Higher the helpful rate the sentiment becomes positive

There will be many negative sentiment reviews in the 2013 and 2014 year

There will be more reviews at the starting of a month

These assumptions will be verified with our plots also we will do text analysis a lot.

**Sentiments vs Helpful rate**

First let’s look whether there any relationship between sentiment of review and helpfulness of it.

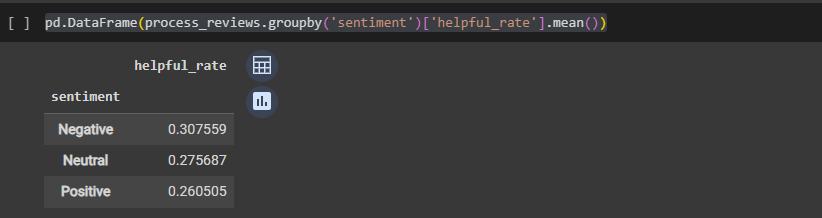
pd.DataFrame(process\_reviews.groupby('sentiment')['helpful\_rate'].mean())

Explanation-

This line of code is creating a new pandas DataFrame that contains the average ‘helpful\_rate’ for each sentiment category in the process\_reviews DataFrame. Here’s what each part is doing:

* process\_reviews.groupby('sentiment'): This groups the process\_reviews DataFrame by the ‘sentiment’ column. The result is a GroupBy object where the DataFrame is split into groups based on each unique value in the ‘sentiment’ column.
* ['helpful\_rate'].mean(): This calculates the mean (average) ‘helpful\_rate’ for each group (i.e., each sentiment category).
* pd.DataFrame(...): This converts the resulting Series object into a DataFrame.

The resulting DataFrame will have the sentiment categories as the index and a single column containing the average ‘helpful\_rate’ for each sentiment category.



From the table we can see that the mean of of helpful rate is higher for any negative reviews than neutral and positive reviews. These mean values might have been influenced by the 0 values in helpful rates. Let’s check how it is distributed through violin plot.

#plot layout

plt.rcParams.update({'font.size': 18})

rcParams['figure.figsize'] = 16,9

# Creating dataframe and removing 0 helpfulrate records

senti\_help= pd.DataFrame(process\_reviews, columns = ['sentiment', 'helpful\_rate'])

senti\_help = senti\_help[senti\_help['helpful\_rate'] != 0.00]

#Plotting phase

sns.violinplot( x=senti\_help["sentiment"], y=senti\_help["helpful\_rate"])

plt.title('Sentiment vs Helpfulness')

plt.xlabel('Sentiment categories')

plt.ylabel('helpful rate')

plt.show()

A **violin plot** is a method of plotting numeric data and is a combination of a box plot and a kernel density plot. It is used to visualize the distribution of numerical data of different variables. It is similar to a box plot, with the addition of a rotated kernel density plot on each side.

Here are the key parts of a violin plot:

* The **white dot** in the middle is the median.
* The **thick gray bar** in the center represents the interquartile range.
* The **thin gray line** that extends from it represents the 95% confidence intervals.
* The **width of the violin** at any given point represents the density or number of data points at that y-value.

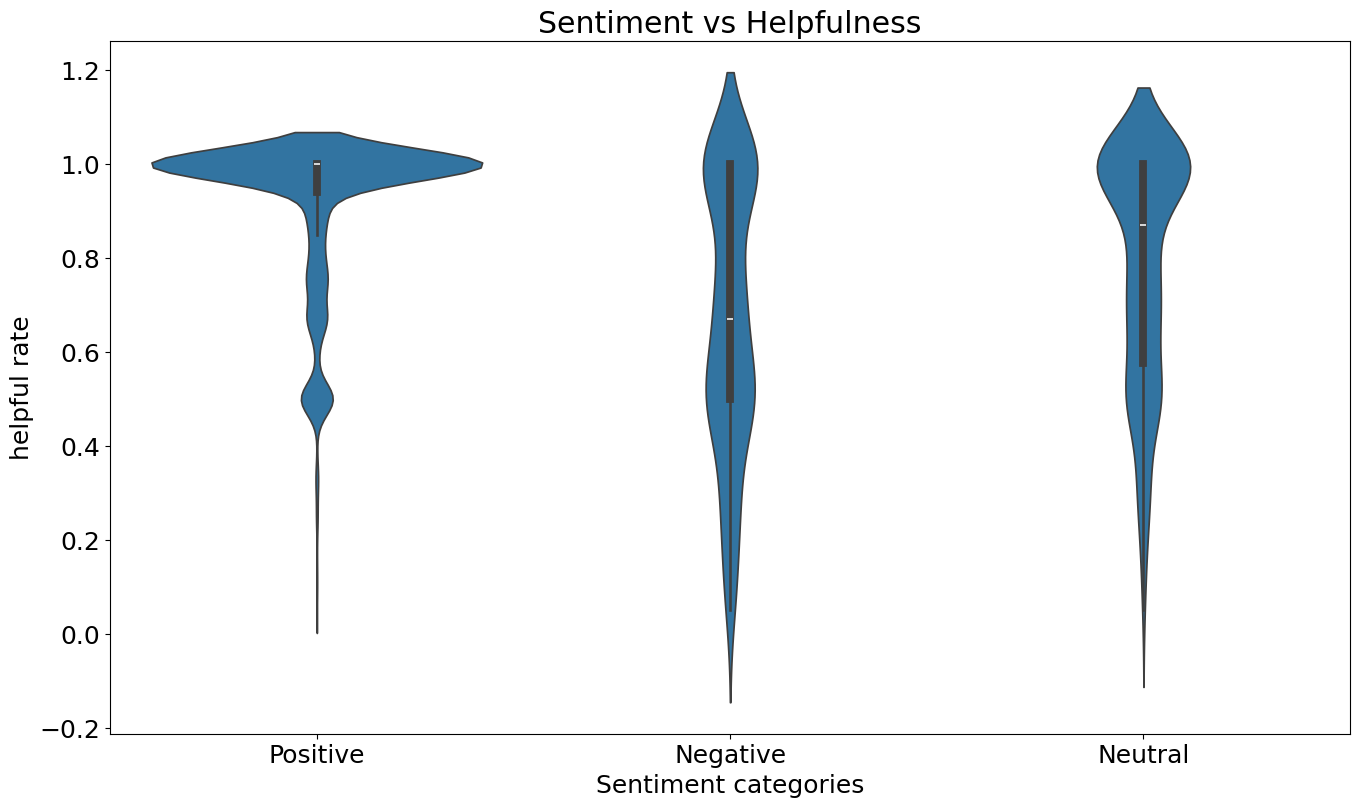
A violin plot has the advantage over a box plot in that it can show nuances in the distribution that the box plot does not. The downside is that the violin plot does not display actual data points, so it’s often good to overlay individual data points on the plot.

Explanation-

This code is creating a violin plot to visualize the distribution of ‘helpful\_rate’ across different sentiment categories. Here’s what each part is doing:

* plt.rcParams.update({'font.size': 18}) and rcParams['figure.figsize'] = 16,9: These lines are setting the global parameters for the plot. The font size is set to 18 and the figure size is set to 16x9.
* senti\_help= pd.DataFrame(process\_reviews, columns = ['sentiment', 'helpful\_rate']): This line is creating a new DataFrame ‘senti\_help’ with the ‘sentiment’ and ‘helpful\_rate’ columns from the process\_reviews DataFrame.
* senti\_help = senti\_help[senti\_help['helpful\_rate'] != 0.00]: This line is filtering out the rows in ‘senti\_help’ where ‘helpful\_rate’ is 0.00.
* sns.violinplot( x=senti\_help["sentiment"], y=senti\_help["helpful\_rate"]): This line is creating a violin plot with the ‘sentiment’ column on the x-axis and the ‘helpful\_rate’ column on the y-axis.
* plt.title('Sentiment vs Helpfulness'), plt.xlabel('Sentiment categories'), and plt.ylabel('helpful rate'): These lines are setting the title of the plot and the labels of the x and y axes.
* plt.show(): This line is displaying the plot.

After running this code, you’ll see a violin plot that shows the distribution of ‘helpful\_rate’ for each sentiment category. The width of the violins at any given y-value represents the density of data points at that ‘helpful\_rate’. This can be useful for understanding how ‘helpful\_rate’ varies with sentiment.



Insights:

From the plot we can declare that more number of positive reviews are having high helpful rate. We got deceived by the mean value, it's better to look at a plot rather than taking some measures of central tendency under such situation. Our first assumption is correct!

**Year vs Sentiment count**

In this block we will see how many reviews were posted based on sentiments in each year from 2004 to 2014

process\_reviews.groupby(['year','sentiment'])['sentiment'].count().unstack().plot(legend=True)

plt.title('Year and Sentiment count')

plt.xlabel('Year')

plt.ylabel('Sentiment count')

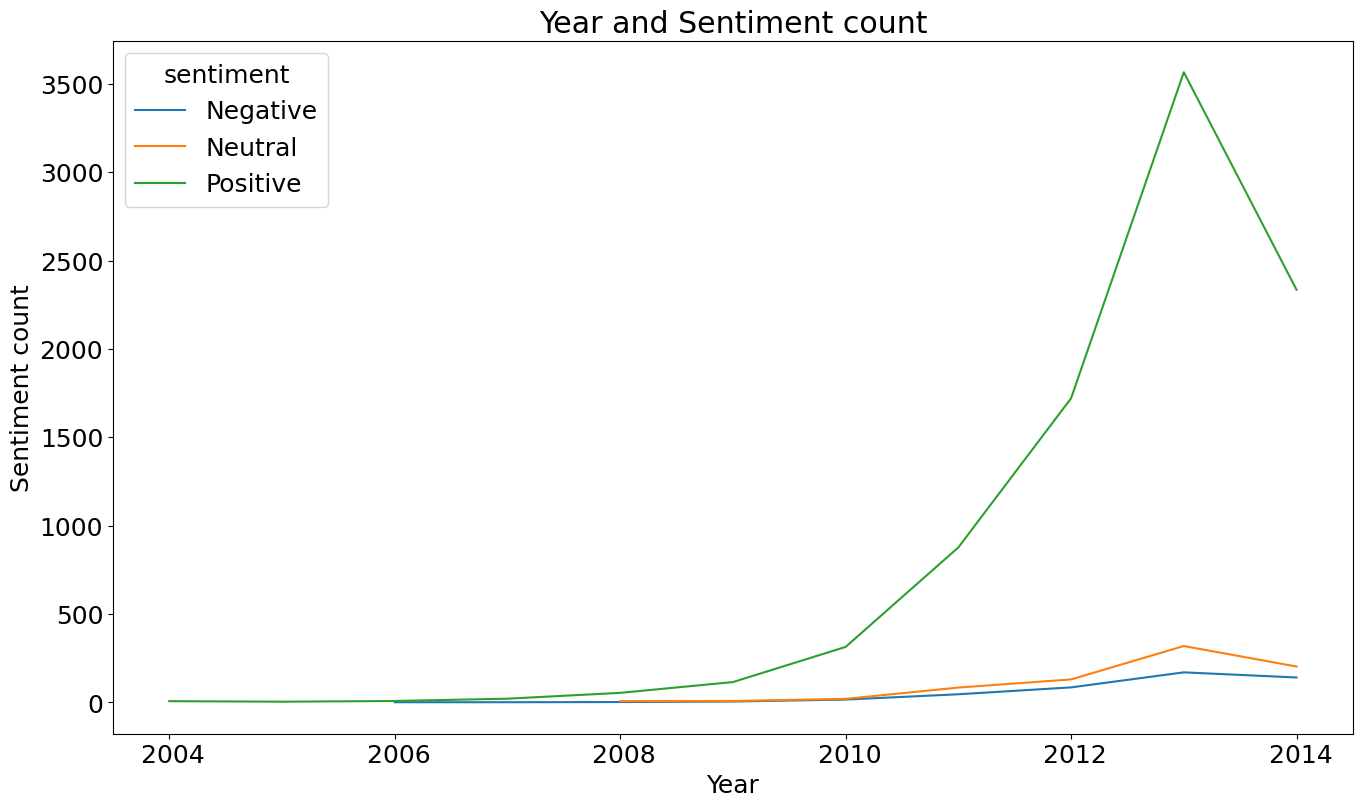
plt.show()

Explanation-

This code is creating a line plot to visualize the count of each sentiment category per year in the process\_reviews DataFrame. Here’s what each part is doing:

* process\_reviews.groupby(['year','sentiment'])['sentiment'].count(): This groups the process\_reviews DataFrame by the ‘year’ and ‘sentiment’ columns and counts the number of occurrences of each sentiment in each year.
* .unstack(): This reshapes the resulting Series (from the groupby operation) into a DataFrame where each sentiment category is a column and each row corresponds to a year.
* .plot(legend=True): This creates a line plot of the DataFrame. The legend=True argument means that a legend is added to the plot.
* plt.title('Year and Sentiment count'), plt.xlabel('Year'), and plt.ylabel('Sentiment count'): These lines are setting the title of the plot and the labels of the x and y axes.
* plt.show(): This line is displaying the plot.

After running this code, you’ll see a line plot that shows the count of each sentiment category for each year. This can be useful for understanding how the sentiment of reviews has changed over time.



Insights:

From the plot we can clearly see the rise in positive reviews from 2010. Reaching its peak around 2013 and there is a dip in 2014, All the review rates were dropped at this time. Negative and neutral reviews are very low as compared to the positive reviews. Our second assumption is wrong!

**Day of month vs Reviews count**

Let's check if there are any relationship between reviews and day of month

#Creating a dataframe

day=pd.DataFrame(process\_reviews.groupby('day')['reviews'].count()).reset\_index()

day['day']=day['day'].astype('int64')

day.sort\_values(by=['day'])

#Plotting the graph

sns.barplot(x="day", y="reviews", data=day)

plt.title('Day vs Reviews count')

plt.xlabel('Day')

plt.ylabel('Reviews count')

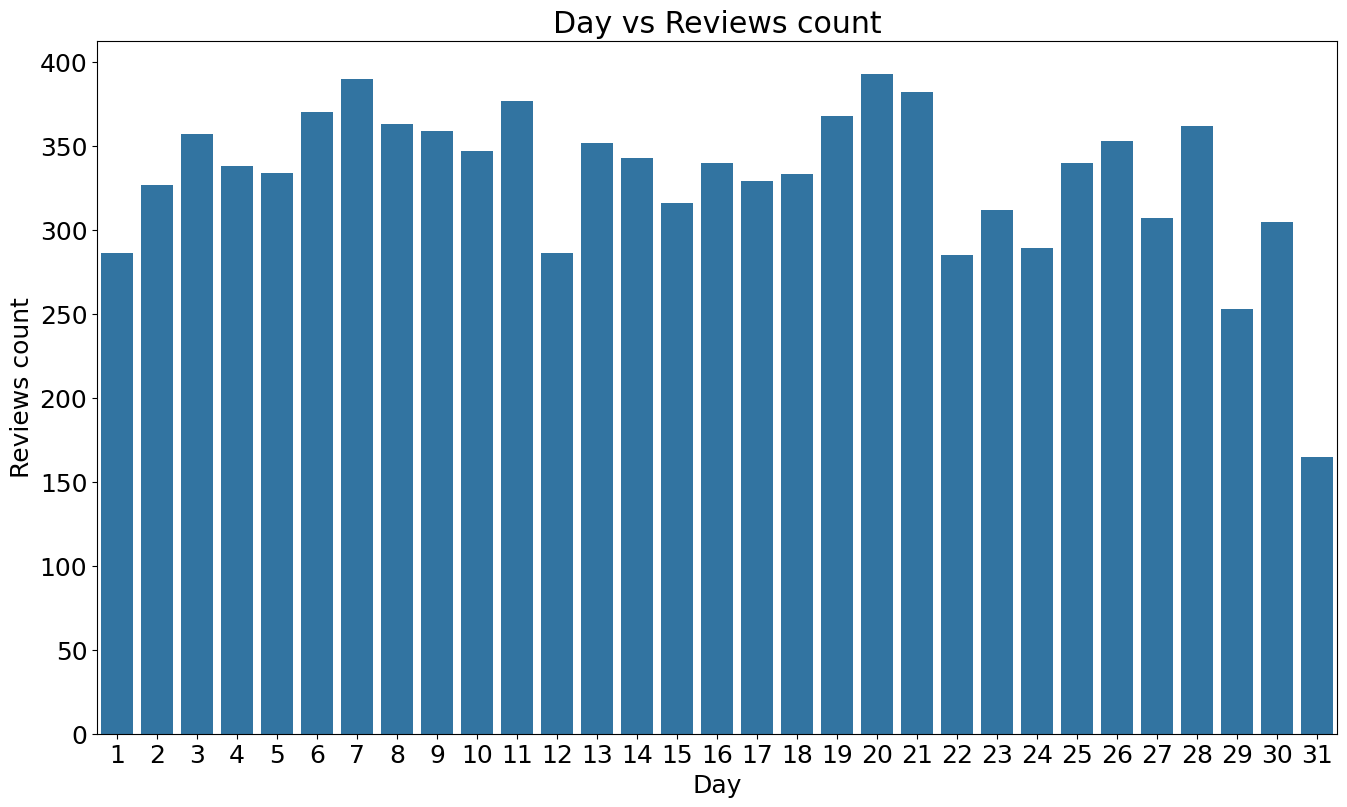
plt.show()

Explanation-

This code is creating a bar plot to visualize the count of reviews for each day of the month in the process\_reviews DataFrame. Here’s what each part is doing:

* day=pd.DataFrame(process\_reviews.groupby('day')['reviews'].count()).reset\_index(): This line is grouping the process\_reviews DataFrame by the ‘day’ column and counting the number of reviews for each day. The result is a new DataFrame ‘day’ with each row corresponding to a day of the month and a column ‘reviews’ containing the count of reviews for that day.
* day['day']=day['day'].astype('int64'): This line is converting the ‘day’ column of the ‘day’ DataFrame to integer type. This is done because the days of the month are integers.
* day.sort\_values(by=['day']): This line is sorting the ‘day’ DataFrame by the ‘day’ column. This ensures that the days are in the correct order when plotted.
* sns.barplot(x="day", y="reviews", data=day): This line is creating a bar plot with the ‘day’ column on the x-axis and the ‘reviews’ column on the y-axis.
* plt.title('Day vs Reviews count'), plt.xlabel('Day'), and plt.ylabel('Reviews count'): These lines are setting the title of the plot and the labels of the x and y axes.
* plt.show(): This line is displaying the plot.

After running this code, you’ll see a bar plot that shows the count of reviews for each day of the month. This can be useful for understanding how the number of reviews varies with the day of the month.



Insights:

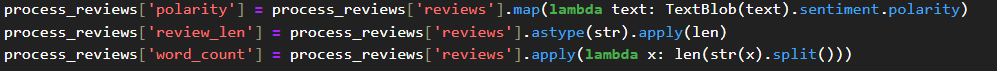
The review counts are more or less uniformly distributed.There isn't much variance between the days. But there is a huge drop at the end of month. Our third assumption is wrong ! Never trust your instincts unless you do EDA.

**Creating few more features for text analysis**

Now, let's create polarity, review length and word count

Polarity: We use Textblob for figuring out the rate of sentiment. It is between [-1,1] where -1 is negative and 1 is positive polarity Review length: length of the review which includes each letters and spaces

Word length: This measure how many words are there in review

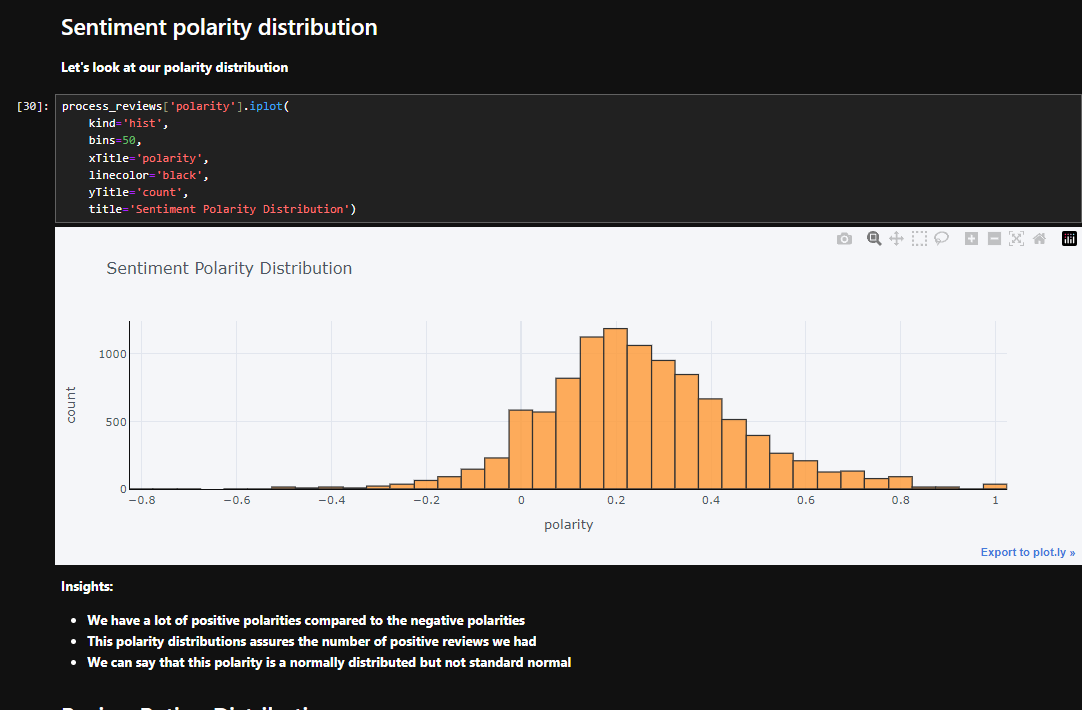


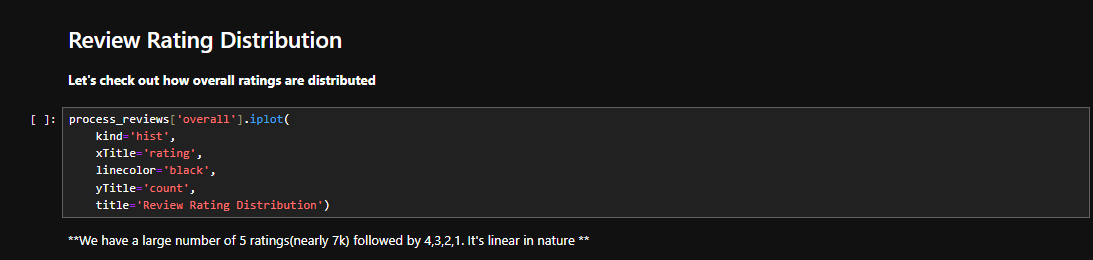


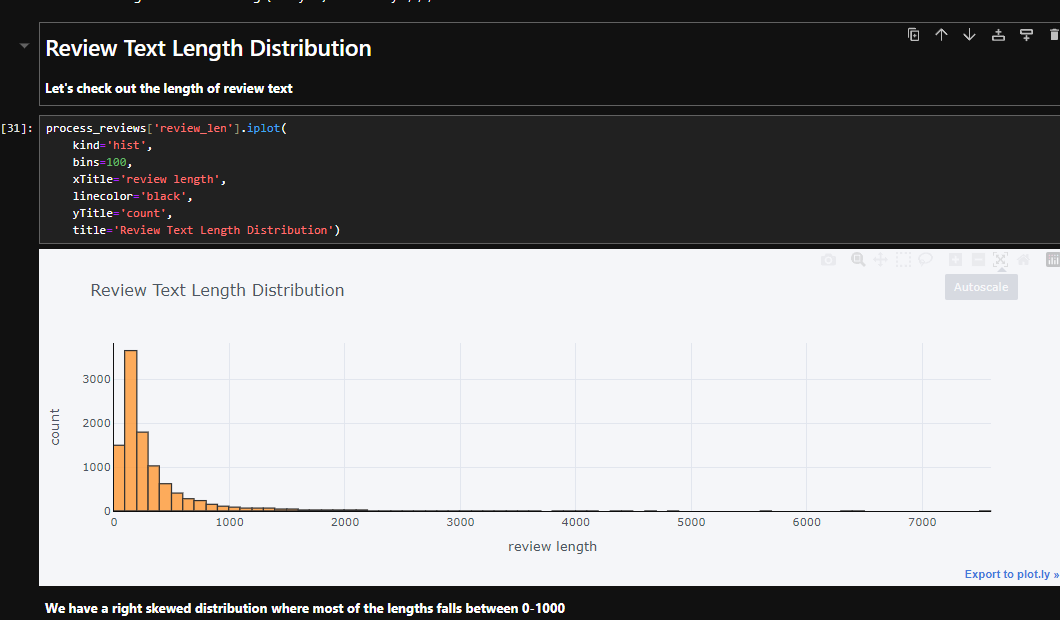


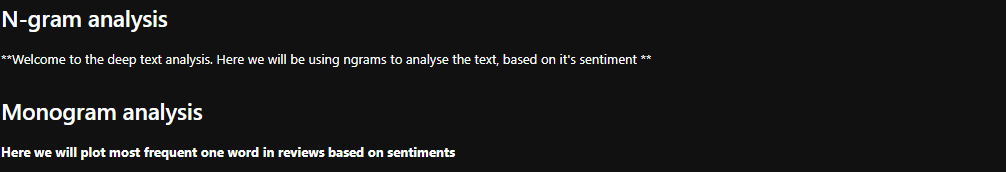
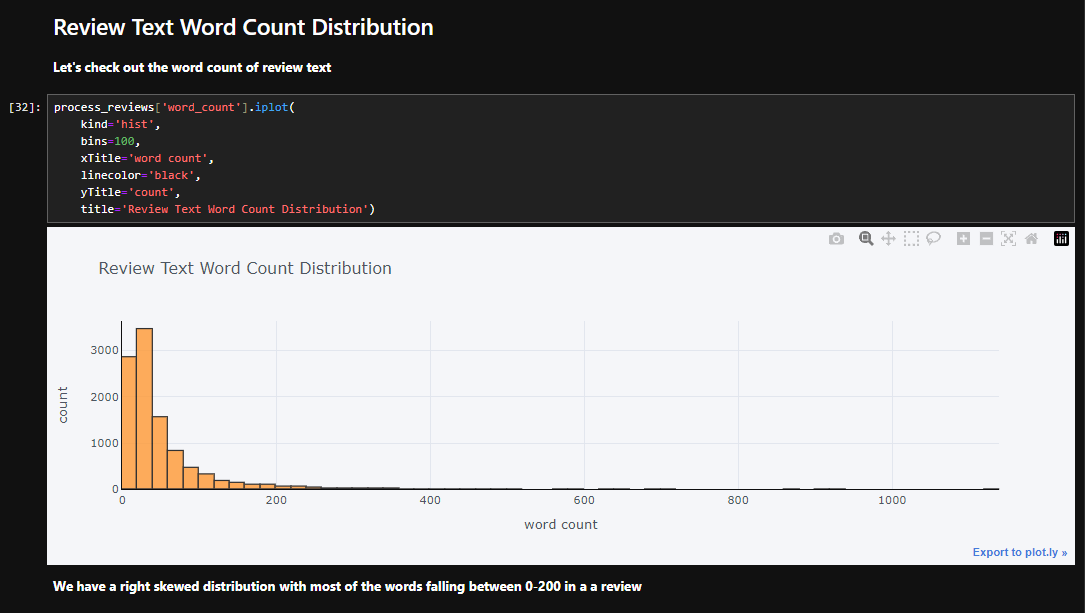


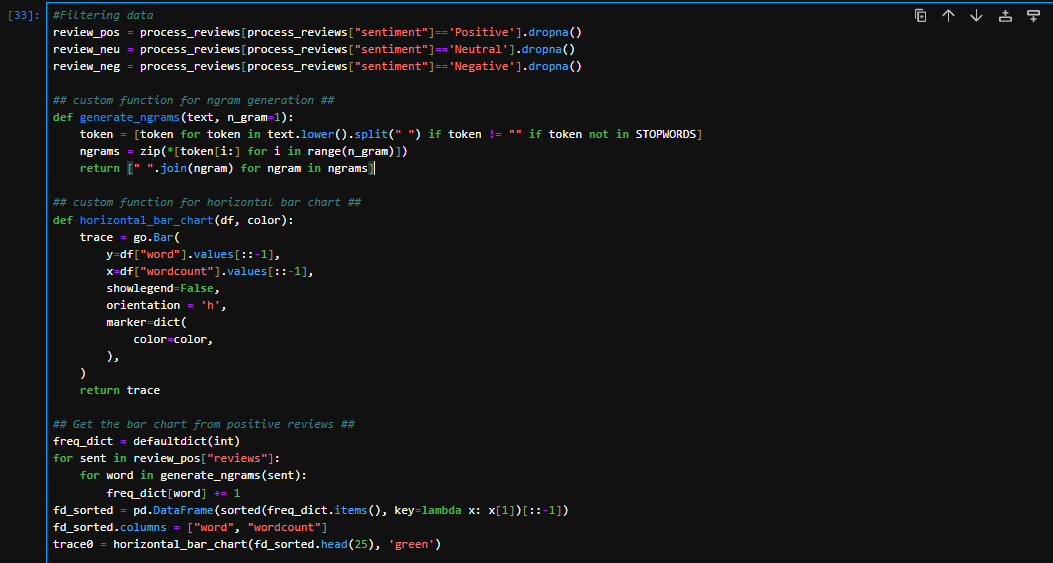
* cf.go\_offline(): This line sets up cufflinks to be used offline. This means that the plots generated by cufflinks will be rendered locally and do not require an internet connection to view.
* cf.set\_config\_file(offline=False, world\_readable=True): This line sets the configuration for cufflinks. The offline=False parameter means that cufflinks will attempt to download the latest version of plotly.js from the CDN, while world\_readable=True means that the plots generated will be public.

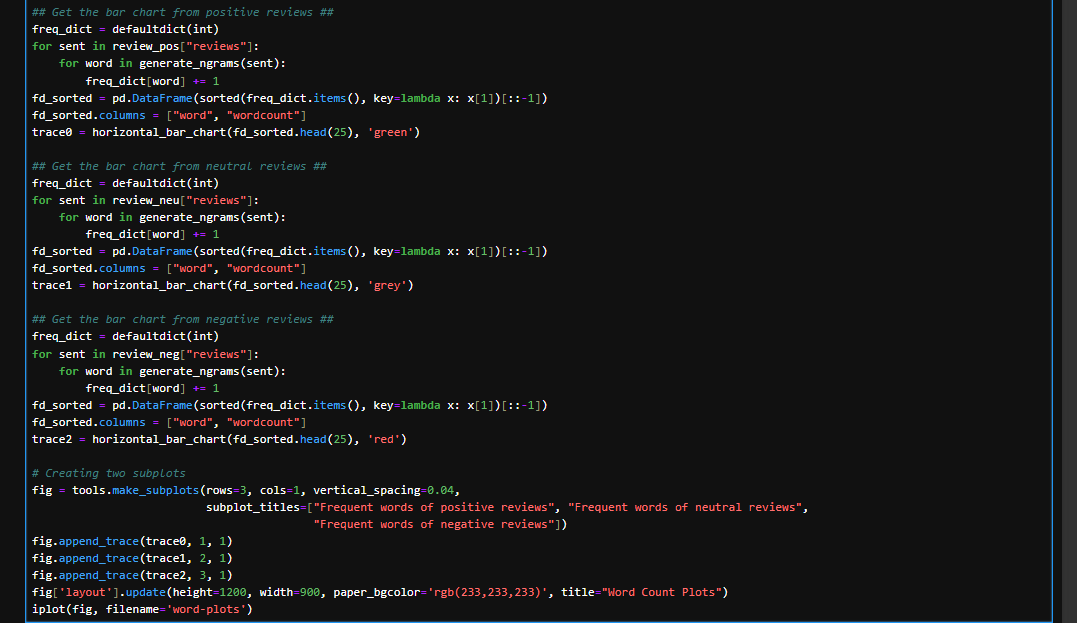










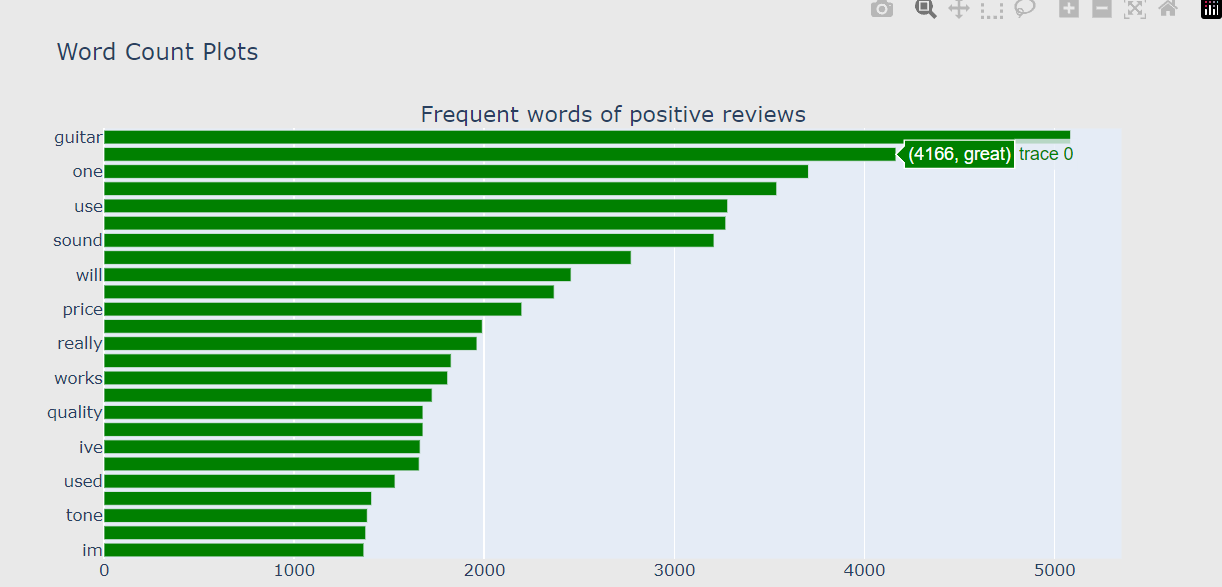


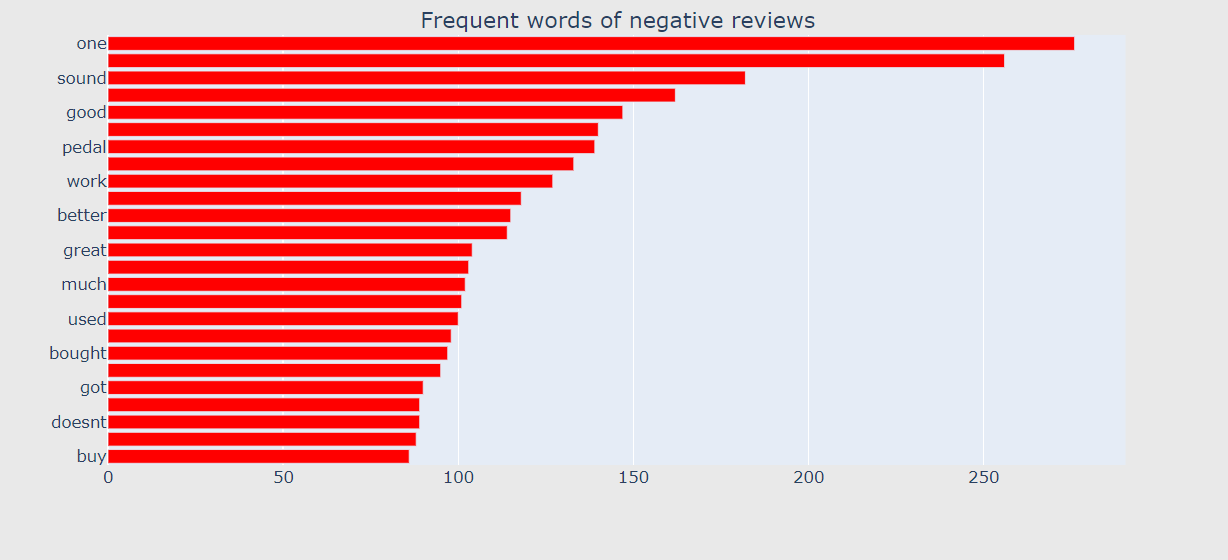
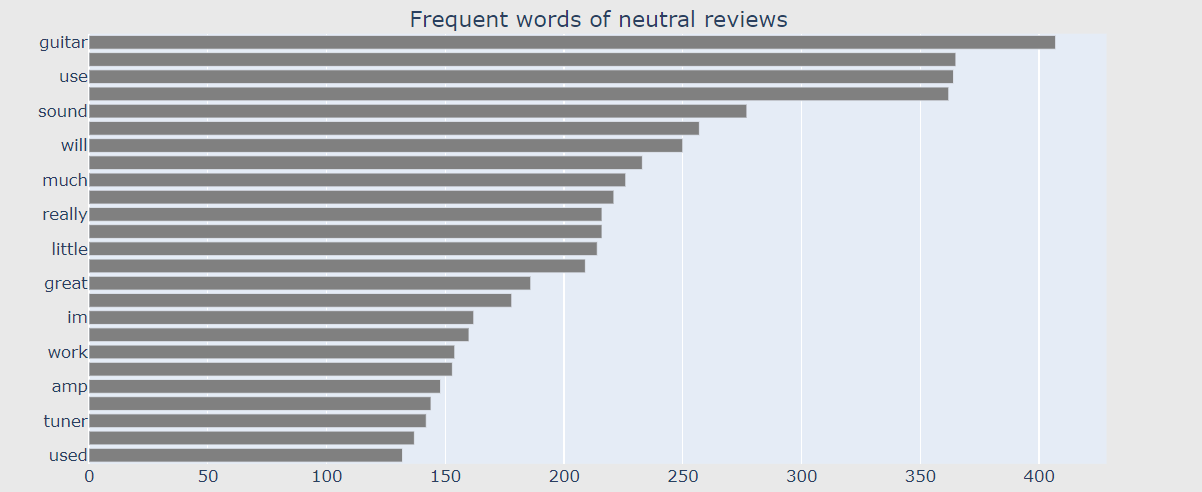
Explanation-

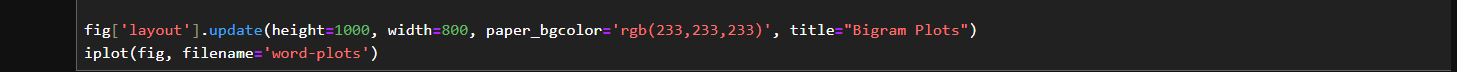
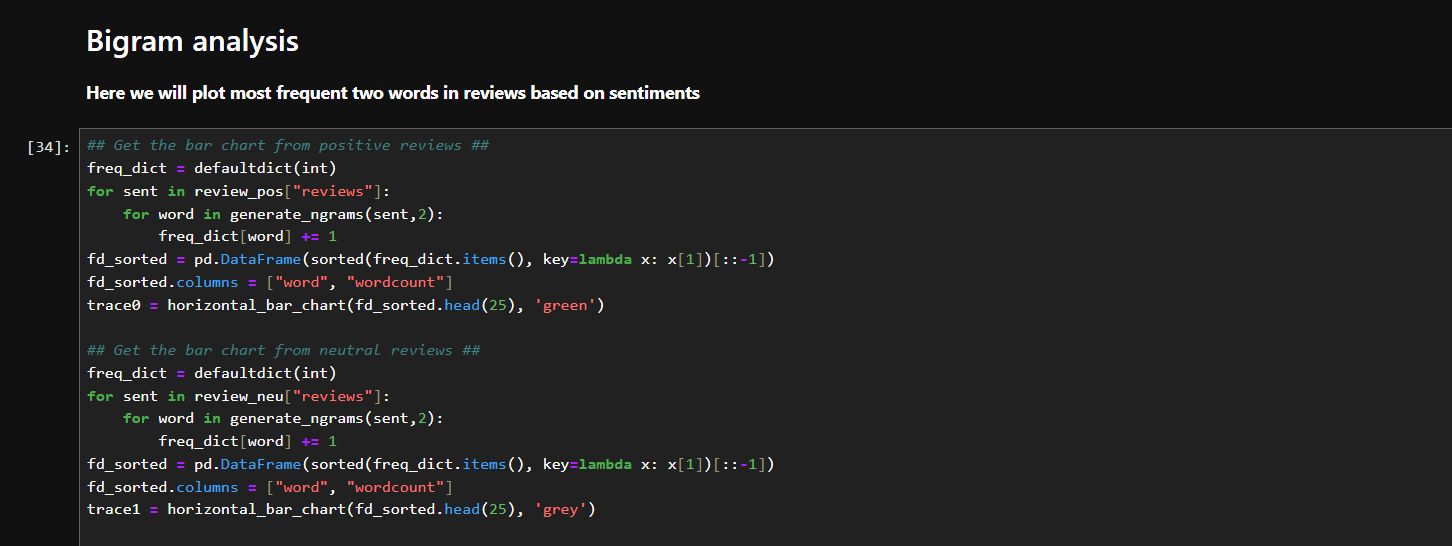
This code is creating a set of horizontal bar charts to visualize the most frequent words in positive, neutral, and negative reviews. Here’s what each part is doing:

1. **Filtering data**: The process\_reviews DataFrame is filtered based on the ‘sentiment’ column to create three new DataFrames: review\_pos, review\_neu, and review\_neg. These DataFrames contain only the positive, neutral, and negative reviews, respectively.
2. **Custom function for ngram generation**: The generate\_ngrams(text, n\_gram=1) function is defined. This function takes a string of text and an optional n\_gram parameter (default is 1), splits the text into words, and generates ngrams (contiguous sequences of n words). The function returns a list of ngrams.
3. **Custom function for horizontal bar chart**: The horizontal\_bar\_chart(df, color) function is defined. This function takes a DataFrame and a color, and returns a plotly Bar object that represents a horizontal bar chart of the data.
4. **Getting the bar chart from reviews**: For each sentiment category (positive, neutral, negative), the code generates ngrams from the reviews, counts the frequency of each ngram, sorts the ngrams by frequency, and creates a horizontal bar chart of the 25 most frequent ngrams.
5. **Creating subplots**: The tools.make\_subplots() function is used to create a subplot for each sentiment category. The bar charts created earlier are added to these subplots.
6. **Displaying the plot**: The iplot(fig, filename='word-plots') line displays the plot.

After running this code, you’ll see a set of horizontal bar charts that show the 25 most frequent words in positive, neutral, and negative reviews. This can be useful for understanding the most common words used in each type of review.





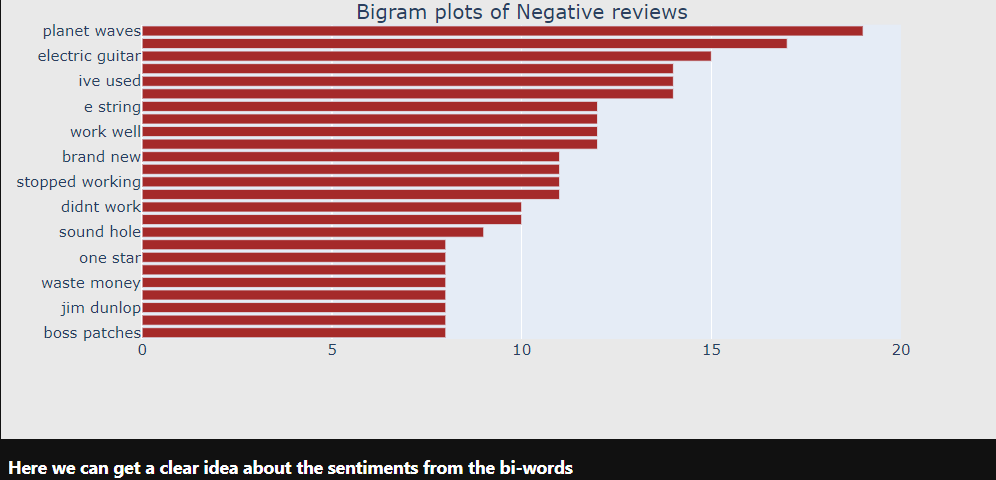
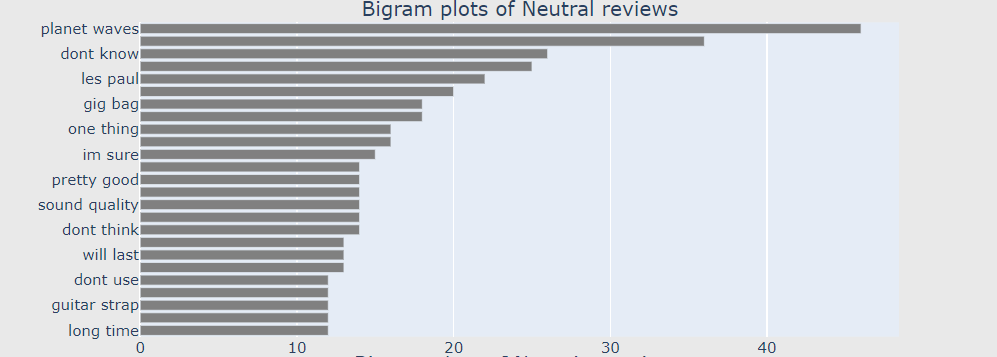
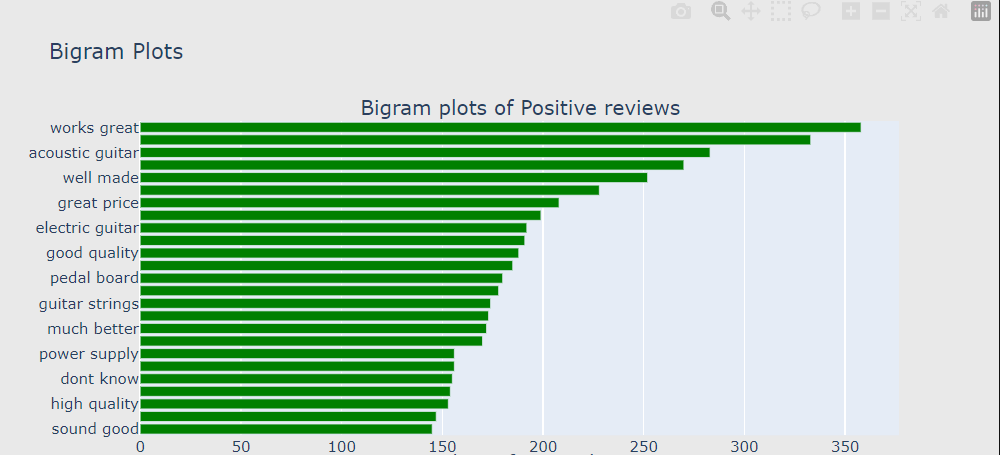


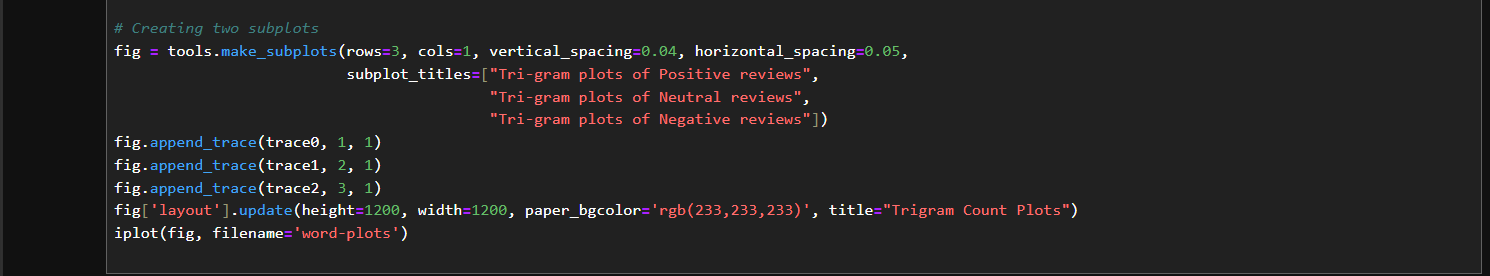
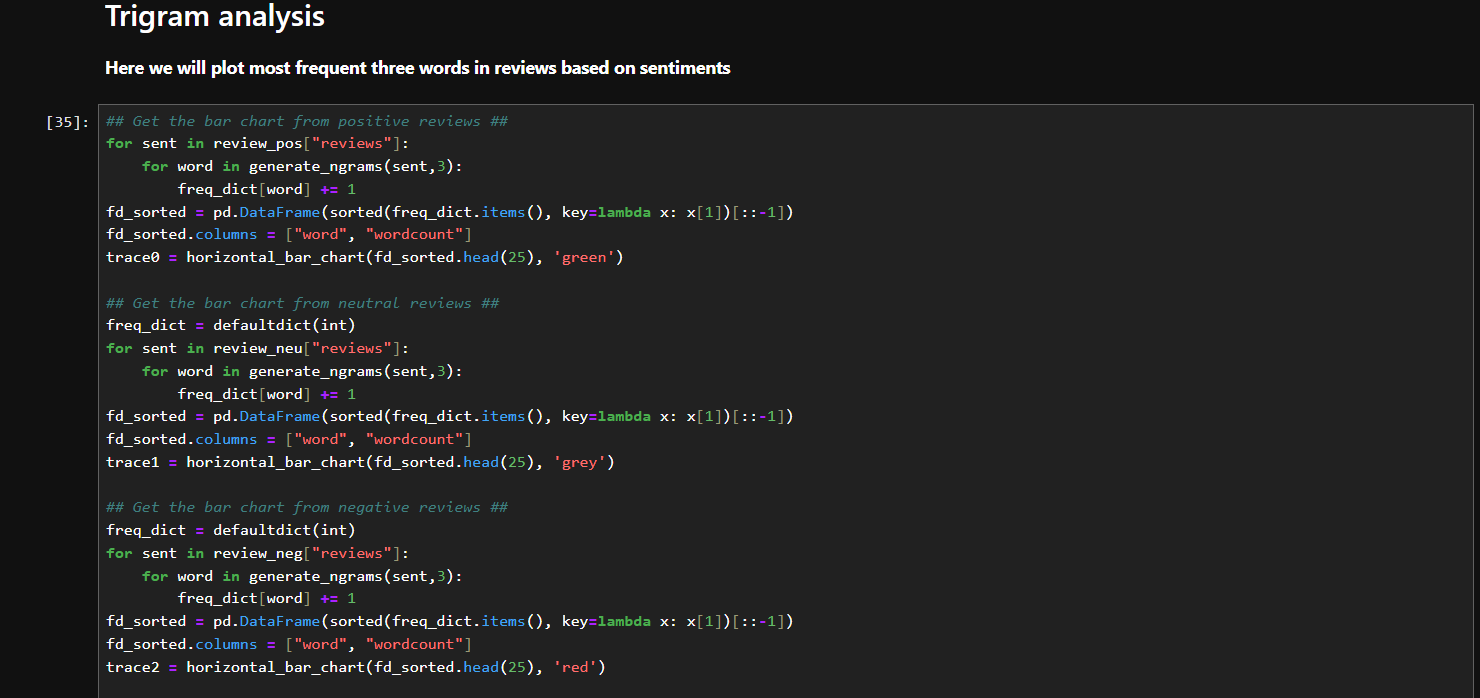
Explanation-

This code is creating a set of horizontal bar charts to visualize the most frequent bigrams (two-word phrases) in positive, neutral, and negative reviews. Here’s what each part is doing:

1. **Generating bigrams and counting their frequencies**: For each sentiment category (positive, neutral, negative), the code generates bigrams from the reviews, counts the frequency of each bigram, and sorts the bigrams by frequency. This is done using the generate\_ngrams(sent,2) function, which generates bigrams when the second argument is 2.
2. **Creating horizontal bar charts**: For each sentiment category, a horizontal bar chart of the 25 most frequent bigrams is created using the horizontal\_bar\_chart() function.
3. **Creating subplots**: The tools.make\_subplots() function is used to create a subplot for each sentiment category. The bar charts created earlier are added to these subplots.
4. **Displaying the plot**: The iplot(fig, filename='word-plots') line displays the plot.

After running this code, you’ll see a set of horizontal bar charts that show the 25 most frequent bigrams in positive, neutral, and negative reviews. This can be useful for understanding the most common phrases used in each type of review.



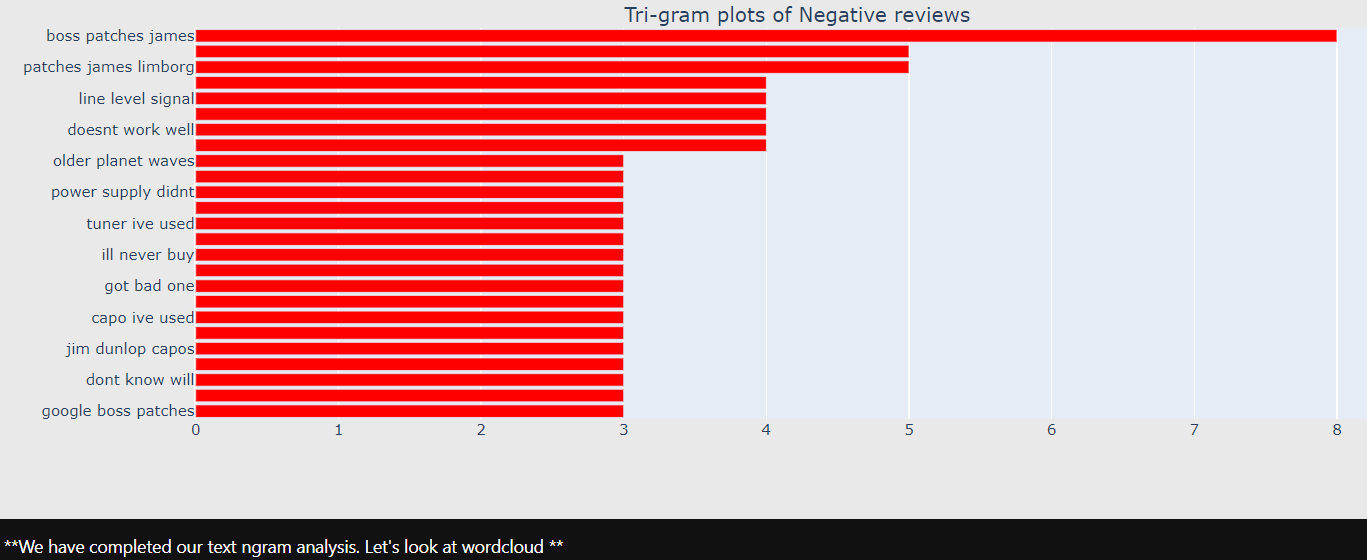
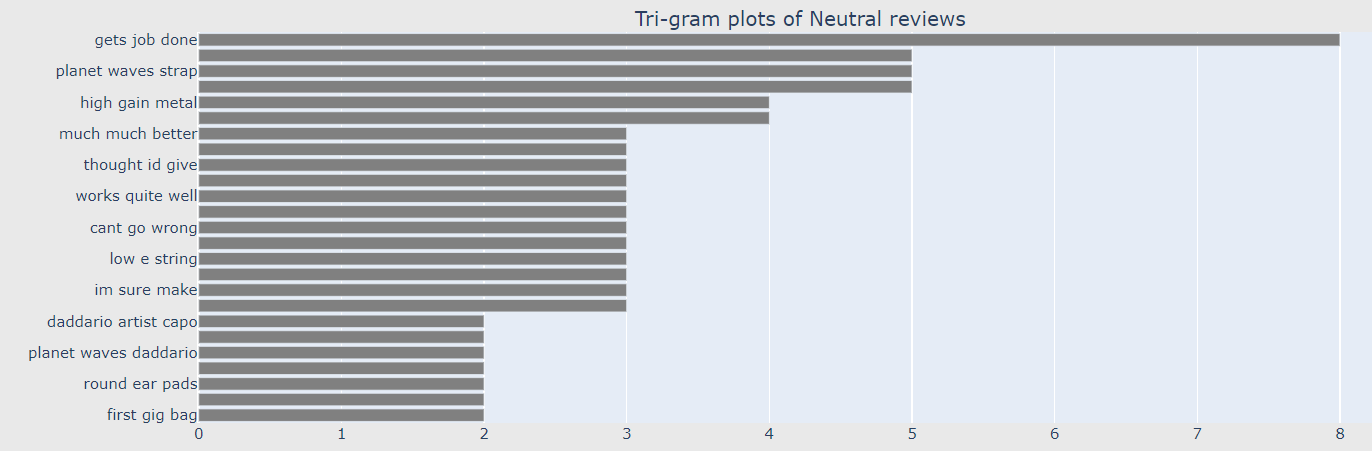
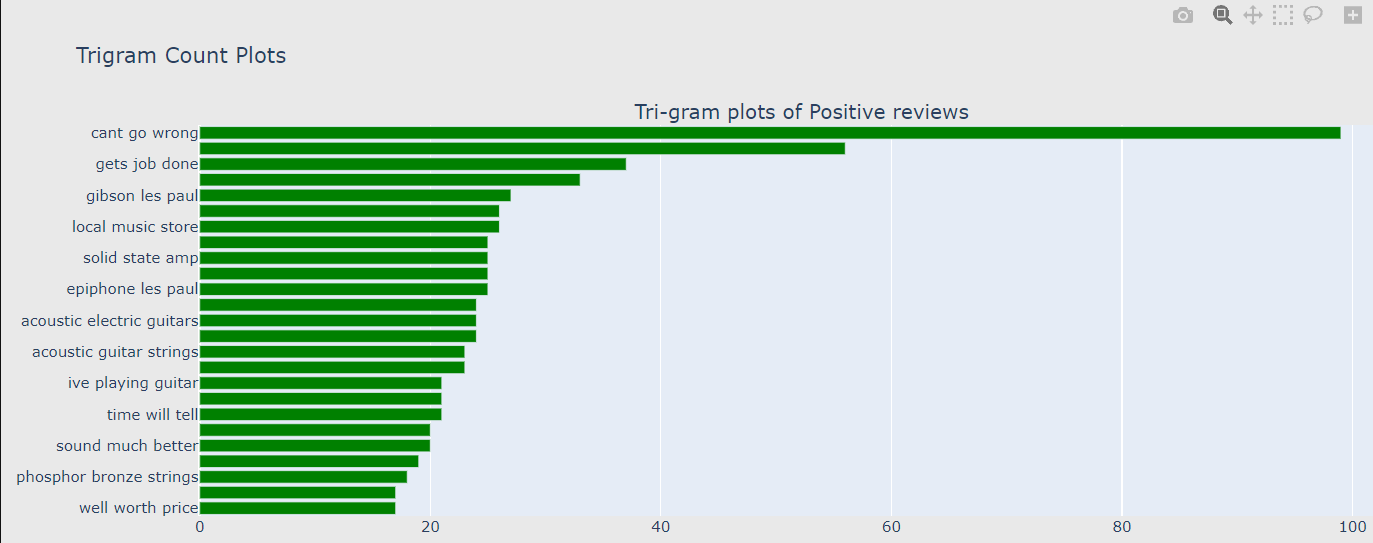


Explanation-

This code is creating a set of horizontal bar charts to visualize the most frequent trigrams (three-word phrases) in positive, neutral, and negative reviews. Here’s what each part is doing:

1. **Generating trigrams and counting their frequencies**: For each sentiment category (positive, neutral, negative), the code generates trigrams from the reviews, counts the frequency of each trigram, and sorts the trigrams by frequency. This is done using the generate\_ngrams(sent,3) function, which generates trigrams when the second argument is 3.
2. **Creating horizontal bar charts**: For each sentiment category, a horizontal bar chart of the 25 most frequent trigrams is created using the horizontal\_bar\_chart() function.
3. **Creating subplots**: The tools.make\_subplots() function is used to create a subplot for each sentiment category. The bar charts created earlier are added to these subplots.
4. **Displaying the plot**: The iplot(fig, filename='word-plots') line displays the plot.

After running this code, you’ll see a set of horizontal bar charts that show the 25 most frequent trigrams in positive, neutral, and negative reviews. This can be useful for understanding the most common phrases used in each type of review.



Rest in Jupyter File