Sentiment Analysis

Khalil Kitar

Mouad aboussad

This notebook features a Sentiment Analysis Mini Project focused on reviews of Amazon Musical Instruments. My interest in Natural Language Processing (NLP) is the main motivation behind creating this project. I believe sentiment analysis plays a crucial role in business development as it provides valuable insights that can drive strategic decisions.

Libraries

Data Libraries

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

NLP Text Libraries

```
import string
import re
import nltk
import nltk.corpus
nltk.download("punkt")
nltk.download("stopwords")
nltk.download("wordnet")
from nltk.stem import WordNetLemmatizer
```

EDA Analysis

```
# Text Polarity
from textblob import TextBlob

# Text Vectorizer
from sklearn.feature_extraction.text import CountVectorizer

# Word Cloud
from wordcloud import WordCloud
```

Feature Engineering

```
# Label Encoding
from sklearn.preprocessing import LabelEncoder
```

```
# TF-IDF Vectorizer
from sklearn.feature_extraction.text import TfidfVectorizer

# Resampling
from imblearn.over_sampling import SMOTE
from collections import Counter

# Splitting Dataset
from sklearn.model_selection import train_test_split
```

Model Selection and Evaluation

```
# Model Building
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import BernoulliNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score

# Hyperparameter Tuning
from sklearn.model_selection import GridSearchCV

# Model Metrics
from sklearn.metrics import confusion_matrix, accuracy_score,
classification_report
```

The Dataset

The dataset for this project is sourced from the Kaggle website and can be accessed under the title "Amazon Musical Instruments Reviews." It is available in two formats: JSON and CSV. For this project, we will be using the CSV format. The dataset provides customer feedback on musical instruments purchased from Amazon.

Read The Dataset

```
dataset = pd.read_csv("Instruments_Reviews.csv")
```

Shape of The Dataset

```
dataset.shape
```

```
(10261, 9)
```

From this, we can infer that the dataset consists of 10261 rows and 9 columns.

Data Preprocessing

Checking Null Values

```
dataset.isnull().sum()
```

```
reviewerID          0
asin                0
reviewerName       27
helpful             0
reviewText          7
overall             0
summary             0
unixReviewTime      0
reviewTime          0
dtype: int64
```

As noted above, the dataset contains two columns with null values: `reviewText` and `reviewerName`. While the `reviewerName` column is not particularly significant, we need to pay attention to the `reviewText` column. Removing rows with missing `reviewText` is not ideal since the associated ratings and summaries could impact our model's performance, even though the number of missing rows is minimal. Therefore, we can address this by filling the empty `reviewText` values with an empty string.

Filling Missing Values

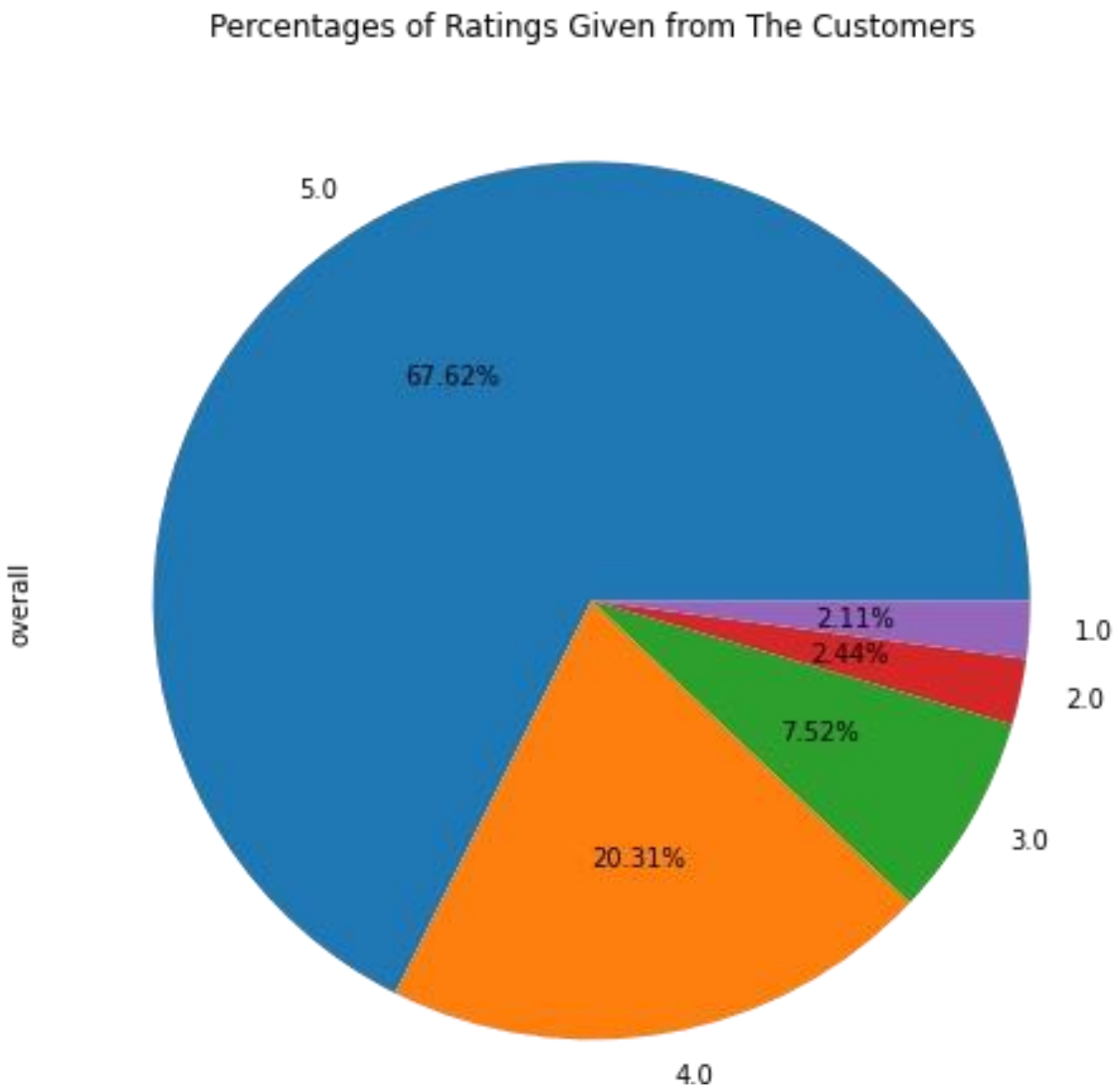
```
dataset.reviewText.fillna(value = "", inplace = True)
```

Concatenate `reviewText` and `summary` Columns

```
dataset["reviews"] = dataset["reviewText"] + " " + dataset["summary"]
dataset.drop(columns = ["reviewText", "summary"], axis = 1, inplace =
True)
```

Percentages of Ratings Given from The Customers

```
dataset.overall.value_counts().plot(kind = "pie", legend = False,
autopct = "%1.2f%%", fontsize = 10, figsize=(8,8))
plt.title("Percentages of Ratings Given from The Customers", loc =
"center")
plt.show()
```

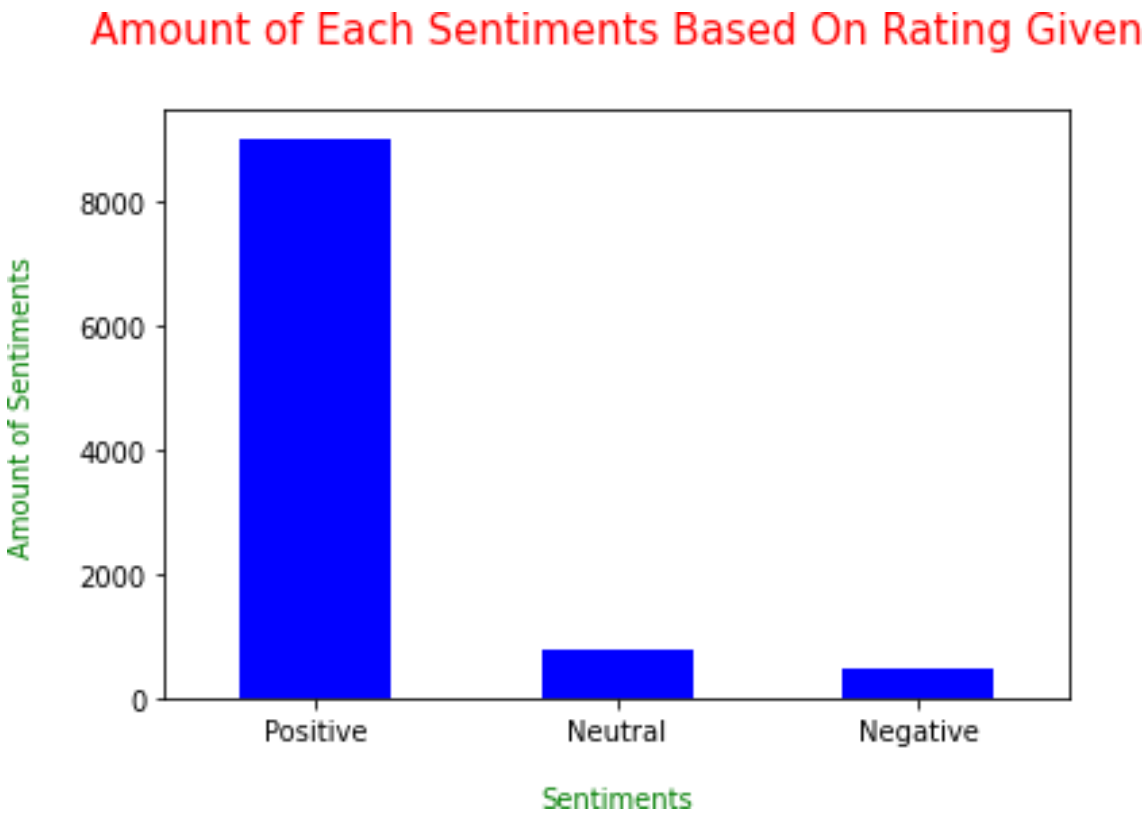


Based on the chart above, most musical instruments sold on Amazon receive perfect ratings of 5.0, indicating the products are generally in good condition. If we categorize ratings above 3 as positive, 3 as neutral, and below 3 as negative, we can see that the dataset contains relatively few negative reviews. This imbalance may impact the model's performance in later stages.

Labelling Products Based On Ratings Given

Our dataset currently lacks a dependent variable, meaning we do not yet have a prediction target. To create this, we will classify sentiments based on the ratings in each row according to the earlier outlined criteria: assigning a Positive label to products with ratings greater than 3.0, a Neutral label to products with ratings equal to 3.0, and a Negative label to products with ratings below 3.0.

```
def Labelling(Rows):
  if(Rows["overall"]>3.0):
    Label = "Positive"
  elif(Rows["overall"]<3.0):
    Label="Negative"
  else:
    Label="Neutral"
  return Label

dataset["sentiment"] = dataset.apply(Labelling, axis = 1)

dataset["sentiment"].value_counts().plot(kind = "bar",color= "blue")

plt.title("Amount of Each Sentiments Based On Rating Given", loc =
"center", fontsize = 15, color = "red", pad = 25)
plt.xlabel("Sentiments",color= "green",fontsize = 10,labelpad = 15)
plt.xticks(rotation=0)
plt.ylabel("AmountofSentiments",color= "green",fontsize = 10, labelpad
= 15)
plt.show()
```



At this stage, we could convert the labels into numeric values, but we will postpone this step for the sake of future experiments. Additionally, it is evident from the graph that most of the data reflects positive sentiments, which aligns with the earlier analysis during data exploration.

Text Preprocessing

Text Cleaning

```
def Text_Cleaning(Text):
  # Lowercase the texts
  Text = Text.lower()

  # Cleaning punctuations in the text
  punc = str.maketrans(string.punctuation, '
'*len(string.punctuation))
  Text = Text.translate(punc)

  # Removing numbers in the text
  Text = re.sub(r'\d+', '', Text)

  # Remove possible links
  Text = re.sub('https?://\S+|www\.\S+', '', Text)

  # Deleting newlines
  Text = re.sub('\n', '', Text)

  return Text
```

Text Processing

```
# Stopwords
Stopwords = set(nltk.corpus.stopwords.words("english")) - set(["not"])


def Text_Processing(Text):
  Processed_Text = list()
  Lemmatizer = WordNetLemmatizer()

  # Tokens of Words
  Tokens = nltk.word_tokenize(Text)

  # Removing Stopwords and Lemmatizing Words
  # To reduce noises in our dataset, also to keep it simple and still

  # powerful, we will only omit the word `not` from the list of
stopwords

  for word in Tokens:
    if word not in Stopwords:
      Processed_Text.append(Lemmatizer.lemmatize(word))
```

```
  return(" ".join(Processed_Text))
```

Applying The Functions

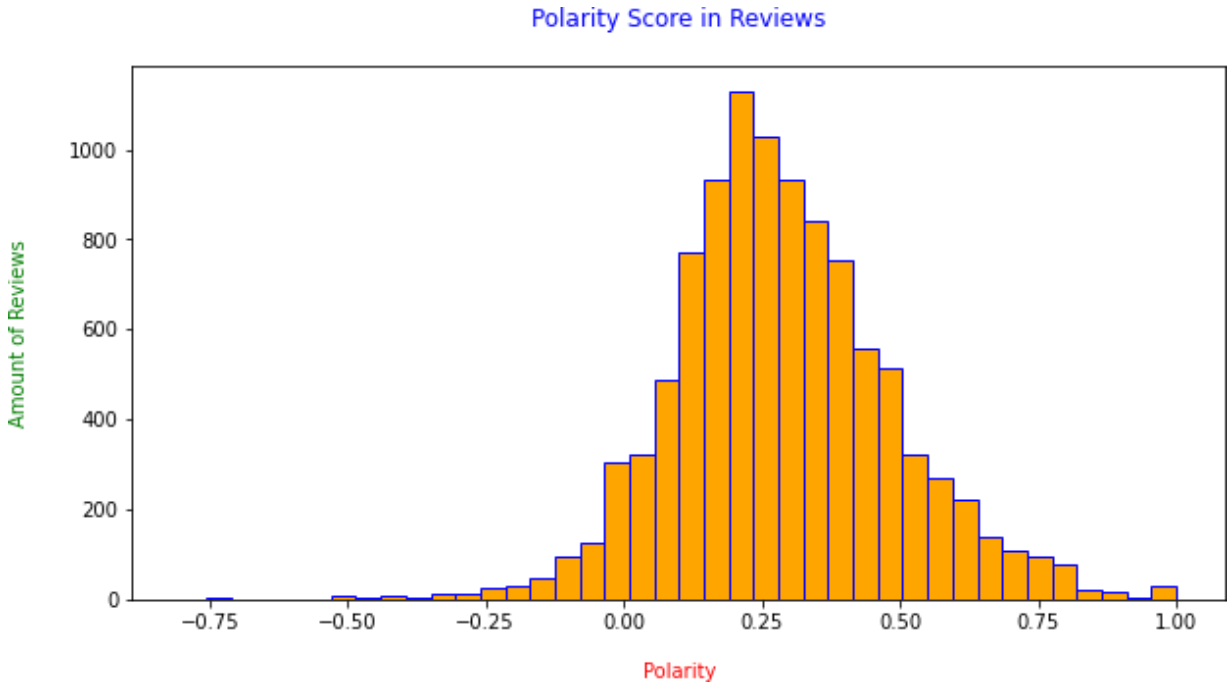
```
dataset["reviews"] = dataset["reviews"].apply(lambda Text:
Text_Cleaning(Text))
dataset["reviews"] = dataset["reviews"].apply(lambda Text:
Text_Processing(Text))
```

Exploratory Data Analysis

Polarity, Review Length, and Word Counts

To justify our analysis before, we will dive further into the dataset a bit more from the polarity of the texts, also from the words used in the reviews. We will generate some new columns in our dataset and visualize it.

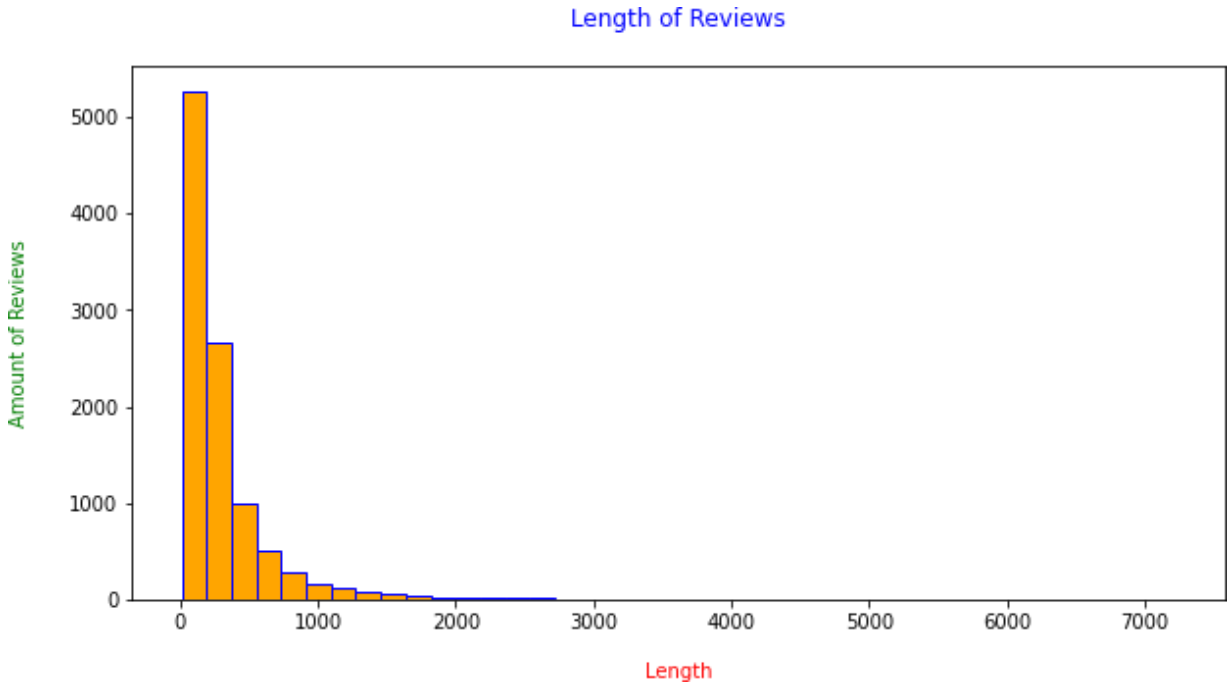
```
dataset["polarity"]=dataset["reviews"].map(lambdaText:
TextBlob(Text).sentiment.polarity)

dataset["polarity"].plot(kind = "hist",bins = 40,edgecolor = "blue",
linewidth = 1, color = "orange", figsize = (10,5))
plt.title("PolarityScoreinReviews",color= "blue",pad= 20)
plt.xlabel("Polarity", labelpad = 15, color = "red")
plt.ylabel("Amount of Reviews", labelpad = 20, color = "green")

plt.show()
```



Reviews with negative polarity are represented in the range of [-1, 0), neutral reviews are at 0.0, and positive reviews fall within the range of (0, 1]. The histogram above confirms that the majority of reviews are skewed towards positive sentiments, supporting our previous analysis. Statistically, this histogram indicates that the data follows a normal distribution pattern, though not a standard distribution. In conclusion, our analysis regarding the distribution of sentiments in the reviews is accurate and aligns with the findings shown in the histogram.

Review Length

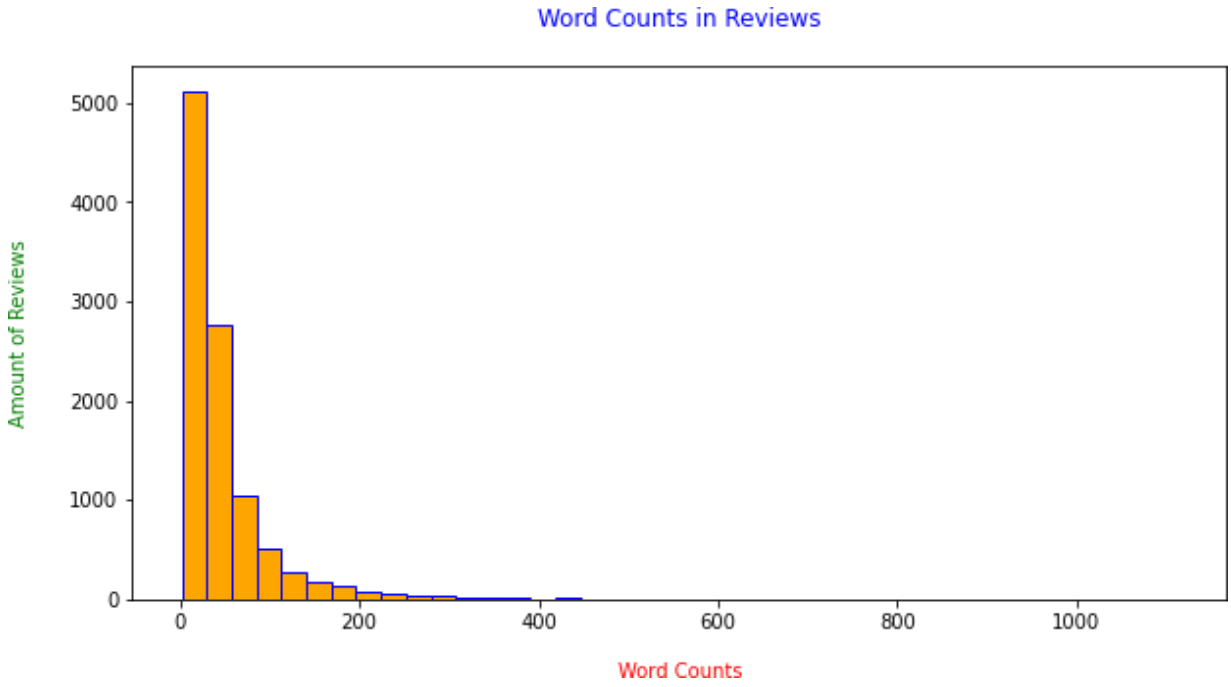
```
dataset["length"] = dataset["reviews"].astype(str).apply(len)

dataset["length"].plot(kind = "hist", bins = 40, edgecolor = "blue",
linewidth = 1, color = "orange", figsize = (10,5))
plt.title("Length of Reviews", color = "blue", pad = 20)
plt.xlabel("Length", labelpad = 15, color = "red")
plt.ylabel("Amount of Reviews", labelpad = 20, color = "green")

plt.show()
```



From this, we can infer that the text length of our reviews ranges approximately between 0 to 1,000 characters. The distribution shows positive skewness, or right-skewed behavior, indicating that reviews rarely exceed 1,000 characters in length. It is important to note that this review length is influenced by the text preprocessing phase, which may remove certain words, so the length does not represent the original text in its entirety. This will similarly impact word count measurements for the reviews.

Word Counts

```
dataset["word_counts"] = dataset["reviews"].apply(lambda x:
len(str(x).split()))

dataset["word_counts"].plot(kind = "hist", bins = 40, edgecolor =
"blue", linewidth = 1, color = "orange", figsize = (10,5))
plt.title("Word Counts in Reviews", color = "blue", pad = 20)
plt.xlabel("Word Counts", labelpad = 15, color = "red")
plt.ylabel("Amount of Reviews", labelpad = 20, color = "green")

plt.show()
```



From the figure above, we can deduce that most reviews contain between 0-200 words. Similar to the character length analysis, the distribution is right-skewed. Additionally, this word count is influenced by the prior text preprocessing phase, which may have altered the original word count by removing certain words.

N-Gram Analysis

N-Gram Function

```
def Gram_Analysis(Corpus, Gram, N):
  # Vectorizer
  Vectorizer = CountVectorizer(stop_words = Stopwords,
ngram_range=(Gram,Gram))

  # N-Grams Matrix
  ngrams = Vectorizer.fit_transform(Corpus)

  # N-Grams Frequency
  Count = ngrams.sum(axis=0)

  # List of Words
  words = [(word, Count[0, idx]) for word, idx in
Vectorizer.vocabulary_.items()]

  # Sort Descending With Key = Count
  words = sorted(words, key = lambda x:x[1], reverse = True)
```

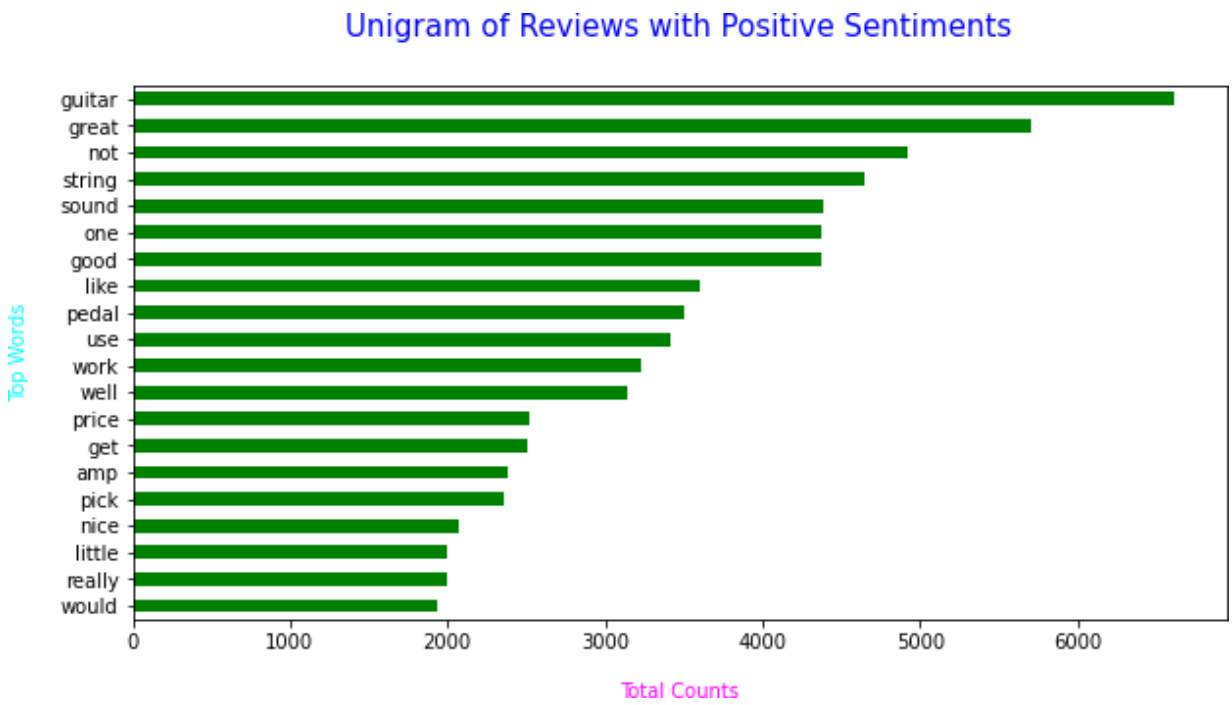
```
    return words[:N]
```

Filter The DataFrame Based On Sentiments

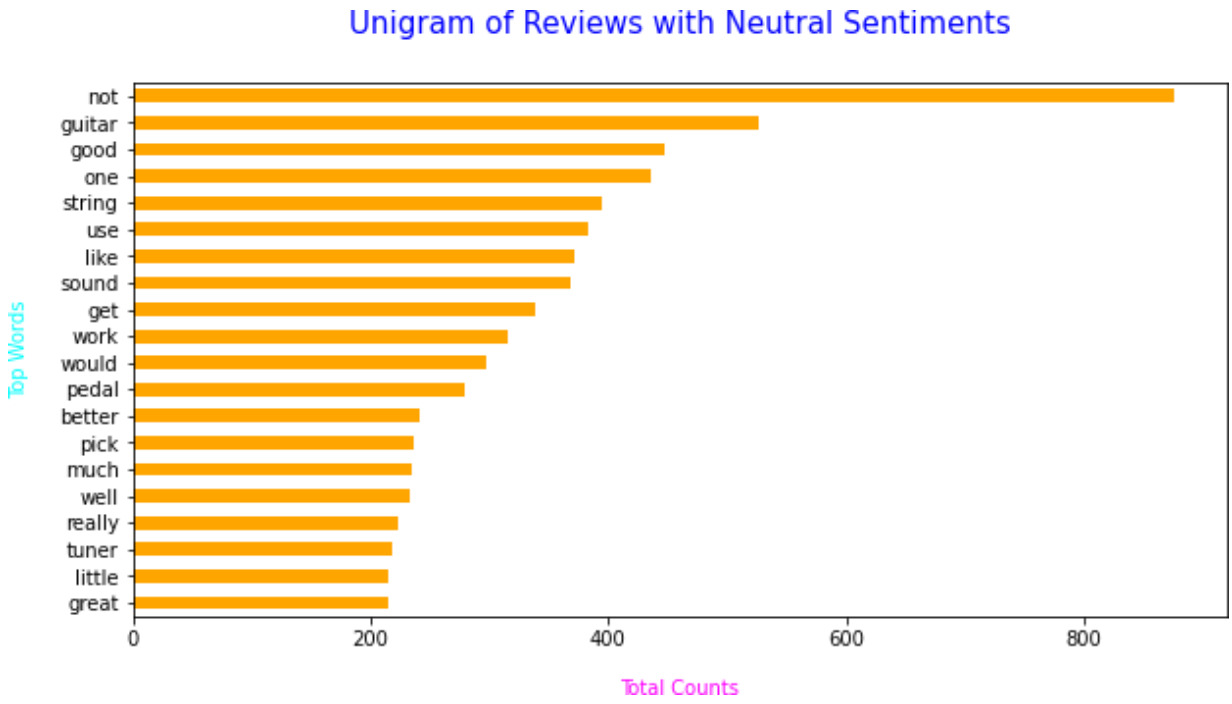
```
# Use dropna() so the base DataFrame is not affected
Positive = dataset[dataset["sentiment"] == "Positive"].dropna()
Neutral = dataset[dataset["sentiment"] == "Neutral"].dropna()
Negative = dataset[dataset["sentiment"] == "Negative"].dropna()
```

Unigram of Reviews Based on Sentiments

```
# Finding Unigram
words = Gram_Analysis(Positive["reviews"], 1, 20)
Unigram = pd.DataFrame(words, columns = ["Words", "Counts"])

# Visualization
Unigram.groupby("Words").sum()["Counts"].sort_values().plot(kind =
"barh", color = "green", figsize = (10, 5))
plt.title("Unigram of Reviews with Positive Sentiments", loc =
"center", fontsize = 15, color = "blue", pad = 25)
plt.xlabel("Total Counts", color = "magenta", fontsize = 10, labelpad
= 15)
plt.xticks(rotation = 0)
plt.ylabel("Top Words", color = "cyan", fontsize = 10, labelpad = 15)
plt.show()
```

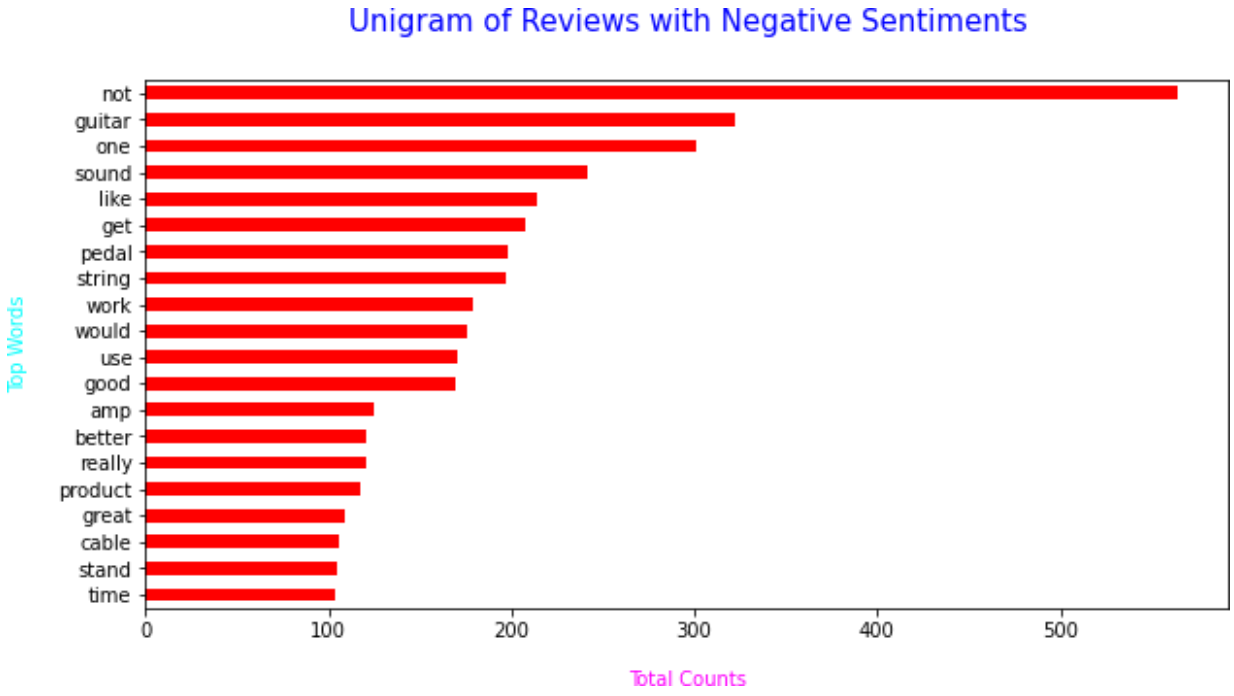


```
# Finding Unigram
words = Gram_Analysis(Neutral["reviews"], 1, 20)
Unigram = pd.DataFrame(words, columns = ["Words", "Counts"])

# Visualization
Unigram.groupby("Words").sum()["Counts"].sort_values().plot(kind =
"barh", color = "orange", figsize = (10, 5))
plt.title("Unigram of Reviews with Neutral Sentiments", loc =
"center", fontsize = 15, color = "blue", pad = 25)
plt.xlabel("Total Counts", color = "magenta", fontsize = 10, labelpad
= 15)
plt.xticks(rotation = 0)
plt.ylabel("Top Words", color = "cyan", fontsize = 10, labelpad = 15)
plt.show()
```




```
# Finding Unigram
words = Gram_Analysis(Negative["reviews"], 1, 20)
Unigram = pd.DataFrame(words, columns = ["Words", "Counts"])

# Visualization
Unigram.groupby("Words").sum()["Counts"].sort_values().plot(kind =
"barh", color = "red", figsize = (10, 5))
plt.title("Unigram of Reviews with Negative Sentiments", loc =
"center", fontsize = 15, color = "blue", pad = 25)
plt.xlabel("Total Counts", color = "magenta", fontsize = 10, labelpad
= 15)
plt.xticks(rotation = 0)
```

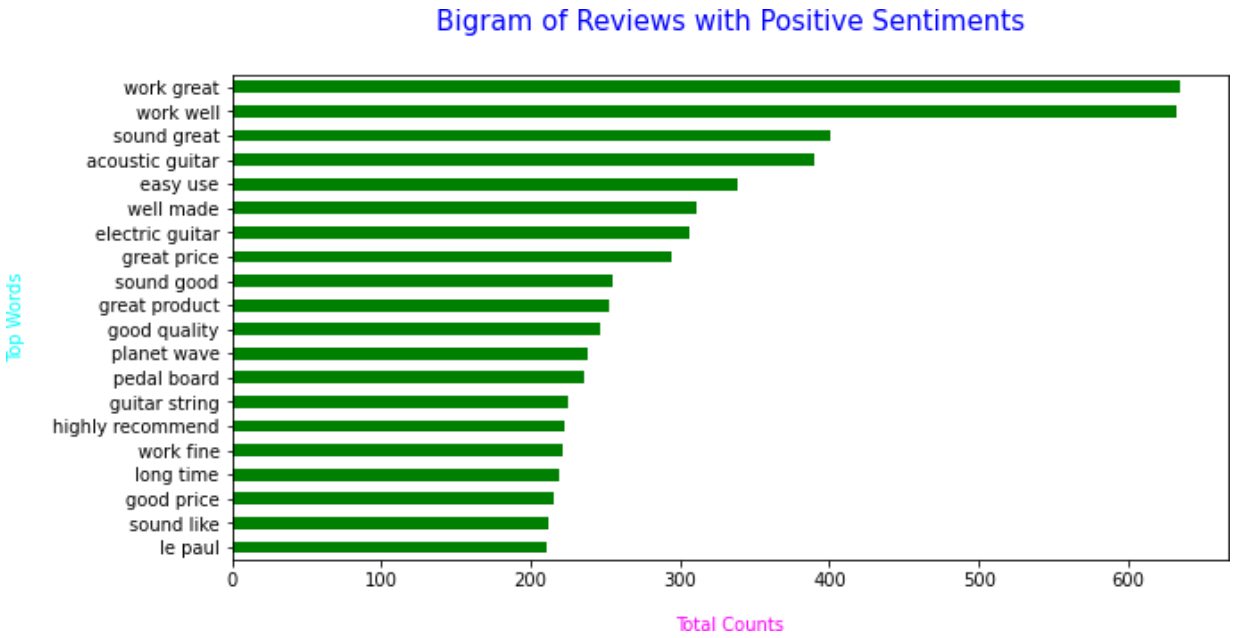
```
plt.ylabel("Top Words", color = "cyan", fontsize = 10, labelpad = 15)
plt.show()
```



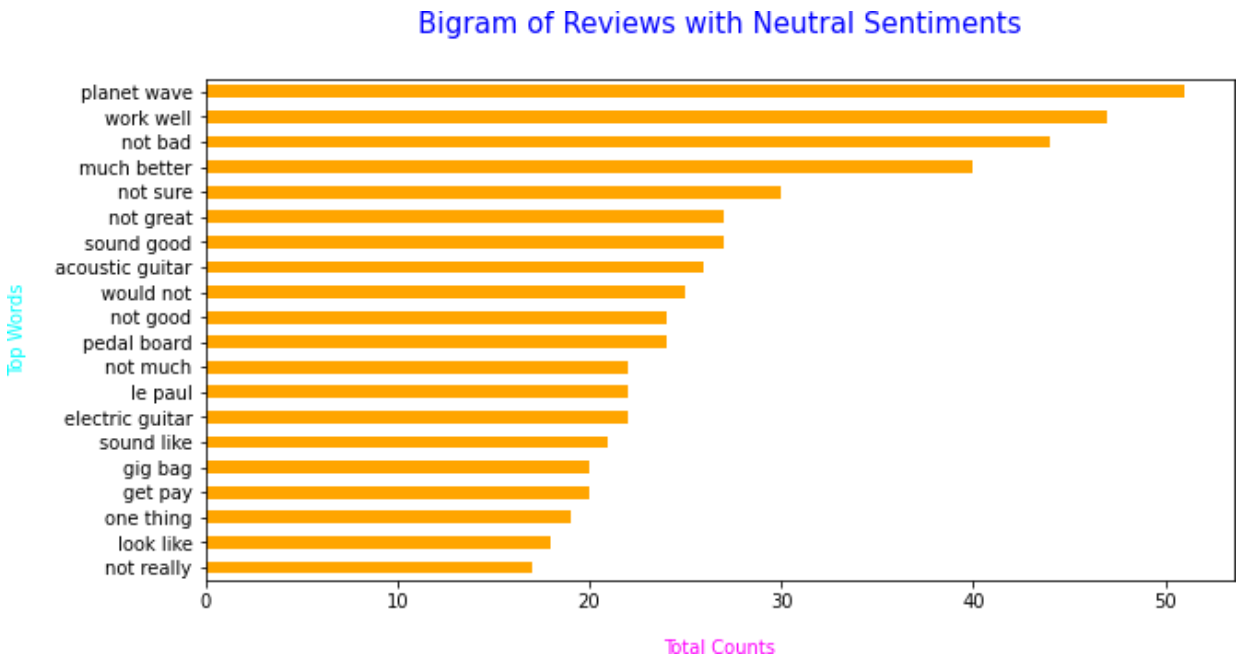
These unigrams are not entirely accurate for sentiment analysis, as we can see that even for positive sentiments, the most common unigram is "guitar," which is an object rather than an indicator of sentiment. However, this does suggest that guitars and related items are among the most frequently purchased. To gain more meaningful insights, we should explore bigrams and assess how accurately they represent each sentiment.

Bigram of Reviews Based On Sentiments

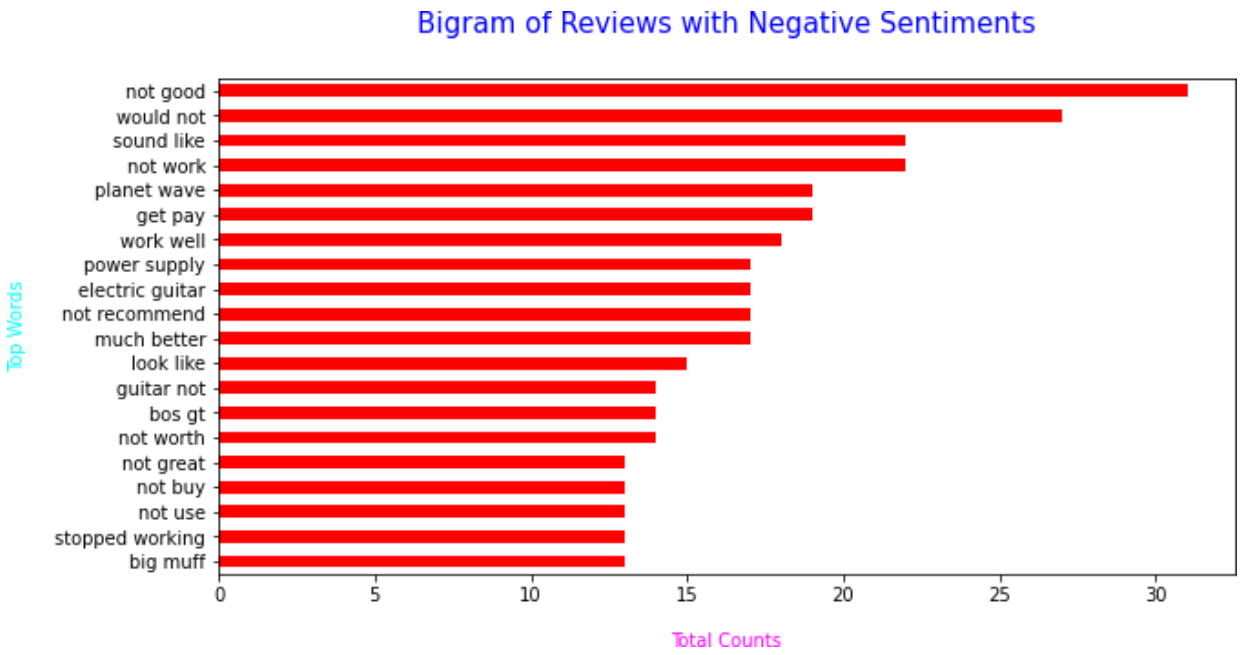
```
# Finding Bigram
words = Gram_Analysis(Positive["reviews"], 2, 20)
Bigram = pd.DataFrame(words, columns = ["Words", "Counts"])

# Visualization
Bigram.groupby("Words").sum()["Counts"].sort_values().plot(kind =
"barh", color = "green", figsize = (10, 5))
plt.title("Bigram of Reviews with Positive Sentiments", loc =
"center", fontsize = 15, color = "blue", pad = 25)
plt.xlabel("Total Counts", color = "magenta", fontsize = 10, labelpad
= 15)
plt.xticks(rotation = 0)
plt.ylabel("Top Words", color = "cyan", fontsize = 10, labelpad = 15)
plt.show()
```




```
# Finding Bigram
words = Gram_Analysis(Neutral["reviews"], 2, 20)
Bigram = pd.DataFrame(words, columns = ["Words", "Counts"])

# Visualization
Bigram.groupby("Words").sum()["Counts"].sort_values().plot(kind =
"barh", color = "orange", figsize = (10, 5))
plt.title("Bigram of Reviews with Neutral Sentiments", loc = "center",
fontsize = 15, color = "blue", pad = 25)
plt.xlabel("Total Counts", color = "magenta", fontsize = 10, labelpad
= 15)
plt.xticks(rotation = 0)
plt.ylabel("Top Words", color = "cyan", fontsize = 10, labelpad = 15)
plt.show()
```



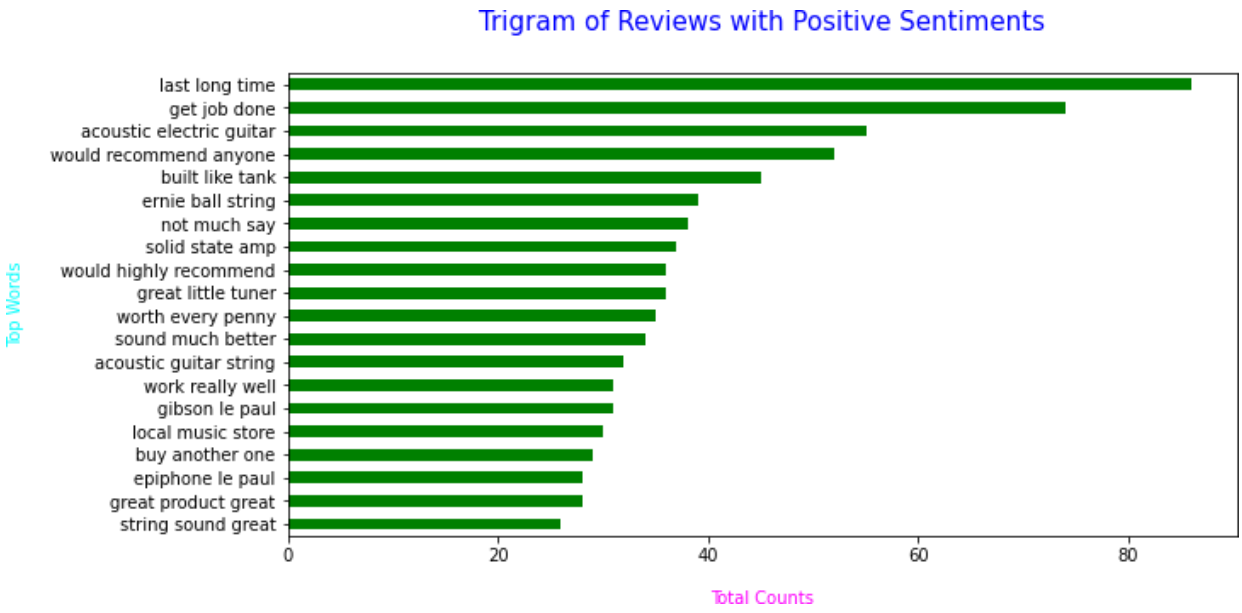

```
#FindingBigram
words=Gram_Analysis(Negative["reviews"],2,20)
Bigram=pd.DataFrame(words,columns=["Words","Counts"])

# Visualization
Bigram.groupby("Words").sum()["Counts"].sort_values().plot(kind=
"barh", color = "red", figsize = (10, 5))
plt.title("BigramofReviewswithNegativeSentiments",loc= "center",
fontsize = 15, color = "blue", pad = 25)
plt.xlabel("TotalCounts",color="magenta",fontsize=10,labelpad
=15)
plt.xticks(rotation=0)
plt.ylabel("TopWords",color= "cyan",fontsize = 10,labelpad = 15)
plt.show()
```



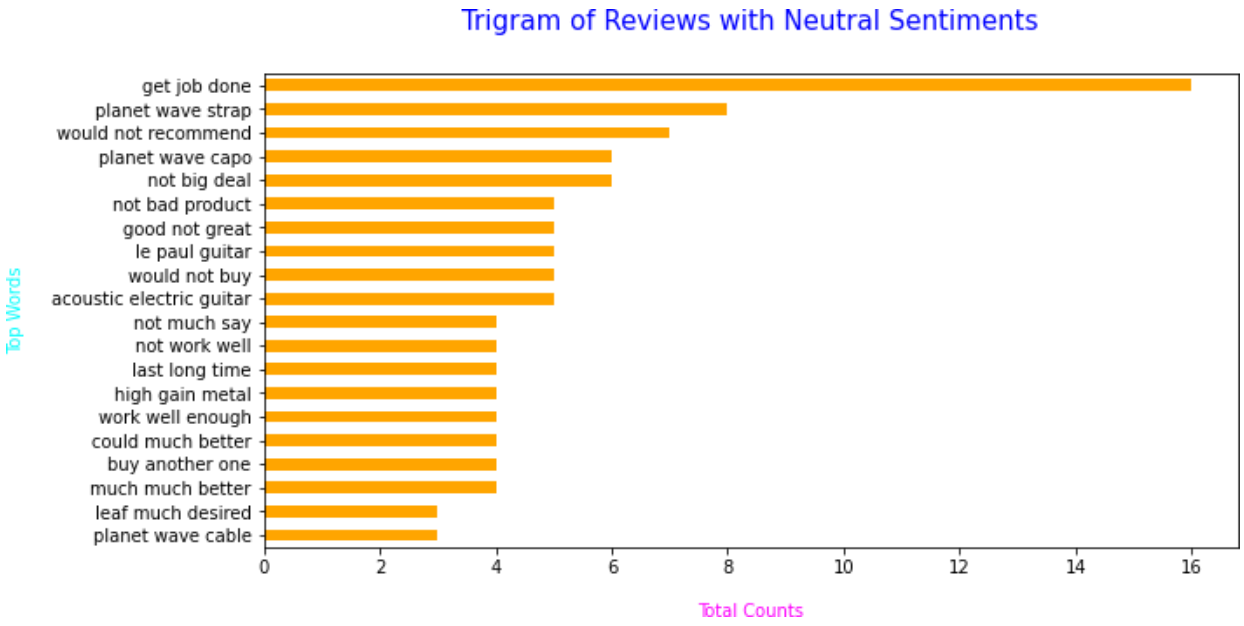
Bigrams provide more meaningful insights than unigrams, as they reveal phrases that better capture positive sentiments. However, we still notice that guitar-related terms appear among the top bigrams, reinforcing the interpretation that guitars and related items are among the most frequently purchased products. This confirms our earlier findings about the popularity of guitars in the dataset.

Trigram of Reviews Based On Sentiments

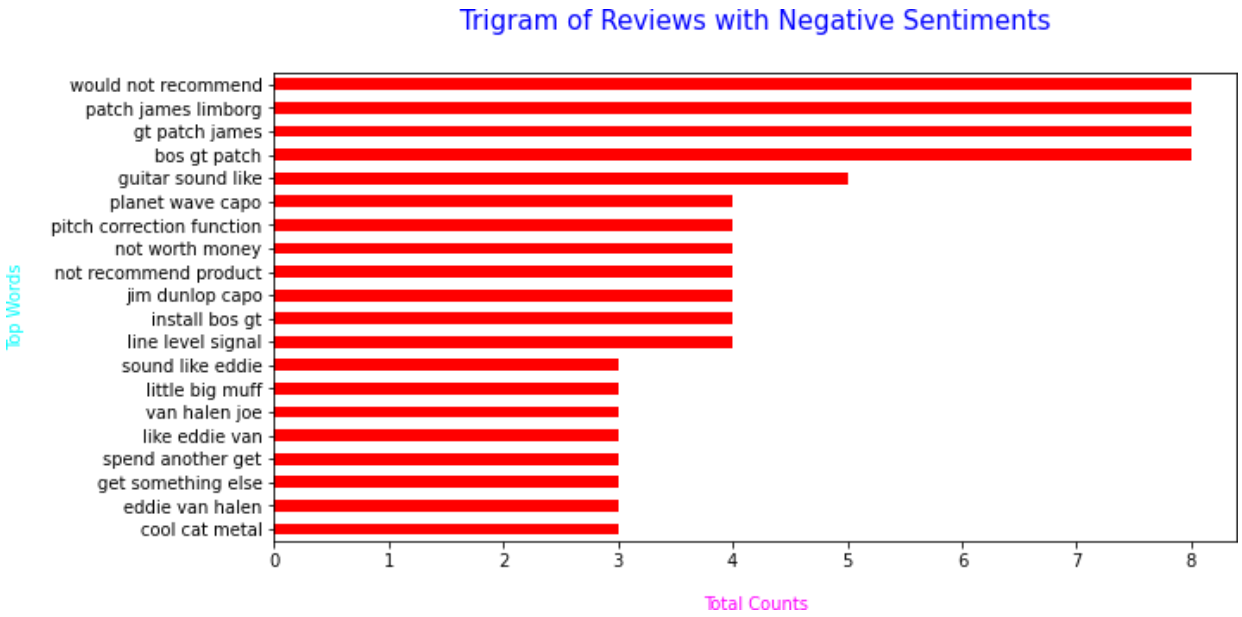
```
# Finding Trigram
words = Gram_Analysis(Positive["reviews"], 3, 20)
Trigram = pd.DataFrame(words, columns = ["Words", "Counts"])

# Visualization
Trigram.groupby("Words").sum()["Counts"].sort_values().plot(kind =
"barh", color = "green", figsize = (10, 5))
plt.title("Trigram of Reviews with Positive Sentiments", loc =
"center", fontsize = 15, color = "blue", pad = 25)
plt.xlabel("Total Counts", color = "magenta", fontsize = 10, labelpad
= 15)
plt.xticks(rotation = 0)
plt.ylabel("Top Words", color = "cyan", fontsize = 10, labelpad = 15)
plt.show()
```




```
# Finding Trigram
words = Gram_Analysis(Neutral["reviews"], 3, 20)
Trigram = pd.DataFrame(words, columns = ["Words", "Counts"])

# Visualization
Trigram.groupby("Words").sum()["Counts"].sort_values().plot(kind =
"barh", color = "orange", figsize = (10, 5))
plt.title("Trigram of Reviews with Neutral Sentiments", loc =
"center", fontsize = 15, color = "blue", pad = 25)
plt.xlabel("Total Counts", color = "magenta", fontsize = 10, labelpad
= 15)
plt.xticks(rotation = 0)
plt.ylabel("Top Words", color = "cyan", fontsize = 10, labelpad = 15)
plt.show()
```




```
# Finding Trigram
words = Gram_Analysis(Negative["reviews"], 3, 20)
Trigram = pd.DataFrame(words, columns = ["Words", "Counts"])

# Visualization
Trigram.groupby("Words").sum()["Counts"].sort_values().plot(kind =
"barh", color = "red", figsize = (10, 5))
plt.title("Trigram of Reviews with Negative Sentiments", loc =
"center", fontsize = 15, color = "blue", pad = 25)
plt.xlabel("Total Counts", color = "magenta", fontsize = 10, labelpad
= 15)
plt.xticks(rotation = 0)
plt.ylabel("Top Words", color = "cyan", fontsize = 10, labelpad = 15)
plt.show()
```



Trigrams provide a slight improvement in describing sentiments more effectively. Notably, negative trigrams convey significant information about poor product experiences, as seen from the prominent terms identified. The NGram Analysis also highlights the importance of our decision to exclude "not" from the list of stopwords. This choice helps preserve the meaning of negation phrases, which adds valuable context and clarity to the sentiment analysis.

Feature Engineering

Drop Insignificant Columns

```
Columns = ["reviewerID", "asin", "reviewerName", "helpful",
"unixReviewTime", "reviewTime", "polarity", "length", "word_counts",
"overall"]
dataset.drop(columns = Columns, axis = 1, inplace = True)
```

To streamline our dataset, we dropped certain columns, leaving us with two columns as independent variables and one column as the dependent variable. Next, we need to encode the labels into numeric values to represent each category for further processing.

Current State of The Dataset

```
dataset.head()
```

```
                                                reviewssentiment
0  notmuchwriteexactlysupposedfilterpopsou...Positive
1  productexactlyquiteaffordablenotrealized...Positive
2  primaryjobdeviceblockbreathwouldotherwis...Positive
3  nicewindscreenprotectsmxlmicpreventspop...Positive
4  popfiltergreatlookperformslikestudiofil...Positive
```

Encoding Our Target Variable

```
Encoder=LabelEncoder()
dataset["sentiment"]=Encoder.fit_transform(dataset["sentiment"])

dataset["sentiment"].value_counts()
```

```
2    9022
1     772
0     467
Name:sentiment,dtype:int64
```

We have successfully encoded the sentiment labels into numeric values to facilitate model training: Positive is encoded as 2, Neutral as 1, and Negative as 0. Next, we need to assign importance to each word in the reviews by weighting them. This can be achieved using the TF-IDF (Term Frequency - Inverse Document Frequency) Vectorizer, which helps quantify the relevance of words in the context of the entire dataset.

TF-IDF Vectorizer

```
# Defining our vectorizer with total words of 5000 and with bigram
model
TF_IDF = TfidfVectorizer(max_features = 5000, ngram_range = (2, 2))

# Fitting and transforming our reviews into a matrix of weighed words
# This will be our independent features
X = TF_IDF.fit_transform(dataset["reviews"])

# Check our matrix shape
X.shape
```

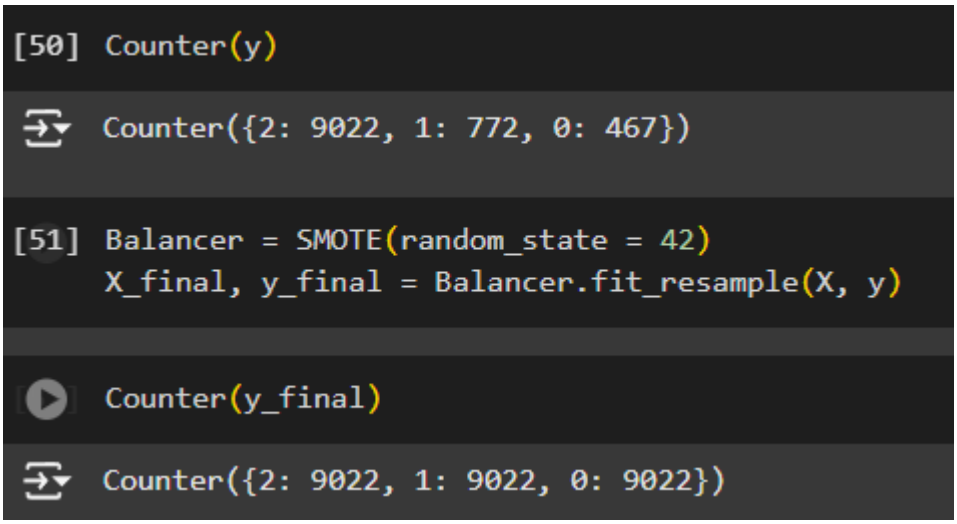
```
(10261, 5000)
```

```
# Declaring our target variable
y = dataset["sentiment"]
```

From the data shape, we have successfully transformed our reviews using the TF-IDF Vectorizer, focusing on the top 7,000 bigram words. However, as previously noted, our dataset is imbalanced, with relatively few neutral and negative sentiments compared to positive ones. To ensure a fair and effective modeling process, we need to balance the dataset before moving forward with training the model.

Resampling Our Dataset

There are several techniques to address imbalanced datasets, such as SMOTE (Synthetic Minority Oversampling Technique) and the Bootstrap Method. For this project, we will use SMOTE, which generates synthetic examples of the minority class to balance the dataset. This approach helps create a more even distribution and improves model performance by ensuring the model is exposed to more balanced training data.



Our data is now balanced, as shown by the counts of each sentiment category before and after applying SMOTE for resampling. This adjustment ensures that the model is trained on an even distribution of sentiments, which can lead to more robust and unbiased performance.

Splitting Our Dataset

```
X_train, X_test, y_train, y_test = train_test_split(X_final, y_final,
test_size = 0.25, random_state = 42)
```

We splitted our dataset into 75:25 portion respectively for the training and test set.

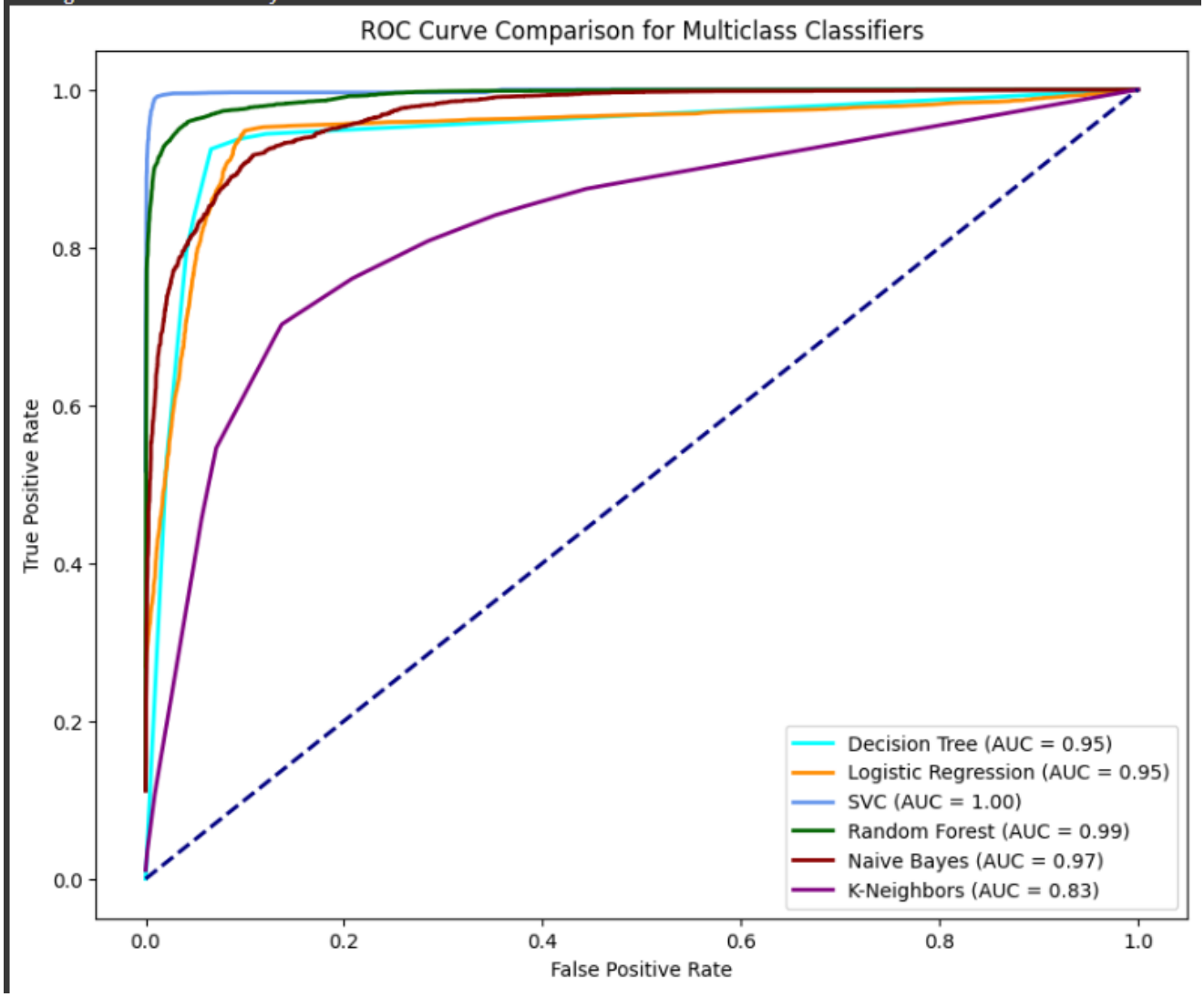
Model Selection and Evaluation

We are uncertain which model will best fit our data. To address this, we will test various classification models and evaluate them using the Confusion Matrix and F1 Score as primary metrics, with additional metrics serving as supplementary support. Before determining the best model, we need to apply cross-validation techniques to ensure reliable and generalizable performance across different models.

Model Building

We are applying K-Fold Cross Validation to our original dataset (before resampling) because this technique is effective even with imbalanced data. It splits the dataset into folds and evaluates each split, ensuring comprehensive validation across all subsets. If we were to use cross-validation on the balanced dataset obtained from resampling, we should expect similar results, reinforcing the reliability of our model evaluation.

```
Decision Tree Test Accuracy: 0.847
Logistic Regression Test Accuracy: 0.880
SVC Test Accuracy: 0.881
Random Forest Test Accuracy: 0.878
Naive Bayes Test Accuracy: 0.819
K-Neighbors Test Accuracy: 0.869
```



Choosing Logistic Regression as the classifier for this task is a strategic decision based on its balance between performance, simplicity, and robustness. Despite not having the highest test accuracy or AUC compared to more complex models like SVC or Random Forest, Logistic Regression offers competitive performance with an accuracy of 0.880 and an AUC of 0.95. This indicates a strong ability to distinguish between classes while maintaining good predictive accuracy. More importantly, Logistic Regression strikes an optimal balance in the bias-variance tradeoff; it has a low variance due to its simplicity, reducing the risk of overfitting, which is a common issue with high-variance models like Decision Trees and Random Forests. This ensures consistent, reliable generalization across different datasets, making it ideal for real-world applications where data can vary. Additionally, Logistic Regression's interpretability provides clear insights into feature impacts, which is invaluable for understanding and explaining model behavior. Overall, choosing Logistic Regression offers a robust, interpretable solution that performs well without sacrificing generalizability or transparency.

Hyperparameter Tuning

```
from sklearn.multiclass import OneVsRestClassifier
Param = {"estimator__C": np.logspace(-4, 4, 50),
         "estimator__penalty": ['l2'],  # Changed to 'l2' to be compatible with 'lbfgs' solver
         'estimator__solver': ['liblinear', 'saga'] # solvers compatible with l1 penalty
        }
# Wrap LogisticRegression with OneVsRestClassifier for multi-label classification
estimator = OneVsRestClassifier(LogisticRegression())
grid_search = GridSearchCV(estimator = estimator, # Use OneVsRestClassifier
                           param_grid = Param,
                           scoring = "accuracy",
                           cv = 10,
                           verbose = 0,
                           n_jobs = -1)

grid_search.fit(X_train, y_train) # y_train can now be a multi-label indicator matrix
best_accuracy = grid_search.best_score_
best_parameters = grid_search.best_params_

print("Best Accuracy: {:.2f} %".format(best_accuracy*100))
print("Best Parameters:", best_parameters)
```

We got a nice accuracy on our training set, which is 95.21% and from our Grid Search, we are also able to find our optimal hyperparameters. It is time to finish our model using these parameters to get the best model of Logistic Regression.

Best Model

```
Classifier = OneVsRestClassifier(LogisticRegression(random_state = 42, C = 6866.488450042998, penalty = 'l2'))
Classifier.fit(X_train, y_train)

Prediction = Classifier.predict(X_test)
```

With our model now complete, we will proceed to test it on the designated test set. The evaluation metrics we will use to assess the model's performance will be based on these predictions. This step will help us understand how well the model generalizes to unseen data.

Metrics

Accuracy On Test Set

```
accuracy_score(y_test,Prediction)

0.9521205851928476
```

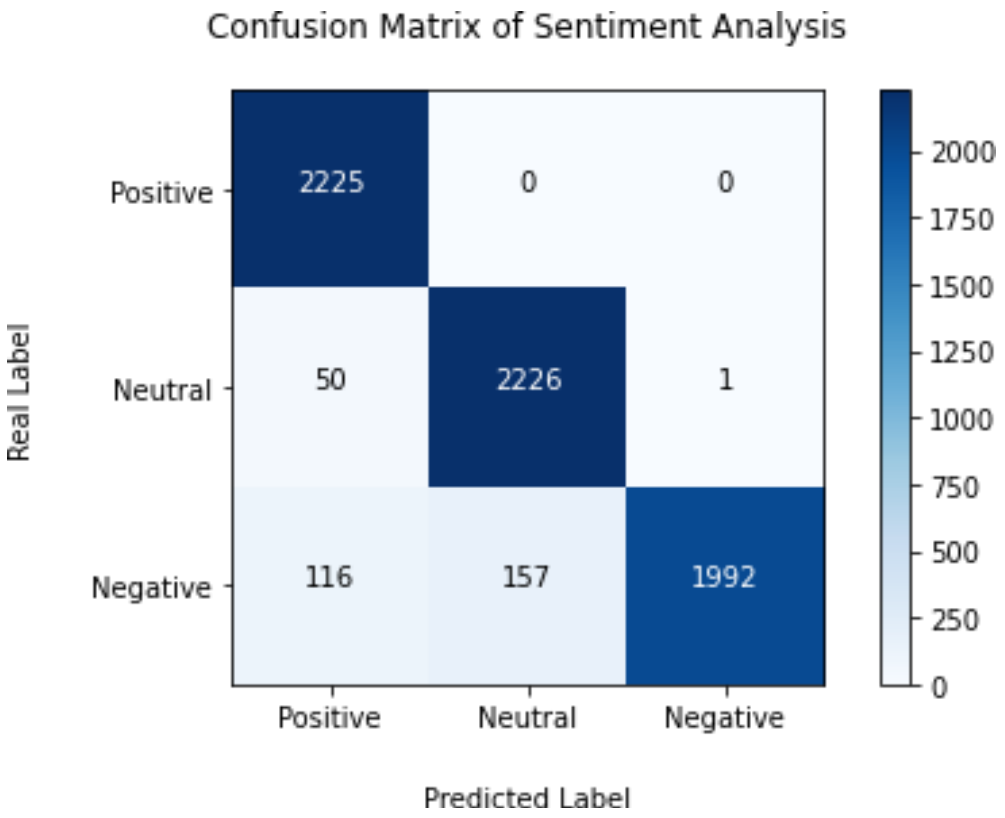
Achieving a high accuracy of 95.21% is impressive, but it's essential to delve deeper into the model's performance. We should examine the Confusion Matrix and F1 Score to get a

clearer picture of how well the model handles different classes, especially given the prior imbalance in the dataset. This analysis will help determine if the model is performing well across all categories or if it is biased towards the dominant class.

Confusion Matrix

```
ConfusionMatrix=confusion_matrix(y_test,Prediction)
```

Visualizing Our Confusion Matrix



The Confusion Matrix shows that the model performs well overall, accurately classifying positive and neutral sentiments. However, it struggles somewhat with identifying negative sentiments, which can be attributed to the initial data imbalance in the original dataset. Fortunately, the impact of this imbalance was mitigated by the SMOTE resampling we applied earlier, helping to improve the model's ability to handle underrepresented classes.

The Confusion Matrix indicates that the model performs well overall, accurately classifying both positive and neutral sentiments. However, it does show some difficulty in correctly identifying negative sentiments. This is likely due to the imbalance in the original dataset, which resulted in fewer examples of negative sentiments for the model to learn from. Fortunately, by applying SMOTE resampling, we were able to reduce the impact of this imbalance, improving the model's capability to better handle underrepresented classes.

Classification Scores

	precision	recall	f1-score	support
0	0.93	1.00	0.96	2225
1	0.93	0.98	0.96	2277
2	1.00	0.88	0.94	2265
accuracy			0.95	6767
macro avg	0.95	0.95	0.95	6767
weighted avg	0.95	0.95	0.95	6767

Overall, we achieved an F1 Score of 95% for each sentiment category, which is excellent. This high score indicates that our model performs well across all classes, balancing precision and recall effectively. Based on this result, we can confidently conclude that our model is robust and performs well on the dataset.