Sentiment Analysis

This is a notebook containing **Sentiment Analysis** Mini Project on *Amazon Musical Instruments Reviews*. I am interested in Natural Language Processing and that is my motivation to make this project. I think that sentiment analysis has a really powerful impacts in business developments because we can gain so many insights from here.

Libraries

Data Libraries

In [1]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

NLP Text Libraries

In [2]:
```python
import string
import re
import nltk
import nltk.corpus
nltk.download("punkt")
nltk.download("stopwords")
nltk.download("wordnet")
from nltk.stem import WordNetLemmatizer
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\Shreyas\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\Shreyas\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\Shreyas\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

EDA Analysis

In [3]:
```python
# Text Polarity
from textblob import TextBlob

# Text Vectorizer
from sklearn.feature_extraction.text import CountVectorizer

# Word Cloud
from wordcloud import WordCloud
```

Feature Engineering

In [4]:
```python
# 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

In [5]:
```python
# 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 that we will use is taken from `Kaggle` website and can be downloaded here:

[`Amazon Musical Instruments Reviews` (https://www.kaggle.com/eswarchandt/amazon-music-reviews?select=Musical_instruments_reviews.csv)](https://www.kaggle.com/eswarchandt/amazon-music-reviews?select=Musical_instruments_reviews.csv)

There are two formats available of the dataset: `JSON` and `CSV`. We will use the `CSV` one in this project.

Overall, the dataset talks about the feedback received after the customers purchased musical instruments from `Amazon`.

Read The Dataset

In [6]:
```python
dataset = pd.read_csv("Instruments_Reviews.csv")
```

Shape of The Dataset

In [7]:
```python
dataset.shape
```

Out[7]:
```
(10261, 9)
```

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

Data Preprocessing

Checking Null Values

In [8]:
```
dataset.isnull().sum()
```

Out[8]:
```
reviewerID         0
asin               0
reviewerName      27
helpful            0
reviewText         7
overall            0
summary            0
unixReviewTime     0
reviewTime         0
dtype: int64
```

From above, there are two columns in the dataset with null values: `reviewText` and `reviewerName` . While the latter one is not really important, we should focus on the first column. We cannot remove these rows because the ratings and summary given from the customers will have some effects to our model later (although the number of missing rows is small). Because of it, we can fill the empty values with an empty string.

Filling Missing Values

In [9]:
```
dataset.reviewText.fillna(value = "", inplace = True)
```

Concatenate `reviewText` and `summary` Columns

In [10]:
```
dataset["reviews"] = dataset["reviewText"] + " " + dataset["summary"]
dataset.drop(columns = ["reviewText", "summary"], axis = 1, inplace = True)
```

Statistic Description of The Dataset

In [11]:
```
dataset.describe(include = "all")
```

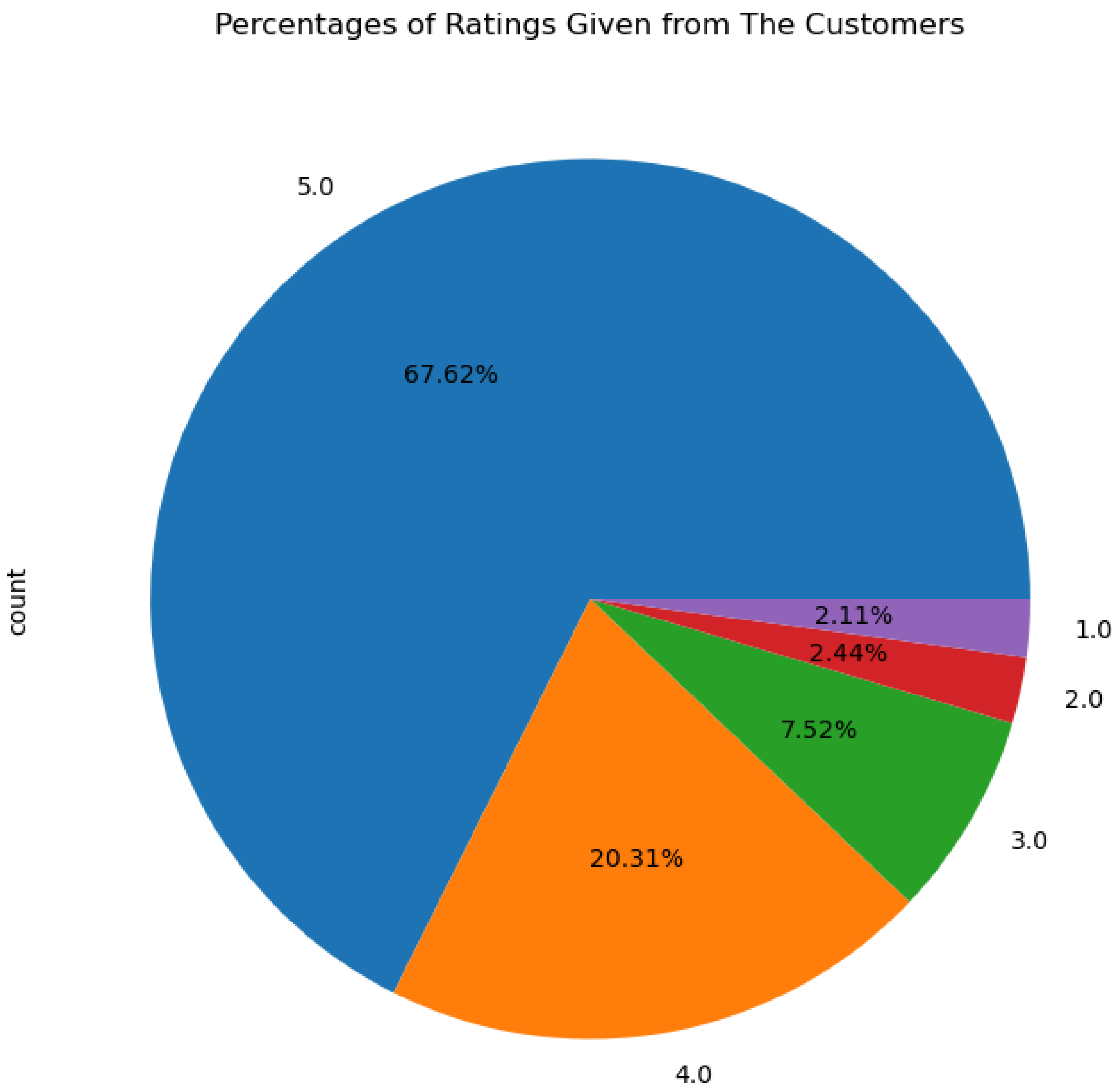
Out[11]:

	reviewerID	asin	reviewerName	helpful	overall	unixReviewTime	reviewTim
count	10261	10261	10234	10261	10261.000000	1.026100e+04	1026
unique	1429	900	1397	269	NaN	NaN	157
top	ADH0O8UVJOT10	B003VWJ2K8	Amazon Customer	[0, 0]	NaN	NaN	01 22, 201
freq	42	163	66	6796	NaN	NaN	4
mean	NaN	NaN	NaN	NaN	4.488744	1.360606e+09	Na
std	NaN	NaN	NaN	NaN	0.894642	3.779735e+07	Na
min	NaN	NaN	NaN	NaN	1.000000	1.095466e+09	Na
25%	NaN	NaN	NaN	NaN	4.000000	1.343434e+09	Na
50%	NaN	NaN	NaN	NaN	5.000000	1.368490e+09	Na
75%	NaN	NaN	NaN	NaN	5.000000	1.388966e+09	Na
max	NaN	NaN	NaN	NaN	5.000000	1.405987e+09	Na

From the description above, we know that the ratings given from the customers will have the range of [1, 5] as shown above. Also, the average rating given to musical instruments sold is 4.48. We can also see our new column `reviews` is there to concate both `summary` and `reviewText` .

Percentages of Ratings Given from The Customers

In [12]:
```
dataset.overall.value_counts().plot(kind = "pie", legend = False, autopct = "%1.2f%%"
plt.title("Percentages of Ratings Given from The Customers", loc = "center")
plt.show()
```



From the chart above, the majority of musical instruments sold on Amazon have perfect ratings of 5.0, meaning the condition of the products are good. If we were to denote that ratings above 3 are `positive` , ratings equal to 3 are `neutral` , and ratings under 3 are `negative` , we know that the number of negative reviews given in the dataset are relatively small. This might affect our model later.

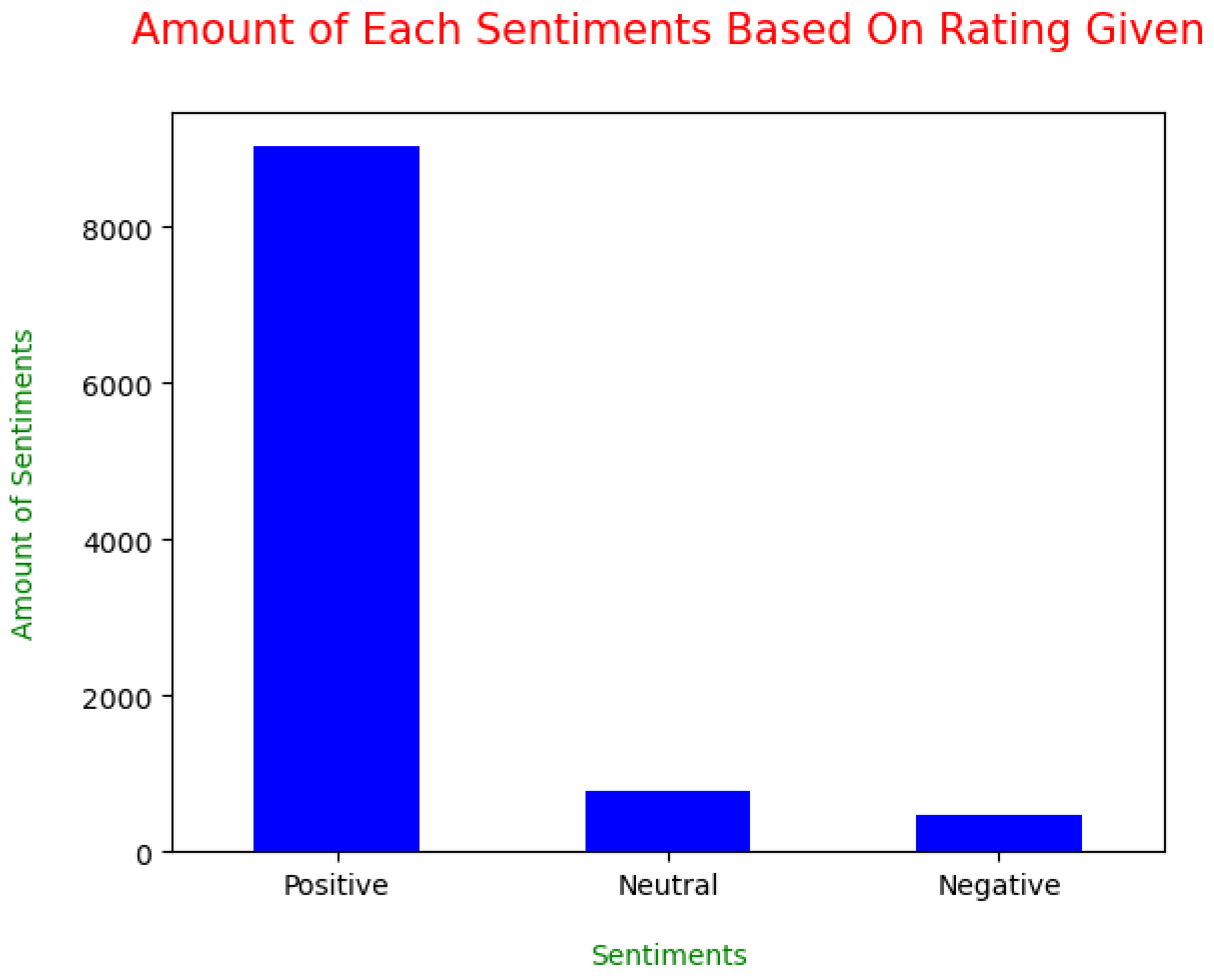
Labelling Products Based On Ratings Given

Our dataset does not have any dependent variable, or in other words we haven't had any prediction target yet. We will categorize each sentiment according to ratings given for each row based on the explanation before: `Positive` Label for products with rating bigger than 3.0, `Neutral` Label for products with rating equal to 3.0, else `Negative` Label.

In [13]:
```python
def Labelling(Rows):
  if(Rows["overall"] > 3.0):
    Label = "Positive"
  elif(Rows["overall"] < 3.0):
    Label = "Negative"
  else:
    Label = "Neutral"
  return Label
```

In [14]:
```python
dataset["sentiment"] = dataset.apply(Labelling, axis = 1)
```

In [15]:
```python
dataset["sentiment"].value_counts().plot(kind = "bar", color = "blue")
plt.title("Amount of Each Sentiments Based On Rating Given", loc = "center", fontsize
plt.xlabel("Sentiments", color = "green", fontsize = 10, labelpad = 15)
plt.xticks(rotation = 0)
plt.ylabel("Amount of Sentiments", color = "green", fontsize = 10, labelpad = 15)
plt.show()
```



In this part we can actually change the labels into numeric values but for the sake of experiments we will do it later. Also, notice that from the graph we can know that most of our data contains positive sentiments, which is true from the exploration before.

Text Preprocessing

Text Cleaning

In [16]:
```
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

In [17]:
```
# 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

In [18]:
```
dataset["reviews"] = dataset["reviews"].apply(lambda Text: Text_Cleaning(Text))
dataset["reviews"] = dataset["reviews"].apply(lambda Text: Text_Processing(Text))
```

Exploratory Data Analysis

Overview of The Dataset

In [19]:
```python
dataset.head(n = 10)
```

Out[19]:

	reviewerID	asin	reviewerName	helpful	overall	unixReviewTime	reviewTime	revie
0	A2IBPI20UZIR0U	1384719342	cassandra tu "Yeah, well, that's just like, u...	[0, 0]	5.0	1393545600	02 28, 2014	not m w exa suppo filter sc
1	A14VAT5EAX3D9S	1384719342	Jake	[13, 14]	5.0	1363392000	03 16, 2013	proc exa q afforda realize
2	A195EZSQDW3E21	1384719342	Rick Bennette "Rick Bennette"	[1, 1]	5.0	1377648000	08 28, 2013	primary dev bl bre wo otherw
3	A2C00NNG1ZQQG2	1384719342	RustyBill "Sunday Rocker"	[0, 0]	5.0	1392336000	02 14, 2014	r windscr prote mxl preve po
4	A94QU4C90B1AX	1384719342	SEAN MASLANKA	[0, 0]	5.0	1392940800	02 21, 2014	pop f great l perfo like stu
5	A2A039TZMZHH9Y	B00004Y2UT	Bill Lewey "blewey"	[0, 0]	5.0	1356048000	12 21, 2012	g bou anot one l heavy c gold
6	A1UPZM995ZAH90	B00004Y2UT	Brian	[0, 0]	5.0	1390089600	01 19, 2014	u mon cable y g rea lifetime
7	AJNFQI3YR6XJ5	B00004Y2UT	Fender Guy "Rick"	[0, 0]	3.0	1353024000	11 16, 2012	use ca run ou pe chain ir fende
8	A3M1PLEYNDEYO8	B00004Y2UT	G. Thomas "Tom"	[0, 0]	5.0	1215302400	07 6, 2008	per epiph sherato mon cable w
9	AMNTZU1YQN1TH	B00004Y2UT	Kurt Robair	[0, 0]	5.0	1389139200	01 8, 2014	mon make b ca lifet warra doe

With the overview above, we know that for sentiment analysis that we will do, `reviews` is important to our model and we should use this aspect as our feature. By using this feature, we will need to predict what our sentiment will be classified into.

About Other Features

In [20]:
```
dataset.describe(include = "all")
```

Out[20]:

	reviewerID	asin	reviewerName	helpful	overall	unixReviewTime	reviewTim
count	10261	10261	10234	10261	10261.000000	1.026100e+04	1026
unique	1429	900	1397	269	NaN	NaN	157
top	ADH0O8UVJOT10	B003VWJ2K8	Amazon Customer	[0, 0]	NaN	NaN	01 22, 201
freq	42	163	66	6796	NaN	NaN	4
mean	NaN	NaN	NaN	NaN	4.488744	1.360606e+09	Na
std	NaN	NaN	NaN	NaN	0.894642	3.779735e+07	Na
min	NaN	NaN	NaN	NaN	1.000000	1.095466e+09	Na
25%	NaN	NaN	NaN	NaN	4.000000	1.343434e+09	Na
50%	NaN	NaN	NaN	NaN	5.000000	1.368490e+09	Na
75%	NaN	NaN	NaN	NaN	5.000000	1.388966e+09	Na
max	NaN	NaN	NaN	NaN	5.000000	1.405987e+09	Na

Now, we will go back to statistic description of our dataset. Intuitively, the other features from our dataset does not really have any impact in determining our sentiment later. We might use the `helpful` part in our model, but as we can see from the description above, the top values of it is [0,0], which means that most users do not really take their votes in it. Because of it, we can also decide that we don't really need it in our model.

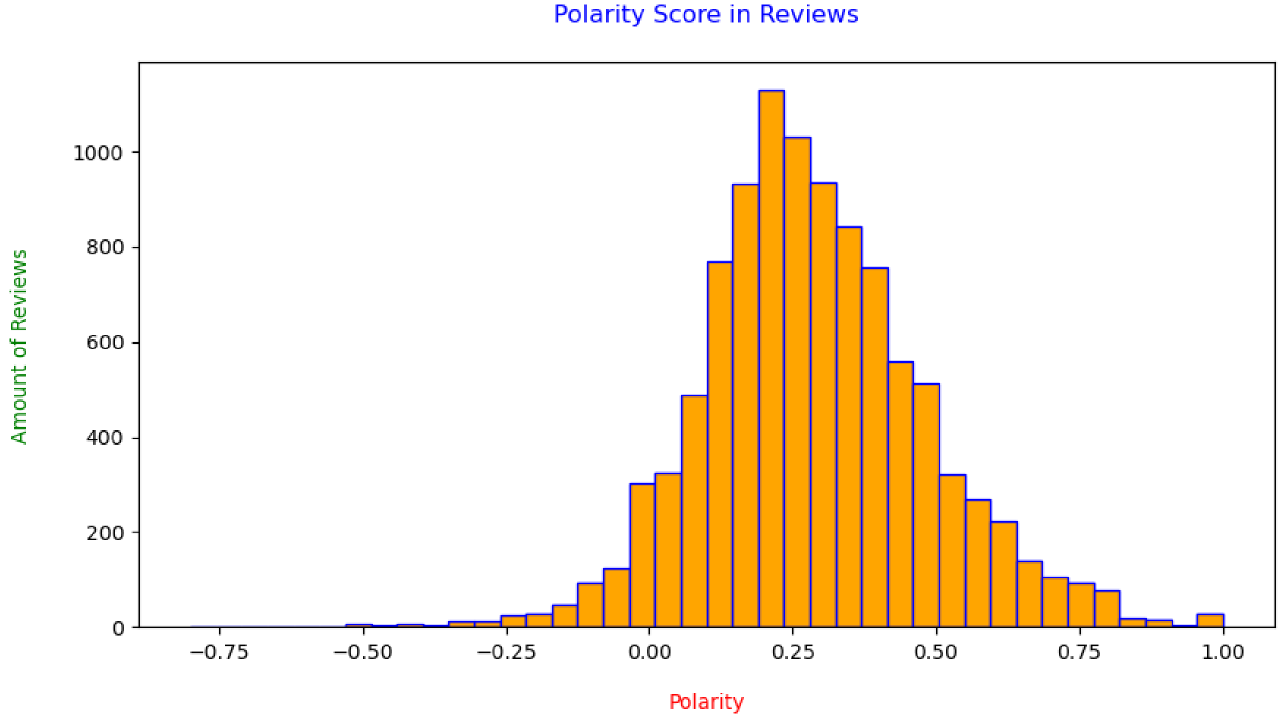
Polarity, Review Length, and Word Counts

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.

Polarity

In [21]:
```
dataset["polarity"] = dataset["reviews"].map(lambda Text: TextBlob(Text).sentiment.pc
```

In [22]:
```
dataset["polarity"].plot(kind = "hist", bins = 40, edgecolor = "blue", linewidth = 1,
plt.title("Polarity Score in Reviews", color = "blue", pad = 20)
plt.xlabel("Polarity", labelpad = 15, color = "red")
plt.ylabel("Amount of Reviews", labelpad = 20, color = "green")

plt.show()
```



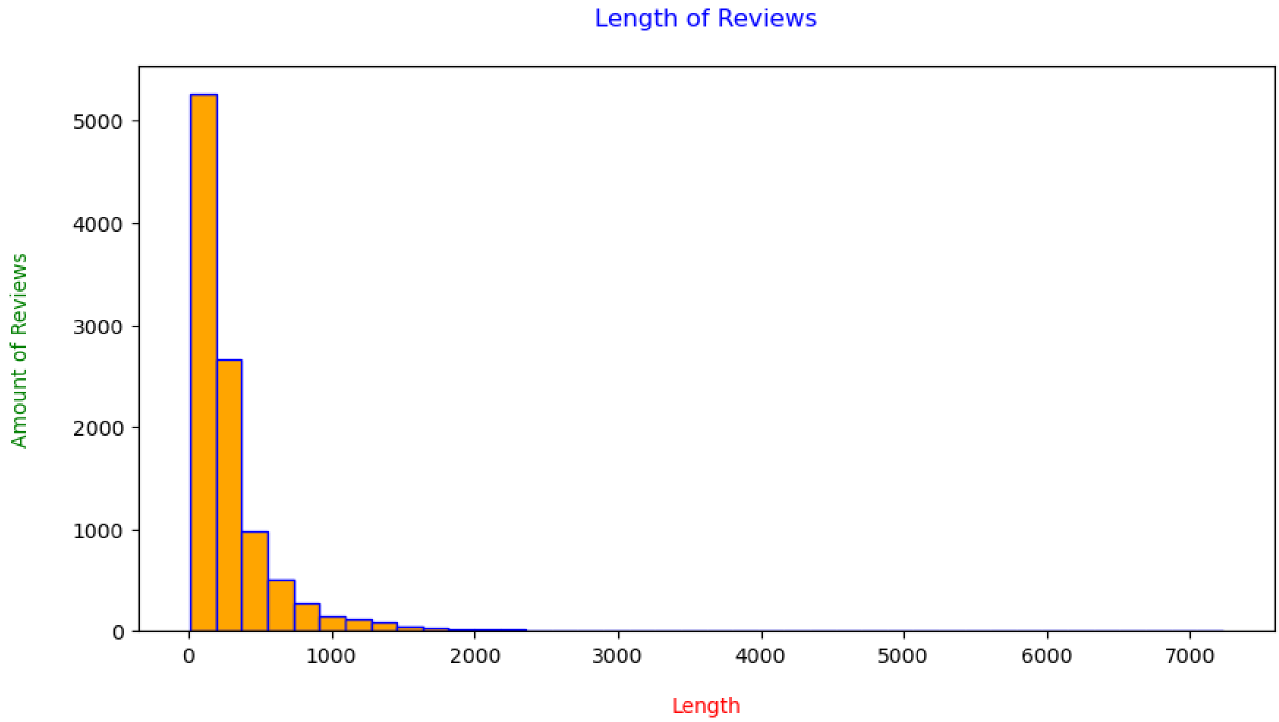
Reviews with negative polarity will be in range of [-1, 0), neutral ones will be 0.0, and positive reviews will have the range of (0, 1].

From the histogram above, we know that most of the reviews are distributed in positive sentiments, meaning that what we extracted from our analysis before is true. Statistically, this histogram shows that our data is normally distributed, but not with standard distribution. In conclusion, we know for sure that our analysis about the amount of sentiments from the reviews is correct and corresponds to the histogram above.

Review Length

In [23]:
```
dataset["length"] = dataset["reviews"].astype(str).apply(len)
```

In [24]:
```
dataset["length"].plot(kind = "hist", bins = 40, edgecolor = "blue", linewidth = 1, c
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()
```

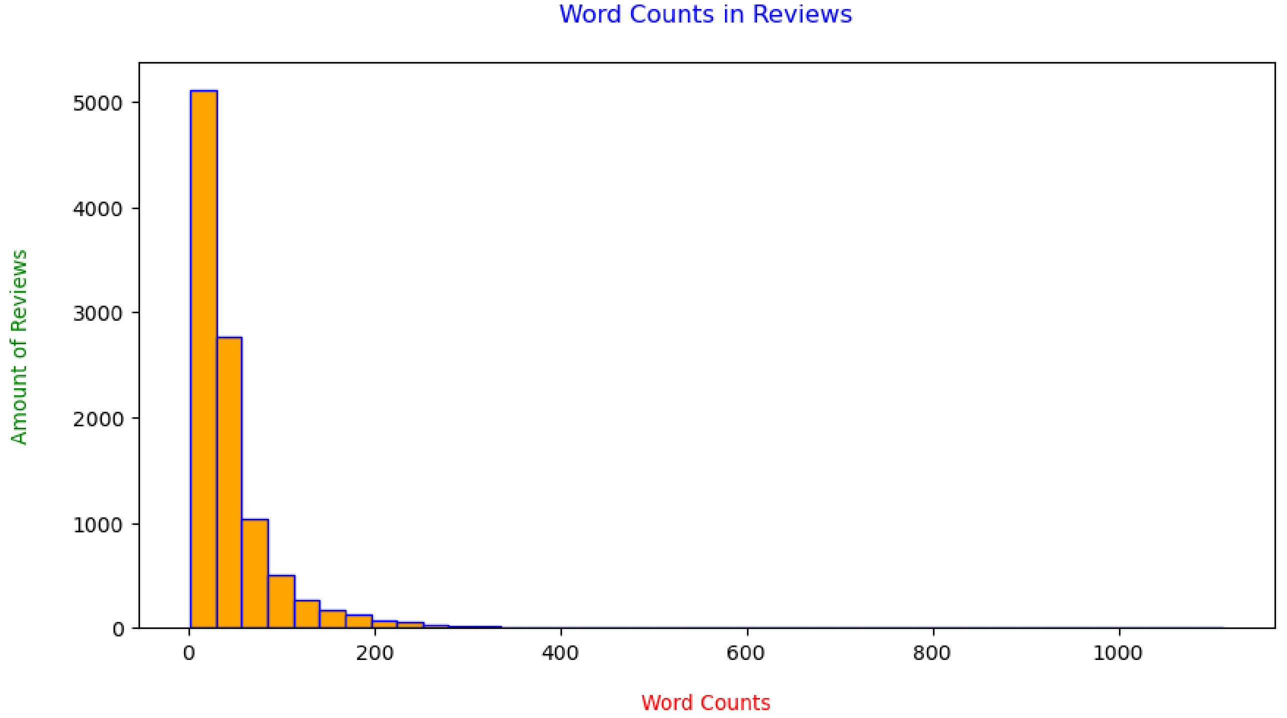


Based on this, we know that our review has text length between approximately 0-1000 characters. The distribution itself has positive skewness, or in other words it is skewed right, and this means that our reviews rarely has larger length than 1000 characters. Of course, the review that we use here is affected by the text preprocessing phase, so the length might not be the actual value of the review itself as some words might have been omitted already. This will also have the same effect when we count the tatal of words in our reviews.

Word Counts

In [25]:
```
dataset["word_counts"] = dataset["reviews"].apply(lambda x: len(str(x).split()))
```

In [26]:
```
dataset["word_counts"].plot(kind = "hist", bins = 40, edgecolor = "blue", linewidth =
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 infer that most of the reviews consist of 0-200 words. Just like before, the distribution is skewed right and the calculation is affected by our text preprocessing phase before.

N-Gram Analysis

N-Gram Function

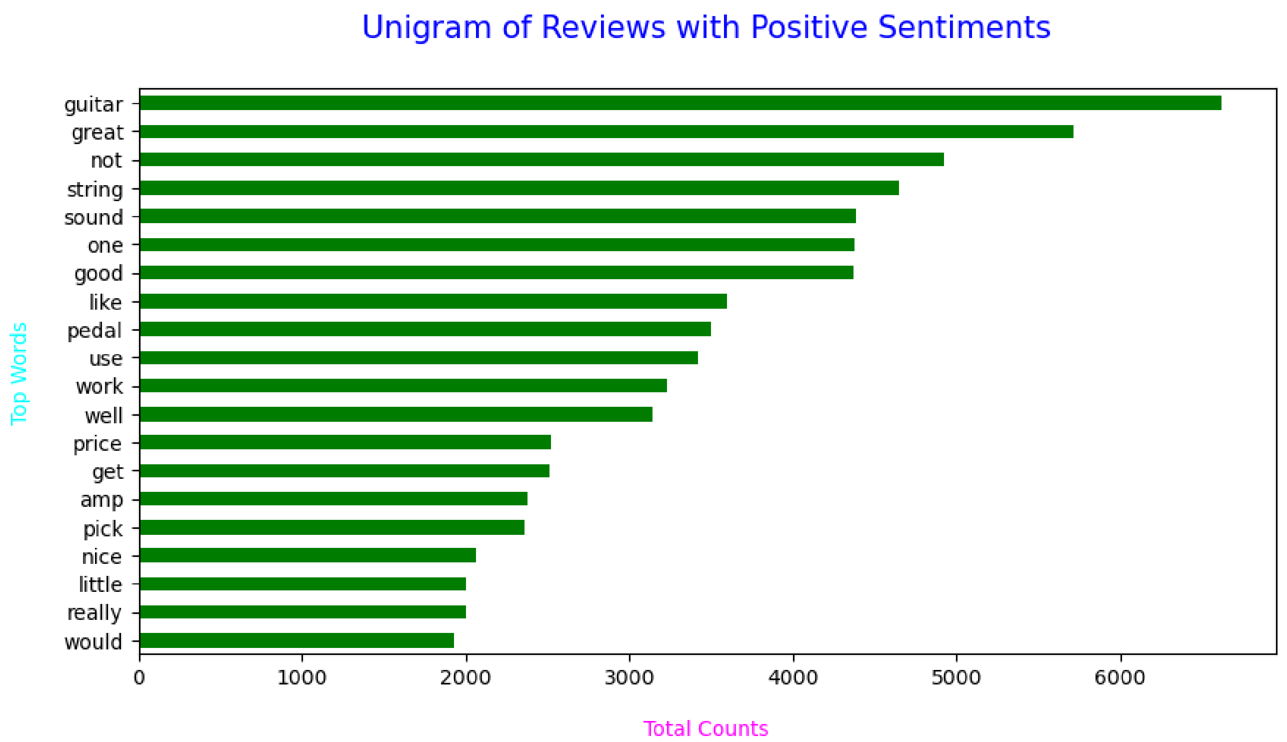
In [27]:
```
def Gram_Analysis(Corpus, Gram, N):
  # Vectorizer
  Vectorizer = CountVectorizer(stop_words = Stopwords, ngram_range=(Gram,Gram))

  # 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

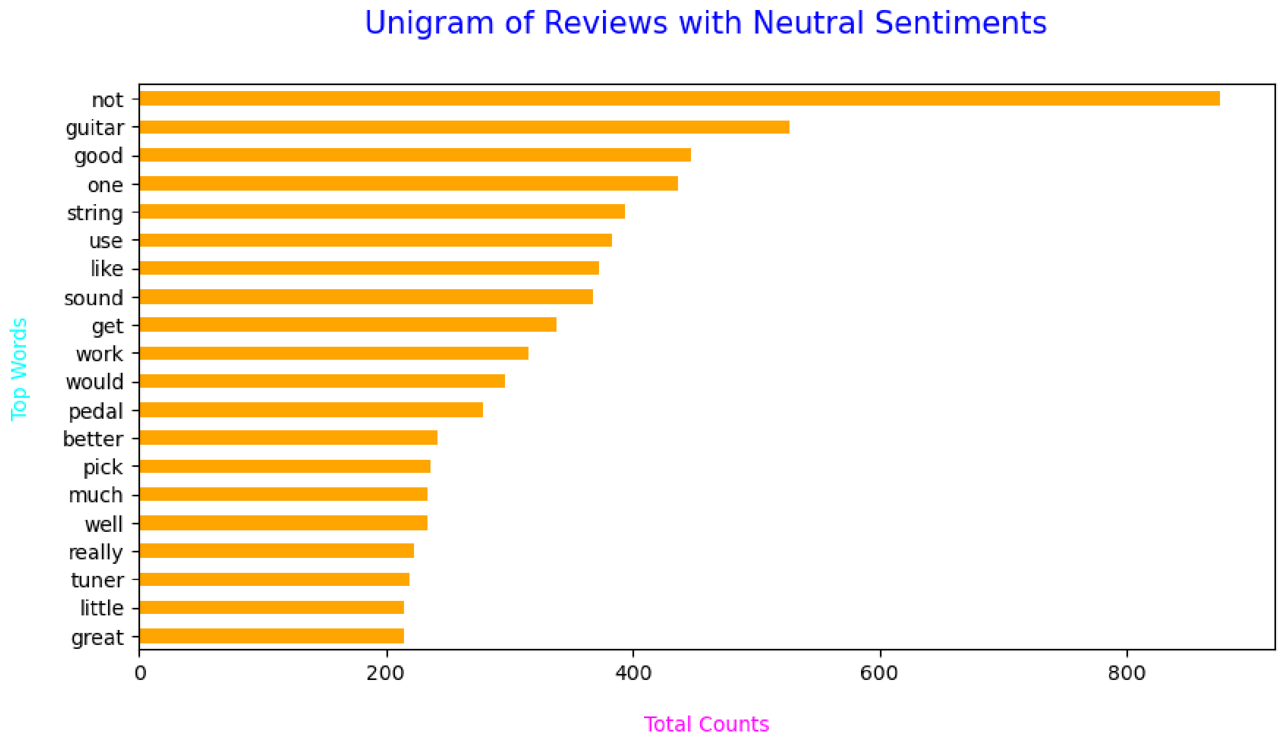
In [28]:
```python
# Use dropna() so the base DataFrame is not affected
Positive = dataset[dataset["sentiment"] == "Positive"].dropna()
Neutral = dataset[dataset["sentiment"] == "Neutral"].dropna()
Negative = dataset[dataset["sentiment"] == "Negative"].dropna()
```

Unigram of Reviews Based on Sentiments

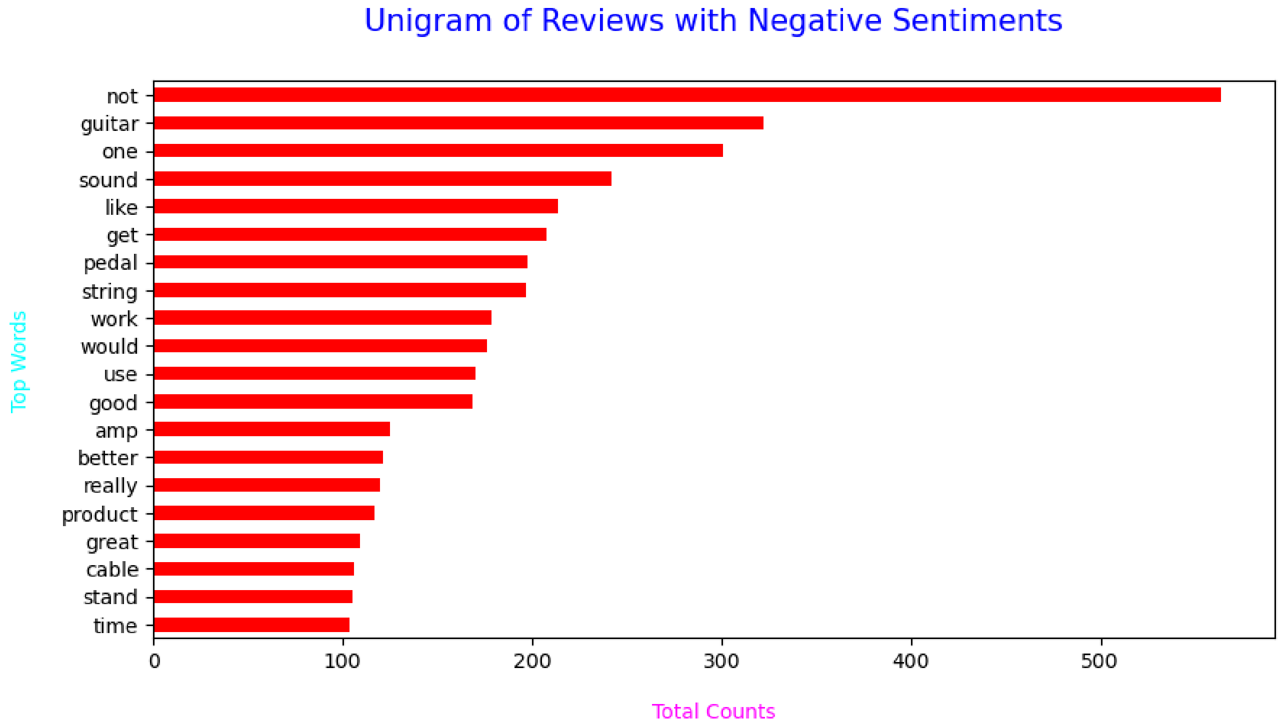
In [29]:
```python
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
import matplotlib.pyplot as plt

# Function to analyze grams
def Gram_Analysis(Corpus, Gram, N):
    Vectorizer = CountVectorizer(stop_words=list(Stopwords), ngram_range=(Gram, Gram)
    ngrams = Vectorizer.fit_transform(Corpus)
    Count = ngrams.sum(axis=0)
    words_freq = [(word, Count[0, idx]) for word, idx in Vectorizer.vocabulary_.items
    words_freq = sorted(words_freq, key=lambda x: x[1], reverse=True)
    return words_freq[:N]

# 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="greer
plt.title("Unigram of Reviews with Positive Sentiments", loc="center", fontsize=15, c
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()
```



In [30]:
```
# 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 = "c
plt.title("Unigram of Reviews with Neutral Sentiments", loc = "center", fontsize = 15
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()
```



In [31]:
```
# 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 = "r
plt.title("Unigram of Reviews with Negative Sentiments", loc = "center", fontsize = 1
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()
```

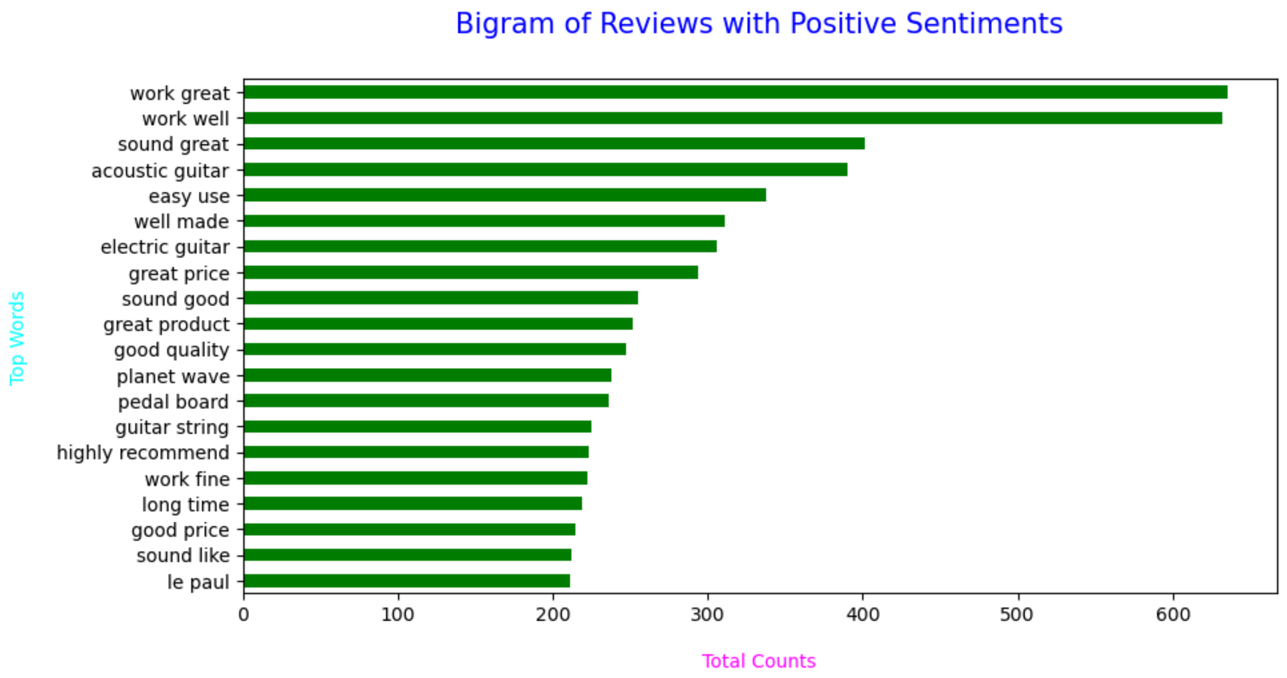


These unigrams are not really accurate, because we can clearly see that even for postive sentiments, the top unigram is the wird `guitar` which is an object, though from here we might know that the most frequently bought items are guitars or the complement of it. We should try to find the bigram and see how accurate it can describe each sentiments

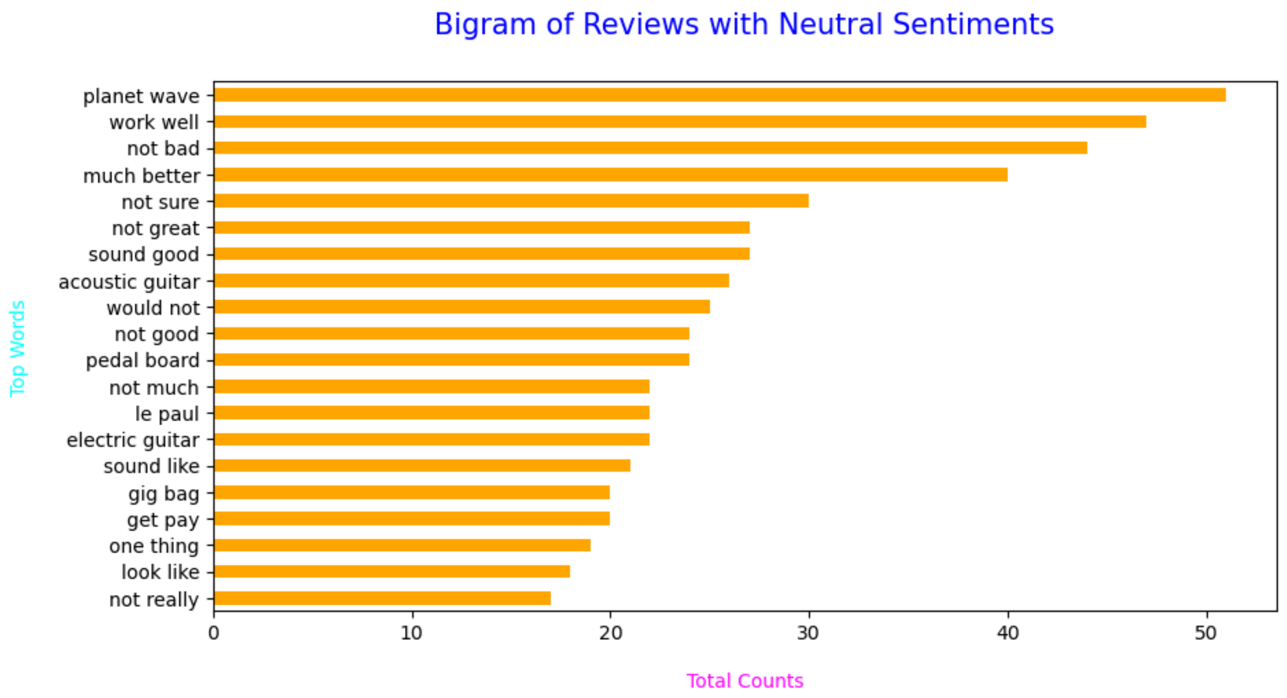
Bigram of Reviews Based On Sentiments

In [32]:

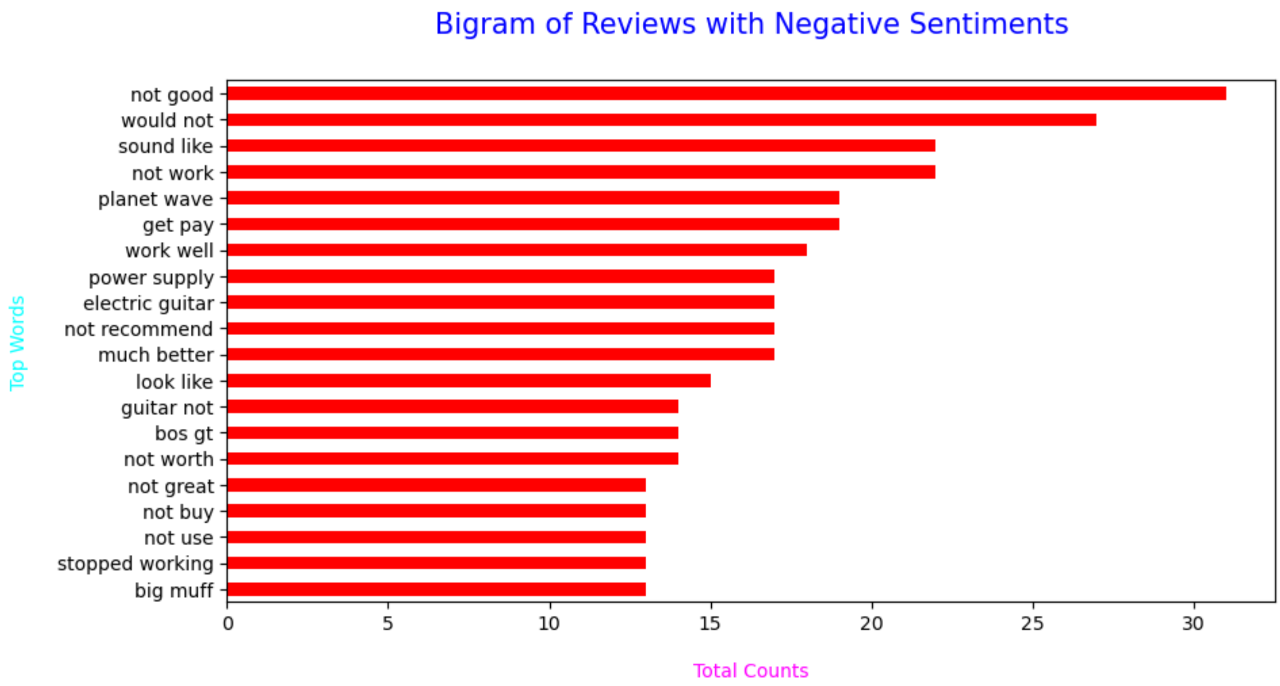
```
# Finding Bigram
words = Gram_Analysis(Positive["reviews"], 2, 20)
Bigram = pd.DataFrame(words, columns = ["Words", "Counts"])

# Visualization
Bigram.groupby("Words").sum()["Counts"].sort_values().plot(kind = "barh", color = "gr
plt.title("Bigram of Reviews with Positive Sentiments", loc = "center", fontsize = 15
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()
```



In [33]:
```
# 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 = "or
plt.title("Bigram of Reviews with Neutral Sentiments", loc = "center", fontsize = 15,
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()
```



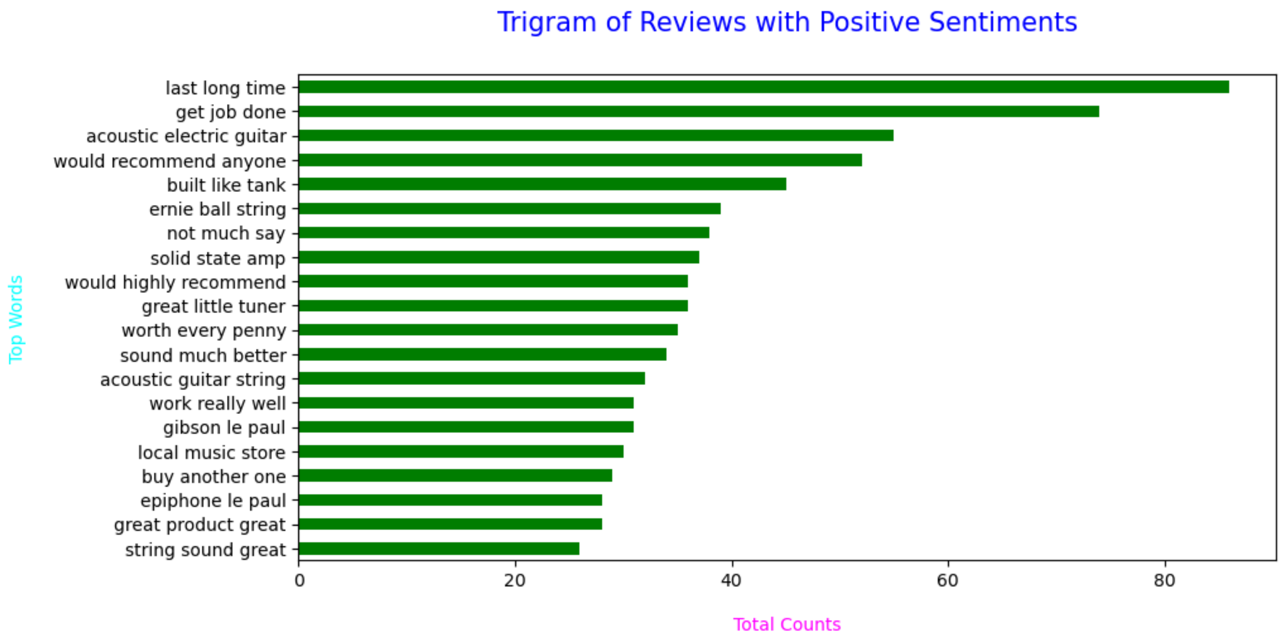
In [34]:
```
# Finding Bigram
words = Gram_Analysis(Negative["reviews"], 2, 20)
Bigram = pd.DataFrame(words, columns = ["Words", "Counts"])

# Visualization
Bigram.groupby("Words").sum()["Counts"].sort_values().plot(kind = "barh", color = "re
plt.title("Bigram of Reviews with Negative Sentiments", loc = "center", fontsize = 15
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()
```



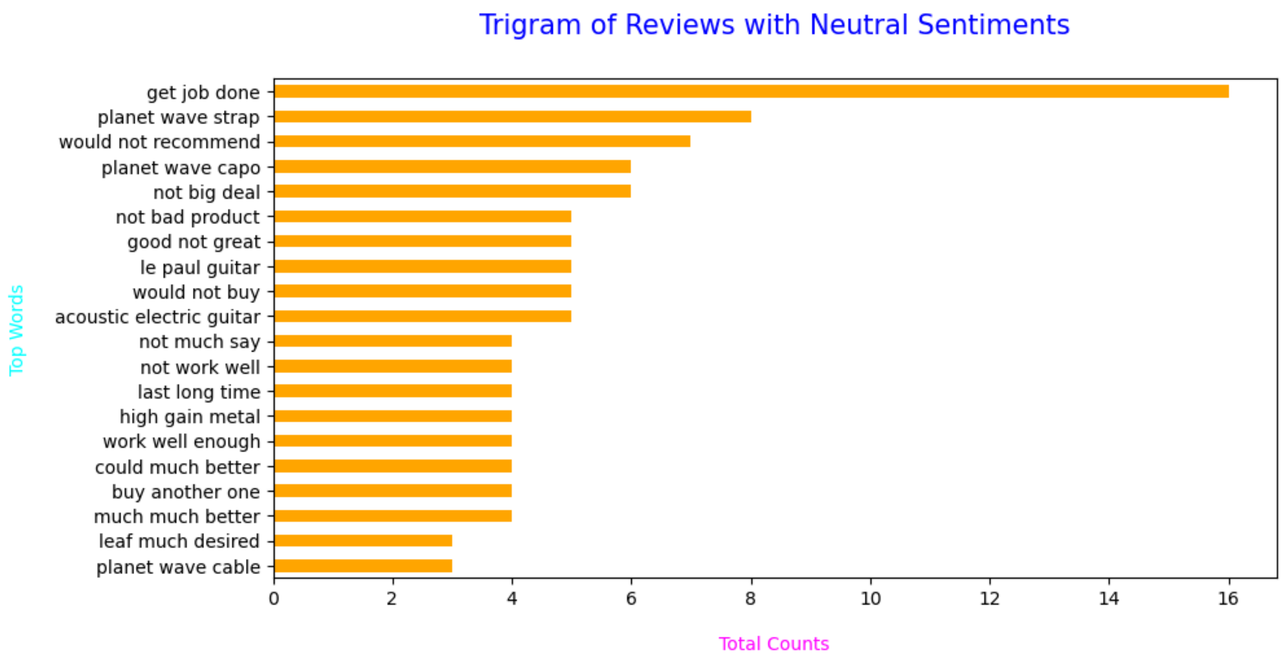
The bigrams work better than the unigrams, because we can actually see some phrases that really describe what a good sentiment is. Although, in some parts we can still see guitar objects as the top words, which make us believe that our interpretation about the most selling items are related to guitars.

Trigram of Reviews Based On Sentiments

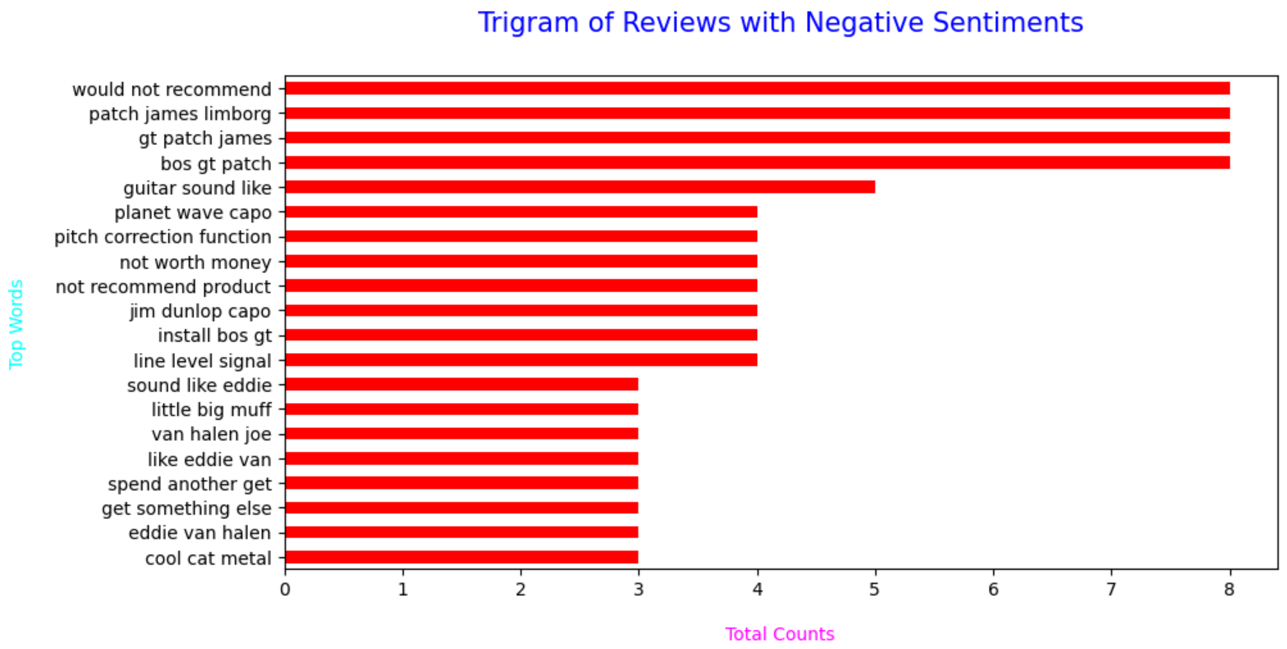
In [35]:
```python
# Finding Trigram
words = Gram_Analysis(Positive["reviews"], 3, 20)
Trigram = pd.DataFrame(words, columns = ["Words", "Counts"])

# Visualization
Trigram.groupby("Words").sum()["Counts"].sort_values().plot(kind = "barh", color = "g
plt.title("Trigram of Reviews with Positive Sentiments", loc = "center", fontsize = 1
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()
```



In [36]:
```python
# 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 = "c
plt.title("Trigram of Reviews with Neutral Sentiments", loc = "center", fontsize = 15
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()
```



In [37]:
```
# 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 = "r
plt.title("Trigram of Reviews with Negative Sentiments", loc = "center", fontsize = 1
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()
```

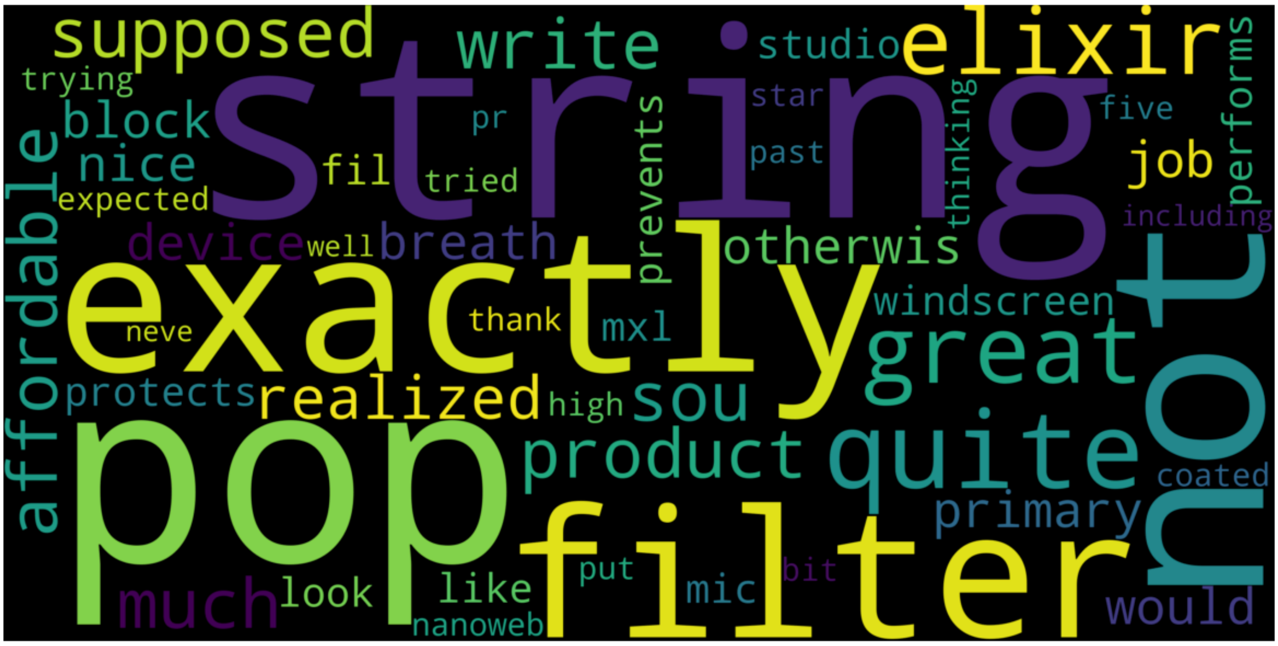


We can say that the trigrams are slightly better to describe each sentiments, although negative trigrams say a lot about bad products which we can infer from the top words above. From the N-Gram Analysis, we can also see how the decision of not removing `not` *in our list of stopwords affects our data as we keep the meaning of negation phrases.*

Word Clouds

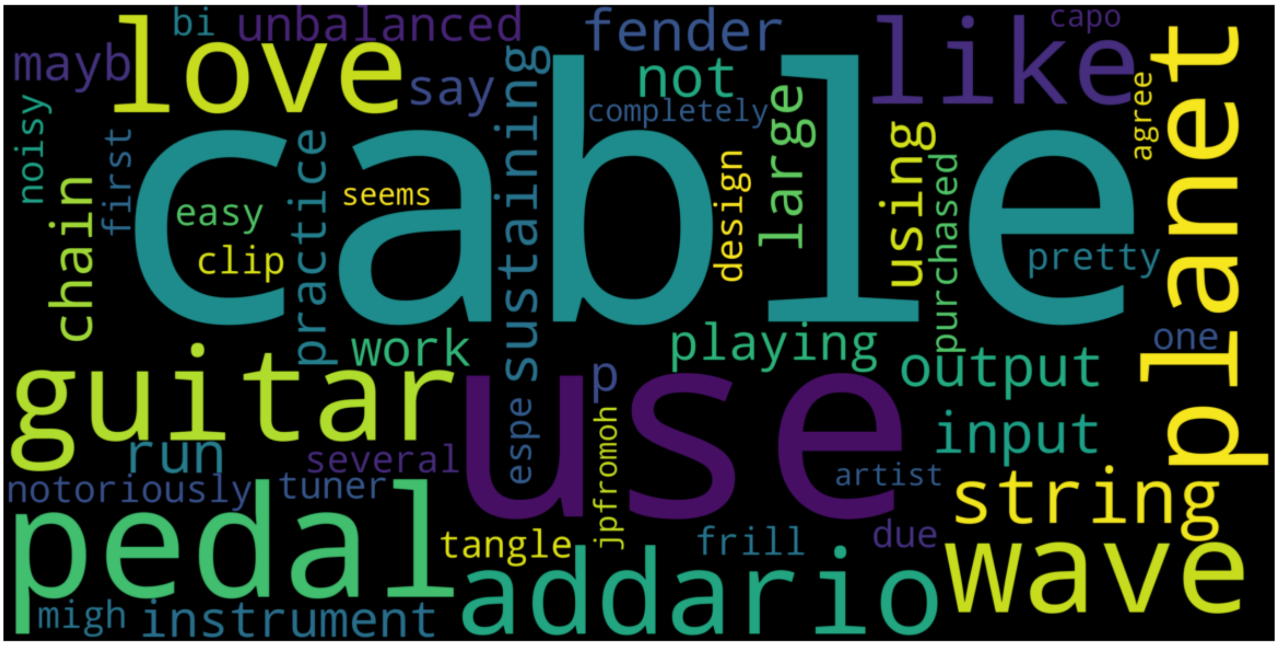
Word Cloud of Reviews with Positive Sentiments

```
In [38]: wordCloud = WordCloud(max_words = 50, width = 3000, height = 1500, stopwords = Stopwo
         plt.figure(figsize = (15, 15))
         plt.imshow(wordCloud, interpolation = "bilinear")
         plt.axis("off")
         plt.show()
```



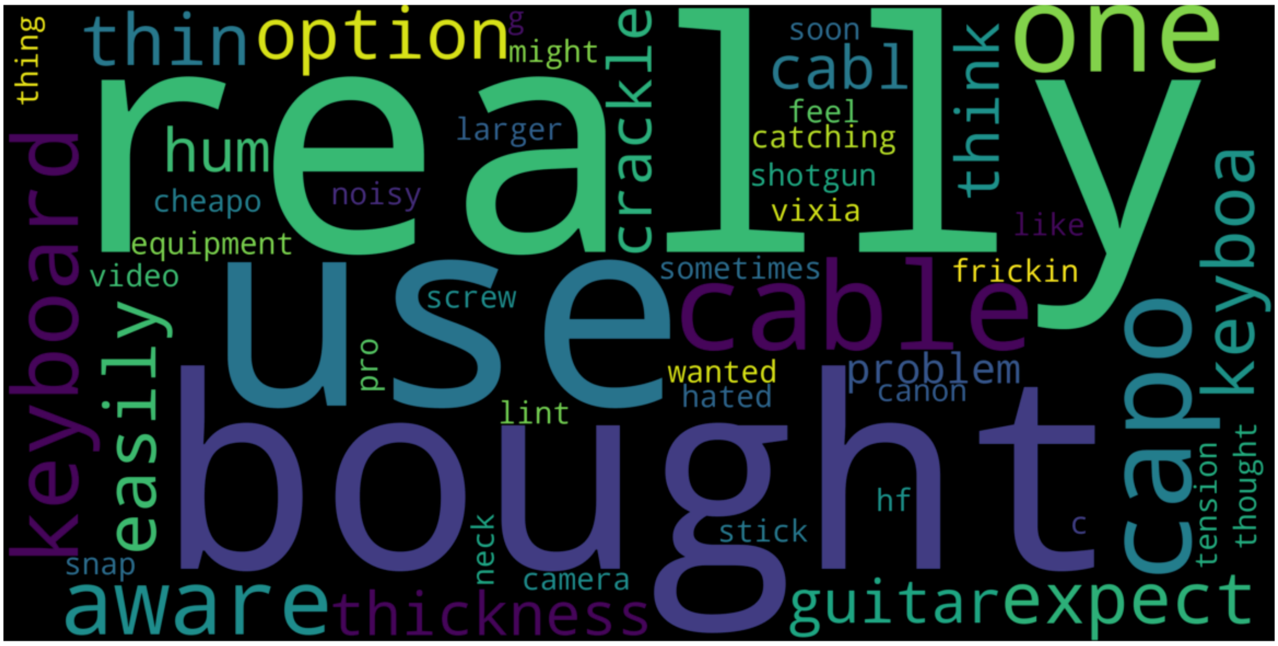
Word Cloud of Reviews with Neutral Sentiments

```
In [39]: wordCloud = WordCloud(max_words = 50, width = 3000, height = 1500, stopwords = Stopwo
         plt.figure(figsize = (15, 15))
         plt.imshow(wordCloud, interpolation = "bilinear")
         plt.axis("off")
         plt.show()
```



Word Cloud of Reviews with Negative Sentiments

```
In [40]: wordCloud = WordCloud(max_words = 50, width = 3000, height = 1500, stopwords = Stopwo
         plt.figure(figsize = (15, 15))
         plt.imshow(wordCloud, interpolation = "bilinear")
         plt.axis("off")
         plt.show()
```



From these word clouds, not only we can see words that really describe our sentiments, but just like our N-Grams Analysis we can see objects being discussed in the reviews given.

Feature Engineering

Drop Insignificant Columns

```
In [41]: Columns = ["reviewerID", "asin", "reviewerName", "helpful", "unixReviewTime", "review
         dataset.drop(columns = Columns, axis = 1, inplace = True)
```

We dropped these columns to make our dataset concise. We now have two columns as our independent variables and the last column as dependent variables. To continue, we must encode our label as a set of numbers corresponding to each categories of it.

Current State of The Dataset

In [42]:
```
dataset.head()
```

Out[42]:

	reviews	sentiment
0	not much write exactly supposed filter pop sou...	Positive
1	product exactly quite affordable not realized ...	Positive
2	primary job device block breath would otherwis...	Positive
3	nice windscreen protects mxl mic prevents pop ...	Positive
4	pop filter great look performs like studio fil...	Positive

Encoding Our Target Variable

In [43]:
```
Encoder = LabelEncoder()
dataset["sentiment"] = Encoder.fit_transform(dataset["sentiment"])
```

In [44]:
```
dataset["sentiment"].value_counts()
```

Out[44]:
```
sentiment
2    9022
1     772
0     467
Name: count, dtype: int64
```

We had successfully encoded our sentiment into numbers so that our model can easily figure it out. From above, we know that the label `Positive` is encoded into 2, `Neutral` into 1, and `Negative` into 0. Now, we have to give importance of each words in the whole review, i.e. giving them weights. We can do this by using TF-IDF (Term Frequency - Inverse Document Frequency) Vectorizer.

TF-IDF Vectorizer

In [45]:
```
# 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
```

Out[45]:
```
(10261, 5000)
```

In [46]:
```
# Declaring our target variable
y = dataset["sentiment"]
```

From the shape, we successfully transformed our reviews with TF-IDF Vectorizer of 7000 top bigram words. Now, as we know from before, our data is kind of imbalanced with very little neutral and negative values compared to positive sentiments. We need to balance our dataset before going into modelling process.

Resampling Our Dataset

There are many ways to do resampling to an imbalanced dataset, such as SMOTE and Bootstrap Method. We will use SMOTE (Synthetic Minority Oversampling Technique) that will randomly generate new replicates of our undersampling data to balance our dataset.

In [47]:
```
Counter(y)
```

Out[47]:
```
Counter({2: 9022, 1: 772, 0: 467})
```

In [48]:
```
Balancer = SMOTE(random_state = 42)
X_final, y_final = Balancer.fit_resample(X, y)
```

In [49]:
```
Counter(y_final)
```

Out[49]:
```
Counter({2: 9022, 1: 9022, 0: 9022})
```

Now our data is already balanced as we can see from the counter of each sentiment categories before and after the resampling with SMOTE.

Splitting Our Dataset

In [50]:
```
X_train, X_test, y_train, y_test = train_test_split(X_final, y_final, test_size = 0.2
```

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

Model Selection and Evaluation

We do not really know what is the best model that fits our data well. Because of that, we will need to try every classification models available and find the best models using the Confusion Matrix and F1 Score as our main metrics, and the rest of the metrics as our support. First, we should do some cross validation techniques in order to find the best model.

Model Building

We are using K-Fold Cross Validation on our early dataset (before resampling) because the CV itself is not affected by the imbalanced dataset as it splits the dataset and takes into account every validations. If we use the CV on the balanced dataset that we got from resampling we should be able to get similar result.

In [51]:
```
DTree = DecisionTreeClassifier()
LogReg = LogisticRegression()
SVC = SVC()
RForest = RandomForestClassifier()
Bayes = BernoulliNB()
KNN = KNeighborsClassifier()

Models = [DTree, LogReg, SVC, RForest, Bayes, KNN]
Models_Dict = {0: "Decision Tree", 1: "Logistic Regression", 2: "SVC", 3: "Random For

for i, model in enumerate(Models):
  print("{} Test Accuracy: {}".format(Models_Dict[i], cross_val_score(model, X, y, cv
```

```
Decision Tree Test Accuracy: 0.8224330977828647
Logistic Regression Test Accuracy: 0.8818828283518491
SVC Test Accuracy: 0.8805184008381876
Random Forest Test Accuracy: 0.8775951834579416
Naive Bayes Test Accuracy: 0.8091794454219505
K-Neighbors Test Accuracy: 0.8474810714983934
```

We got six models on our sleeves and from the results of 10-Fold Cross Validation, we know that the `Logistic Regression` model is the best model with the highest accuracy, slightly beating the `SVC`. Because of this, we will use the best model in predicting our sentiment, also to tune our parameter and evaluate the end-result of how well the model works.

Hyperparameter Tuning

In [52]:
```python
Param = {"C": np.logspace(-4, 4, 50), "penalty": ['l1', 'l2']}
grid_search = GridSearchCV(estimator = LogisticRegression(random_state = 42), param_g

grid_search.fit(X_train, y_train)
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)
```

```
D:\annaconda\Lib\site-packages\sklearn\model_selection\_validation.py:547: FitFailed
Warning:
500 fits failed out of a total of 1000.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_score
='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
500 fits failed with the following error:
Traceback (most recent call last):
  File "D:\annaconda\Lib\site-packages\sklearn\model_selection\_validation.py", line
895, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "D:\annaconda\Lib\site-packages\sklearn\base.py", line 1474, in wrapper
    return fit_method(estimator, *args, **kwargs)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "D:\annaconda\Lib\site-packages\sklearn\linear_model\_logistic.py", line 117
2, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
             ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "D:\annaconda\Lib\site-packages\sklearn\linear_model\_logistic.py", line 67,
in _check_solver
    raise ValueError(
ValueError: Solver lbfgs supports only 'l2' or None penalties, got l1 penalty.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
D:\annaconda\Lib\site-packages\sklearn\model_selection\_search.py:1051: UserWarning:
One or more of the test scores are non-finite: [       nan 0.33509045        nan 0.3
3834185        nan 0.35238192
        nan 0.38371342        nan 0.4330757         nan 0.48815209
        nan 0.54382003        nan 0.59480792        nan 0.64303666
        nan 0.68042733        nan 0.7148626         nan 0.74185886
        nan 0.76422475        nan 0.78122055        nan 0.79102436
        nan 0.79269928        nan 0.79649251        nan 0.80373423
        nan 0.81279878        nan 0.82437586        nan 0.83624824
        nan 0.85186464        nan 0.8642299         nan 0.87698908
        nan 0.88777767        nan 0.89694059        nan 0.9052664
        nan 0.91024213        nan 0.91758227        nan 0.92083365
        nan 0.92147409        nan 0.92186816        nan 0.9225086
        nan 0.92497161        nan 0.9260064         nan 0.92792754
        nan 0.92901128        nan 0.93058778        nan 0.93206559
        nan 0.93433196        nan 0.93802669        nan 0.93787888
        nan 0.94044051        nan 0.94201689        nan 0.9429529
        nan 0.94388905        nan 0.94595797        nan 0.94699248
        nan 0.94827346        nan 0.94871681]
  warnings.warn(

Best Accuracy: 94.87 %
Best Parameters: {'C': 10000.0, 'penalty': 'l2'}
```

```
D:\annaconda\Lib\site-packages\sklearn\linear_model\_logistic.py:469: ConvergenceWar
ning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-lear
n.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (h
ttps://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
```

We got a nice accuracy on our training set, which is 94.80% 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

In [53]:
```
Classifier = LogisticRegression(random_state = 42, C = 6866.488450042998, penalty = '
Classifier.fit(X_train, y_train)

Prediction = Classifier.predict(X_test)
```


```
D:\annaconda\Lib\site-packages\sklearn\linear_model\_logistic.py:469: ConvergenceWar
ning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-lear
n.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (h
ttps://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
```


Now that our model is done, we will test our model on our test set. The metrics that we will evaluate is based on this prediction that we made here.

Metrics

Accuracy On Test Set

In [54]:
```
accuracy_score(y_test, Prediction)
```

Out[54]:
```
0.9522683611644747
```

Really high accuracy that we got here, 95.21%. Still, we need to look out for the Confusion Matrix and F1 Score to find out about our model performance.

Confusion Matrix

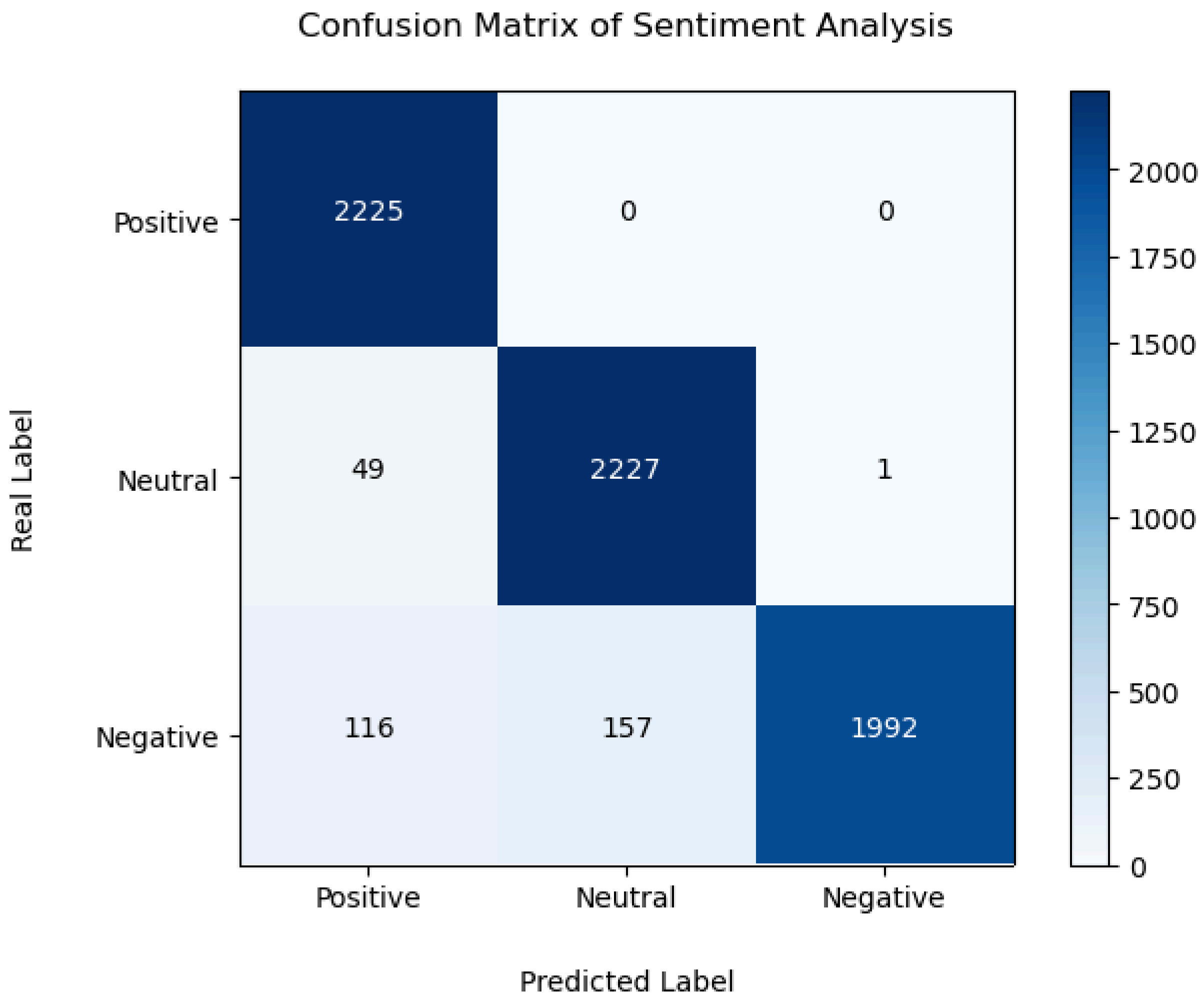
In [55]:
```
ConfusionMatrix = confusion_matrix(y_test, Prediction)
```

Visualizing Our Confusion Matrix

In [56]:
```
# Plotting Function for Confusion Matrix
def plot_cm(cm, classes, title, normalized = False, cmap = plt.cm.Blues):

  plt.imshow(cm, interpolation = "nearest", cmap = cmap)
  plt.title(title, pad = 20)
  plt.colorbar()
  tick_marks = np.arange(len(classes))
  plt.xticks(tick_marks, classes)
  plt.yticks(tick_marks, classes)

  if normalized:
    cm = cm.astype('float') / cm.sum(axis = 1)[: np.newaxis]
    print("Normalized Confusion Matrix")
  else:
    print("Unnormalized Confusion Matrix")

  threshold = cm.max() / 2
  for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
      plt.text(j, i, cm[i, j], horizontalalignment = "center", color = "white" if cm[

  plt.tight_layout()
  plt.xlabel("Predicted Label", labelpad = 20)
  plt.ylabel("Real Label", labelpad = 20)
```

In [57]:
```
plot_cm(ConfusionMatrix, classes = ["Positive", "Neutral", "Negative"], title = "Conf
```

```
Unnormalized Confusion Matrix
```



What we can gain from the Confusion Matrix above is that the model overall works well. It is able to categorize both positive and neutral sentiments correctly, while it seems to struggle a bit at determining negative sentiments. Of course, this is the effect of imbalanced data that we got from our original dataset, and luckily we can minimize the effect thanks to our SMOTE resampling before

Classification Scores

In [58]:
```
print(classification_report(y_test, Prediction))
```

```
              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, to each of our sentiment categories, we got F1 Score of 95%, which is great and because of that we can conclude that our model works well on the dataset.

Conclusion

Dataset

1. Our dataset contains many features about user reviews on musical instruments. But, we rarely need those features as our model variables because those features are not really important for sentiment analysis.
2. We might need to omit our part of removing stopwords in our preprocessing phase, because there might be some important words in determining user sentiments in our model.
3. From our text analysis, we know that most of the transactions made are related to guitars or other string-based instruments. We can say that guitar got a really high attention from the customers' pool and the sellers can emphasize their products on this instruments.

Model

1. We tried almost all classification models available. By using 10-Fold Cross Validation, we get that Logistic Regression Model got the best accuracy and we decided to use this model and tune it.
2. On our attempt on making prediction to our test set, we also received a nice accuracy and high F1 Score. This means that our model works well on sentiment analysis.
3. We need to consider more Cross Validation Method, such as Stratified K-Fold so that we do not really need to do resampling on our dataset. Also, we are fine without data scaling, but it is highly suggested to do it.

Sources of Learning

These articles and notebooks are great and really useful for sentiment analysis and NLP. Check it out!

1. Text Preprocessing in Python: Steps, Tools, and Examples (https://medium.com/@datamonsters/text-preprocessing-in-python-steps-tools-and-examples-bf025f872908)
2. Sentiment Analysis — ML project from Scratch to Production (Web Application) (https://medium.com/towards-artificial-intelligence/sentiment-analysis-from-scratch-to-production-web-api-3382f19748e8)
3. Updated Text Preprocessing techniques for Sentiment Analysis (https://towardsdatascience.com/updated-text-preprocessing-techniques-for-sentiment-analysis-549af7fe412a)
4. Amazon Instrument: Sentimental Analysis (https://www.kaggle.com/nayansakhiya/amazon-instrument-sentimental-analysis)
5. Sentiment Analysis | Amazon reviews (https://www.kaggle.com/benroshan/sentiment-analysis-amazon-reviews#Story-Generation-and-Visualization-from-reviews)
6. SMOTE for Imbalanced Classification with Python (https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/)

Other Documentations:

1. Pandas
2. Matplotlib
3. Scikit-Learn
4. Natural Language Toolkit