sentiment-analy

June 28, 2024

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

[ ]:
```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 /root/nltk_data…
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data…
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data…
```

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

# Text Vectorizer
from sklearn.feature_extraction.text import CountVectorizer

# Word Cloud
from wordcloud import WordCloud
```

[ ]:
```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 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
```

[ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

[ ]:
```
dataset = pd.read_csv("/content/drive/MyDrive/Senti/Instruments_Reviews.csv")
```

[ ]:
```
dataset.shape
```

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

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

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

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

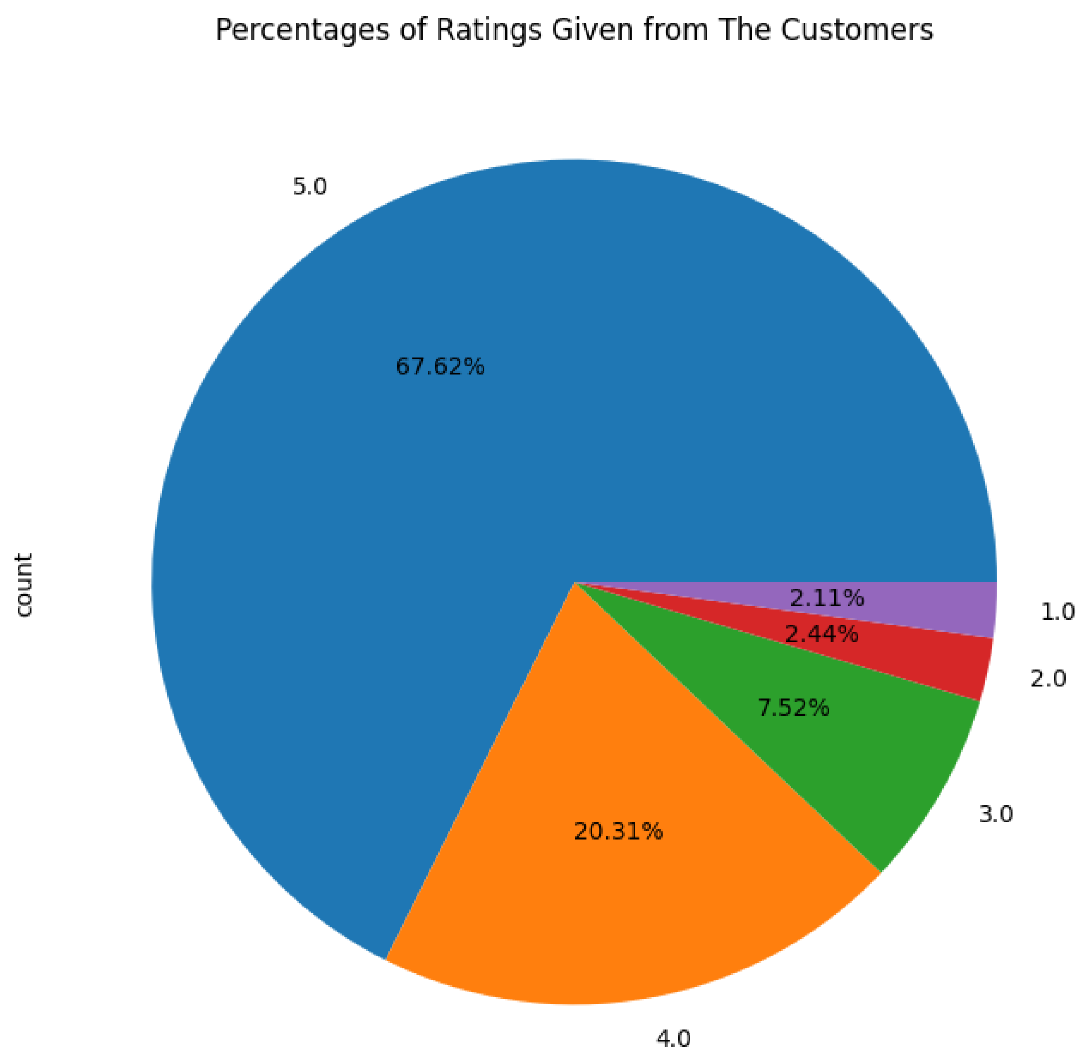
[ ]:
```python
dataset["reviews"] = dataset["reviewText"] + " " + dataset["summary"]
dataset.drop(columns = ["reviewText", "summary"], axis = 1, inplace = True)
```

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

[ ]:
```
          reviewerID        asin     reviewerName helpful       overall  \
count          10261       10261            10234   10261  10261.000000
unique          1429         900             1397     269           NaN
top     ADH0O8UVJOT10  B003VWJ2K8  Amazon Customer  [0, 0]           NaN
freq              42         163               66    6796           NaN
mean             NaN         NaN              NaN     NaN      4.488744
std              NaN         NaN              NaN     NaN      0.894642
min              NaN         NaN              NaN     NaN      1.000000
25%              NaN         NaN              NaN     NaN      4.000000
50%              NaN         NaN              NaN     NaN      5.000000
75%              NaN         NaN              NaN     NaN      5.000000
max              NaN         NaN              NaN     NaN      5.000000

        unixReviewTime   reviewTime  \
count     1.026100e+04        10261
unique             NaN         1570
top                NaN  01 22, 2013
freq               NaN           40
mean      1.360606e+09          NaN
std       3.779735e+07          NaN
min       1.095466e+09          NaN
25%       1.343434e+09          NaN
50%       1.368490e+09          NaN
75%       1.388966e+09          NaN
max       1.405987e+09          NaN

                                                  reviews
count                                               10261
unique                                              10261
top     Not much to write about here, but it does exac…
freq                                                    1
mean                                                  NaN
std                                                   NaN
min                                                   NaN
25%                                                   NaN
50%                                                   NaN
75%                                                   NaN
max                                                   NaN
```

[ ]:
```python
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")
```

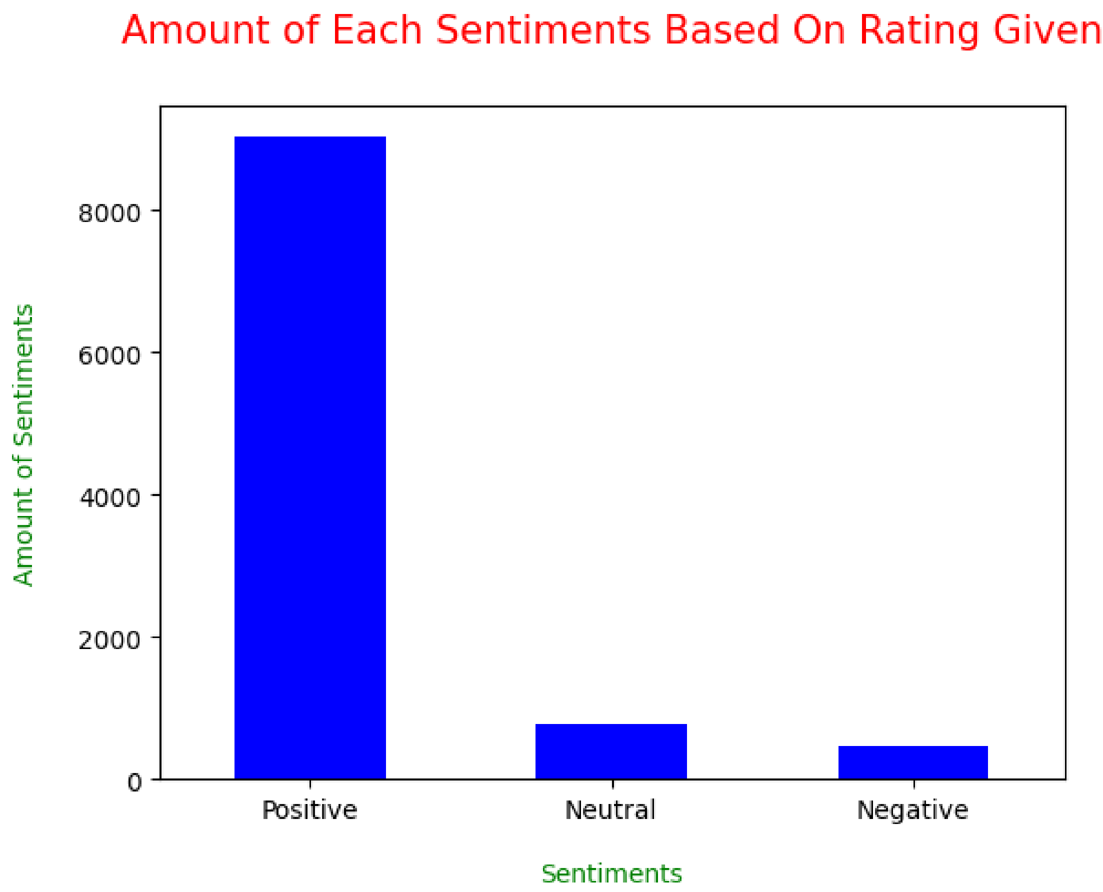
```
plt.show()
```



[ ]:
```
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("Amount of Sentiments", color = "green", fontsize = 10, labelpad =␣
  ↪15)
plt.show()
```




```
[ ]: 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
```

[ ]:
```
# 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))
```

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

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

[ ]:
```
       reviewerID        asin  \
0  A2IBPI20UZIR0U  1384719342
1  A14VAT5EAX3D9S  1384719342
2  A195EZSQDW3E21  1384719342
3  A2C00NNG1ZQQG2  1384719342
4   A94QU4C90B1AX  1384719342
5  A2A039TZMZHH9Y  B00004Y2UT
6  A1UPZM995ZAH90  B00004Y2UT
7   AJNFQI3YR6XJ5  B00004Y2UT
8  A3M1PLEYNDEY08  B00004Y2UT
9   AMNTZU1YQN1TH  B00004Y2UT
```

```
                                     reviewerName   helpful  overall  \
0  cassandra tu "Yeah, well, that's just like, u…    [0, 0]      5.0
1                                            Jake  [13, 14]      5.0
2                     Rick Bennette "Rick Bennette"    [1, 1]      5.0
3                         RustyBill "Sunday Rocker"    [0, 0]      5.0
4                                    SEAN MASLANKA    [0, 0]      5.0
5                               Bill Lewey "blewey"    [0, 0]      5.0
6                                            Brian    [0, 0]      5.0
7                                 Fender Guy "Rick"    [0, 0]      3.0
8                                  G. Thomas "Tom"    [0, 0]      5.0
9                                      Kurt Robair    [0, 0]      5.0

   unixReviewTime  reviewTime  \
0      1393545600  02 28, 2014
1      1363392000  03 16, 2013
2      1377648000  08 28, 2013
3      1392336000  02 14, 2014
4      1392940800  02 21, 2014
5      1356048000  12 21, 2012
6      1390089600  01 19, 2014
7      1353024000  11 16, 2012
8      1215302400   07 6, 2008
9      1389139200   01 8, 2014

                                             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
5  good bought another one love heavy cord gold c…  Positive
6  used monster cable year good reason lifetime w…  Positive
7  use cable run output pedal chain input fender …   Neutral
8  perfect epiphone sheraton ii monster cable wel…  Positive
9  monster make best cable lifetime warranty does…  Positive
```

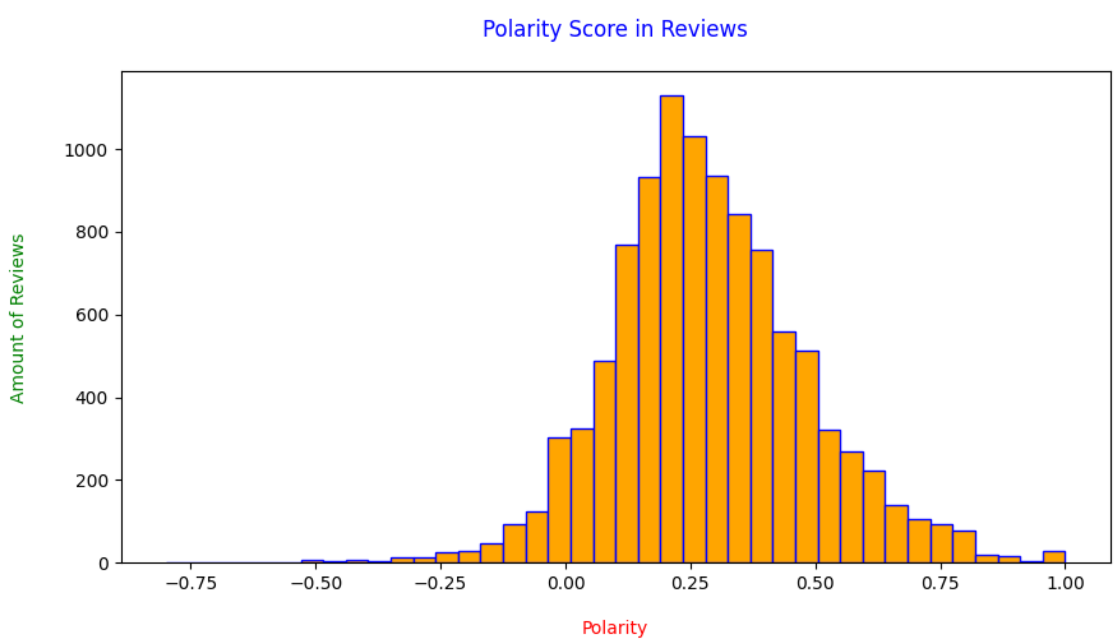
[ ]:
```
dataset.describe(include = "all")
```

[ ]:
```
           reviewerID        asin     reviewerName helpful       overall  \
count           10261       10261            10234   10261  10261.000000
unique           1429         900             1397     269           NaN
top     ADH0O8UVJOT10  B003VWJ2K8  Amazon Customer  [0, 0]           NaN
freq               42         163               66    6796           NaN
mean              NaN         NaN              NaN     NaN      4.488744
std               NaN         NaN              NaN     NaN      0.894642
min               NaN         NaN              NaN     NaN      1.000000
25%               NaN         NaN              NaN     NaN      4.000000
```

```
50%                 NaN          NaN                NaN     NaN       5.000000
75%                 NaN          NaN                NaN     NaN       5.000000
max                 NaN          NaN                NaN     NaN       5.000000

        unixReviewTime  reviewTime                reviews sentiment
count     1.026100e+04       10261                  10261     10261
unique             NaN        1570                  10254         3
top                NaN  01 22, 2013  good string five star  Positive
freq               NaN          40                      3      9022
mean      1.360606e+09         NaN                    NaN       NaN
std       3.779735e+07         NaN                    NaN       NaN
min       1.095466e+09         NaN                    NaN       NaN
25%       1.343434e+09         NaN                    NaN       NaN
50%       1.368490e+09         NaN                    NaN       NaN
75%       1.388966e+09         NaN                    NaN       NaN
max       1.405987e+09         NaN                    NaN       NaN
```

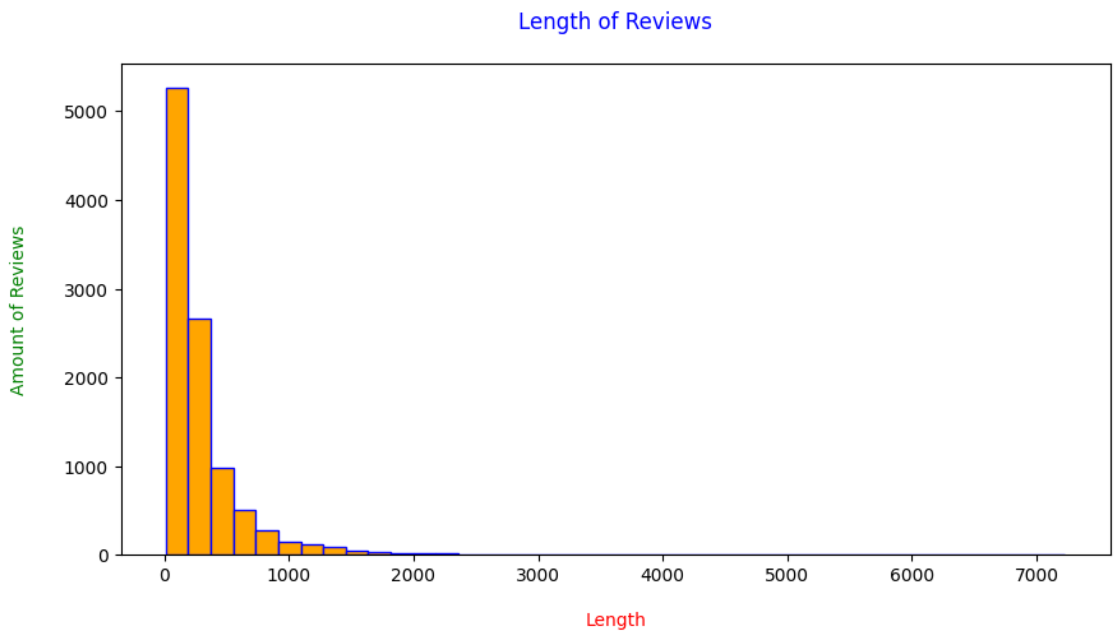
[ ]:
```
dataset["polarity"] = dataset["reviews"].map(lambda Text: TextBlob(Text).
 ↪sentiment.polarity)
```

[ ]:
```
dataset["polarity"].plot(kind = "hist", bins = 40, edgecolor = "blue",␣
 ↪linewidth = 1, color = "orange", figsize = (10,5))
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()
```



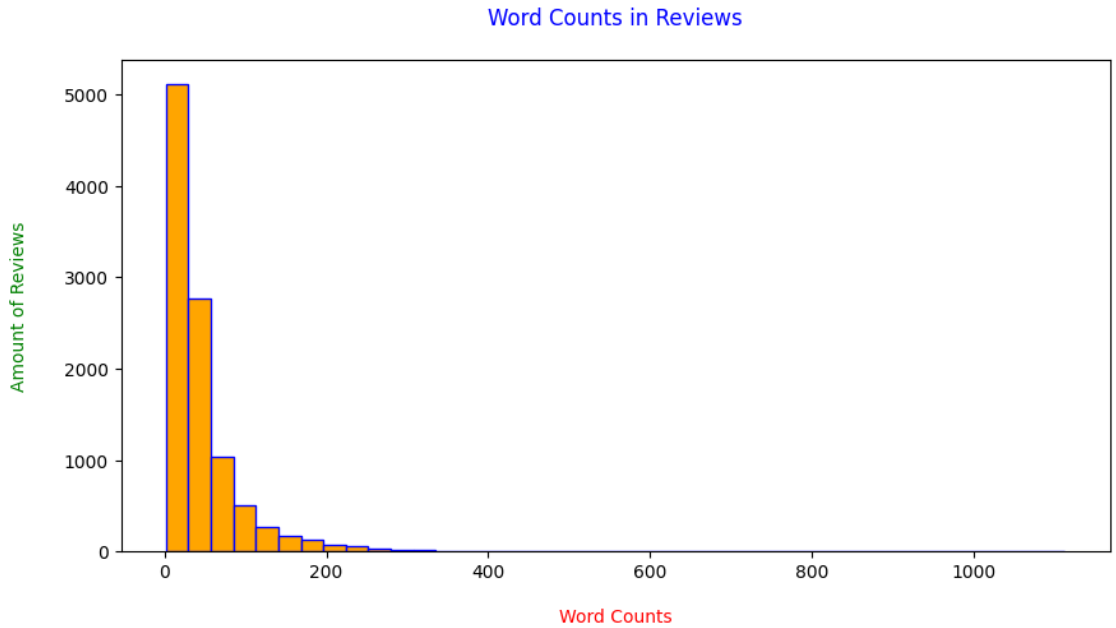
[ ]:
```
dataset["length"] = dataset["reviews"].astype(str).apply(len)
```

[ ]:
```
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()
```



[ ]:
```
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()
```

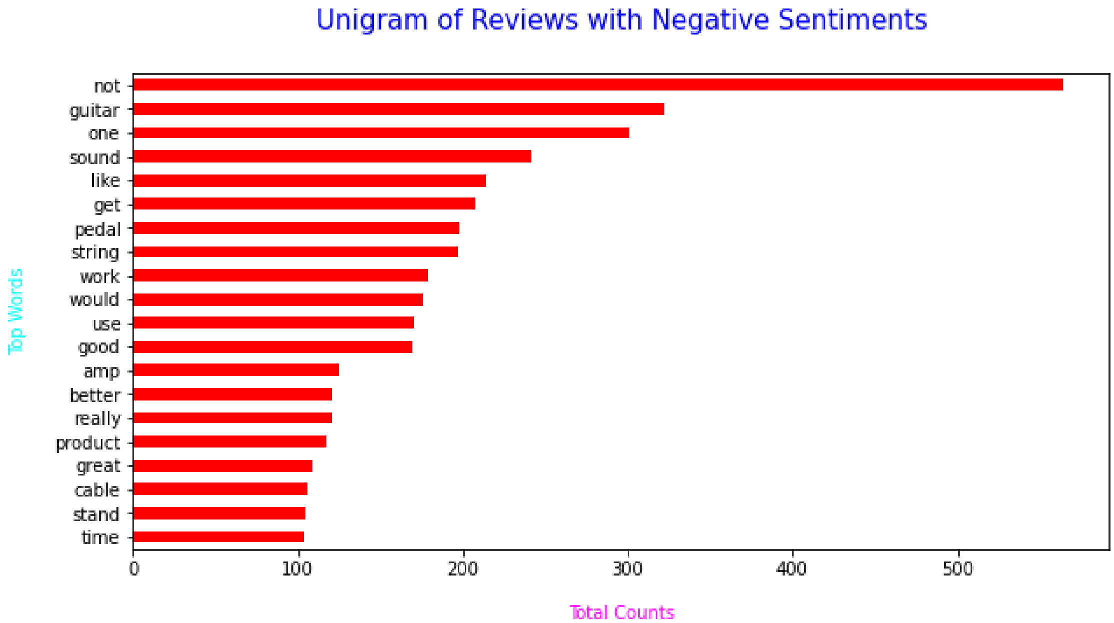



```
[ ]: def Gram_Analysis(Corpus, Gram, N):
       # Vectorizer
       Vectorizer = CountVectorizer(stop_words = Stopwords, ngram_range=(Gram,Gram))

       # 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]
```

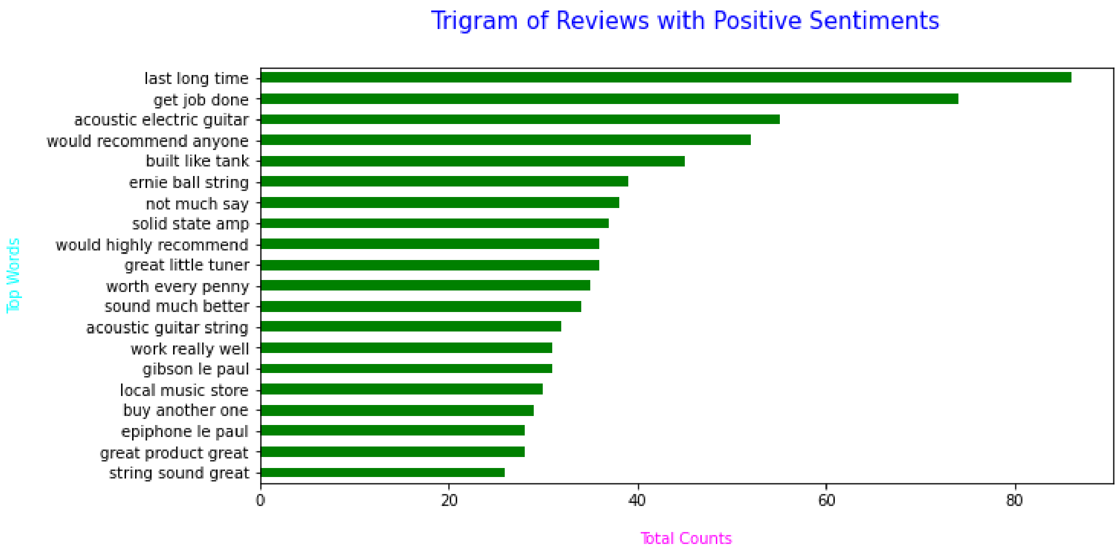
```
[ ]: # Use dropna() so the base DataFrame is not affected
     Positive = dataset[dataset["sentiment"] == "Positive"].dropna()
     Neutral = dataset[dataset["sentiment"] == "Neutral"].dropna()
     Negative = dataset[dataset["sentiment"] == "Negative"].dropna()
```

```
[ ]: # Finding Unigram
     words = Gram_Analysis(Negative["reviews"], 1, 20)
```

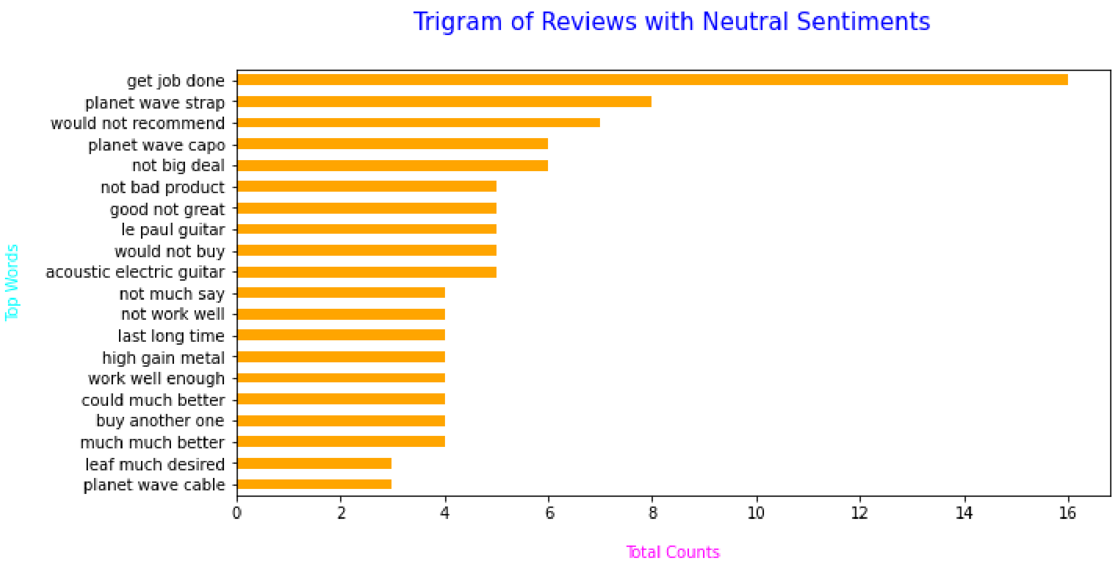
```
Unigram = pd.DataFrame(words, columns = ["Words", "Counts"])

# 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()
```



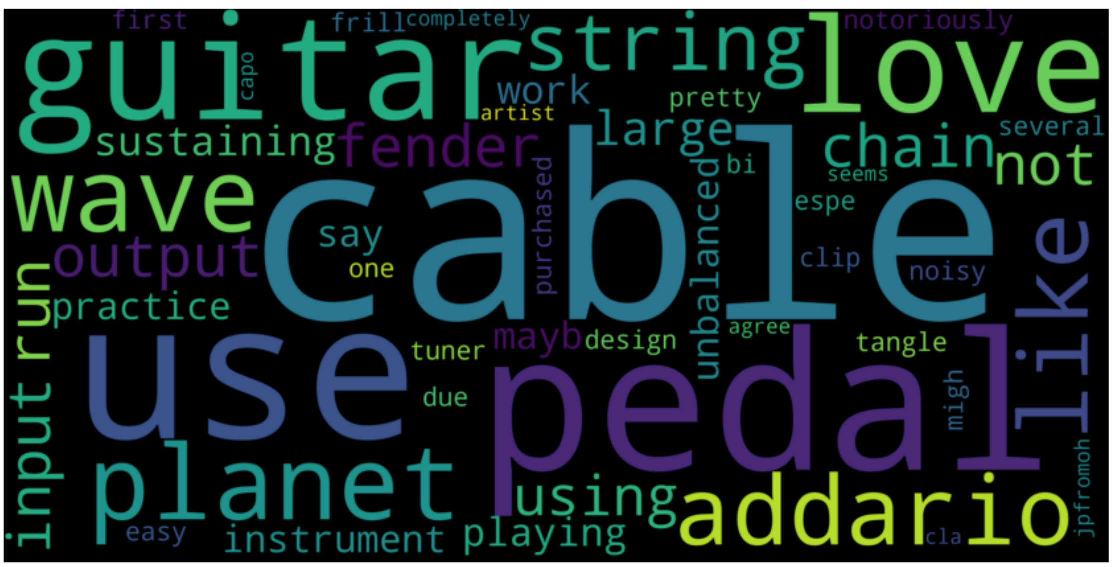
[ ]:
```
# Finding Trigram
words = Gram_Analysis(Positive["reviews"], 3, 20)
Trigram = pd.DataFrame(words, columns = ["Words", "Counts"])

# 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()
```




```
[ ]: wordCloud = WordCloud(max_words = 50, width = 3000, height = 1500, stopwords =
     ↪Stopwords).generate(str(Neutral["reviews"]))
     plt.figure(figsize = (15, 15))
     plt.imshow(wordCloud, interpolation = "bilinear")
     plt.axis("off")
     plt.show()
```



Word Cloud of Reviews with Negative Sentiments

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

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

[ ]:
```
                                             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
```

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

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

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

[ ]:
```
# 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"]
```

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

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

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

```
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:87:
FutureWarning: Function safe_indexing is deprecated; safe_indexing is deprecated
```

```
in version 0.22 and will be removed in version 0.24.
  warnings.warn(msg, category=FutureWarning)
```

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

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

[ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X_final, y_final, test_size␣
↪= 0.25, random_state = 42)
```

[ ]:
```
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 Forest", 4: "Naive Bayes", 5: "K-Neighbors"}

for i, model in enumerate(Models):
  print("{} Test Accuracy: {}".format(Models_Dict[i], cross_val_score(model, X,␣
↪y, cv = 10, scoring = "accuracy").mean()))
```

```
Decision Tree Test Accuracy: 0.8197050968869757
Logistic Regression Test Accuracy: 0.8818828283518491
SVC Test Accuracy: 0.8805184008381876
Random Forest Test Accuracy: 0.8770101983293189
Naive Bayes Test Accuracy: 0.8091794454219505
K-Neighbors Test Accuracy: 0.8474810714983934
```

Hyperparameter Tuning

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

[ ]:
```
0.9521205851928476
```

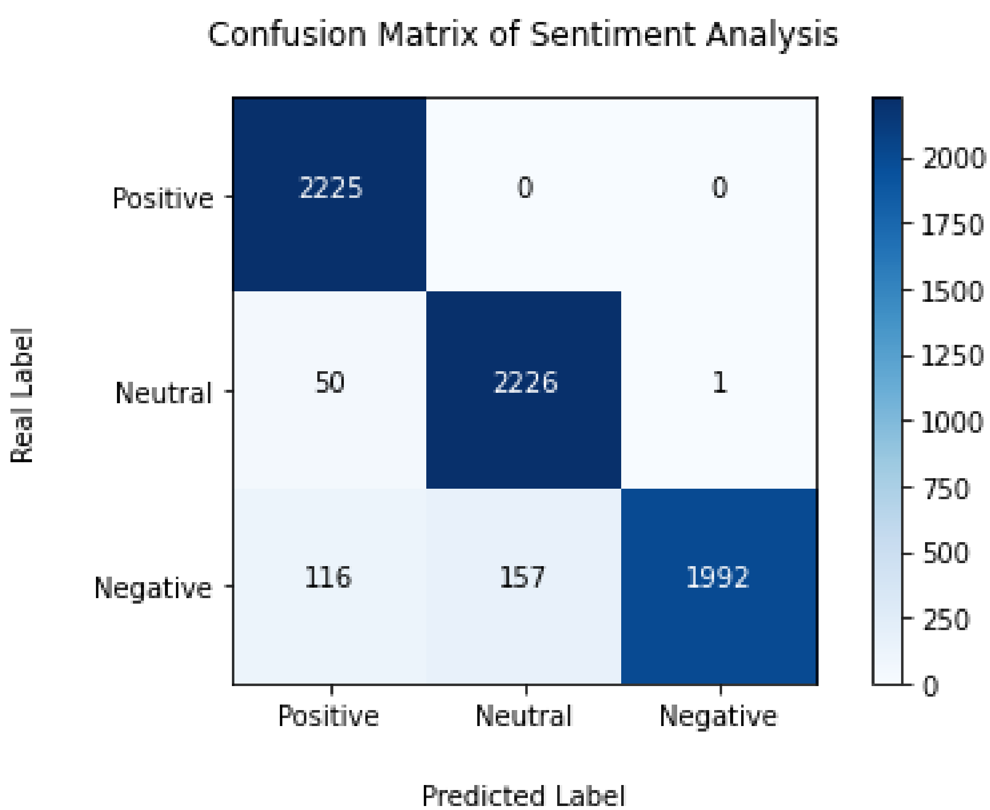
[ ]:
```
ConfusionMatrix = confusion_matrix(y_test, Prediction)
```

[ ]:
```
# 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[i, j] > threshold else "black")

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

```
[ ]: plot_cm(ConfusionMatrix, classes = ["Positive", "Neutral", "Negative"], title =␣
     ↪"Confusion Matrix of Sentiment Analysis")
```

```
Unnormalized Confusion Matrix
```



[ ]:

```python
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
```