Model Development Phase

Project Name : Amazon Instrument Reviews

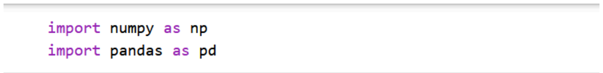
Model Selection Report

In this project, multiple machine learning models were evaluated for sentiment classification of Amazon Musical Instrument Reviews. Factors such as accuracy, confusion matrix, classification report were considered to select the most suitable model.

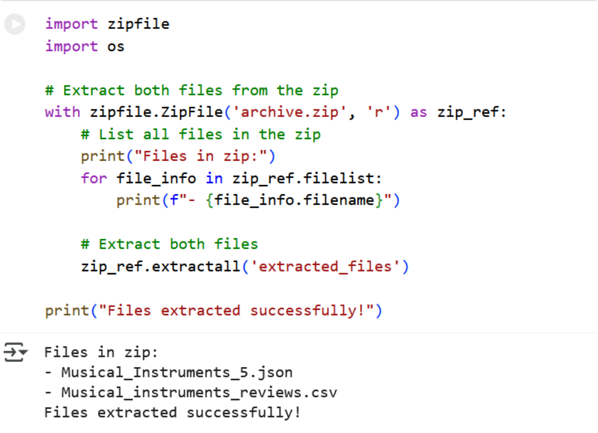
Model	Description
Model - 1 : Logistic Regression	· A simple linear model applied to TF-IDF features. · Served as a baseline for sentiment classification.
Model - 2 : Random Forest	· An ensemble learning method with multiple decision trees. · Achieved higher accuracy and robustness compared to Logistic Regression. Selected as the final model.

Model	Description

Model 1 – Training Setup & Visualizations

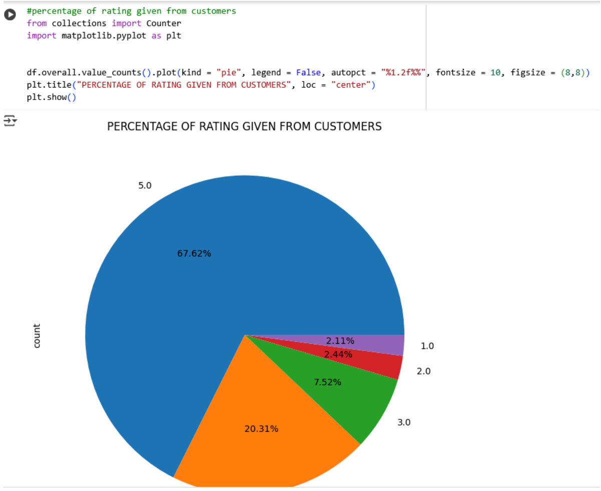


```
import numpy as np
import pandas as pd
```

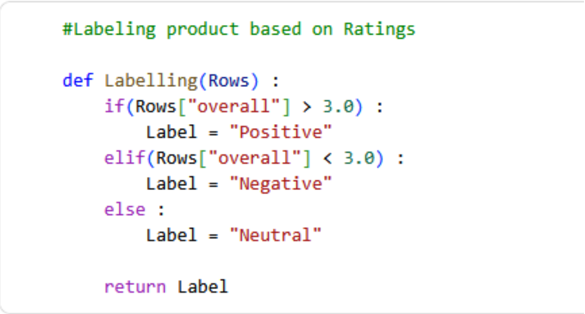


```
import zipfile
import os

# Extract both files from the zip
with zipfile.ZipFile('archive.zip', 'r') as zip_ref:
    # List all files in the zip
    print("Files in zip:")
    for file_info in zip_ref.filelist:
        print(f"- {file_info.filename}")

    # Extract both files
    zip_ref.extractall('extracted_files')

print("Files extracted successfully!")
```

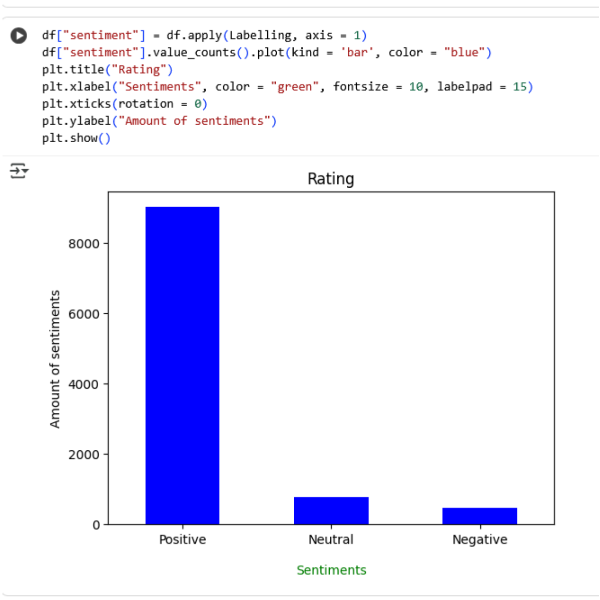
```
Files in zip:
- Musical_Instruments_5.json
- Musical_instruments_reviews.csv
Files extracted successfully!
```



```
#percentage of rating given from customers
from collections import Counter
import matplotlib.pyplot as plt


df.overall.value_counts().plot(kind = "pie", legend = False, autopct = "%1.2f%%", fontsize = 10, figsize = (8,8))
plt.title("PERCENTAGE OF RATING GIVEN FROM CUSTOMERS", loc = "center")
plt.show()
```

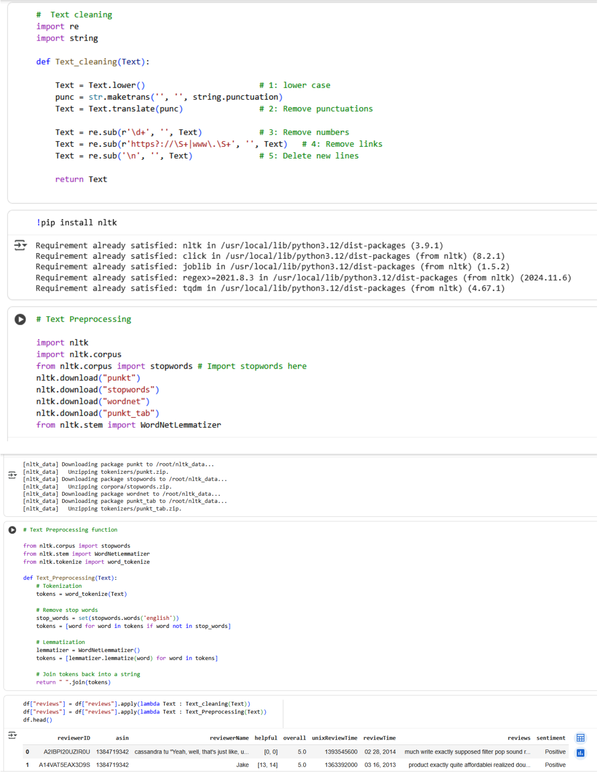


```
#Labeling product based on Ratings

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

    return Label
```



```
df["sentiment"] = df.apply(Labelling, axis = 1)
df["sentiment"].value_counts().plot(kind = 'bar', color = "blue")
plt.title("Rating")
plt.xlabel("Sentiments", color = "green", fontsize = 10, labelpad = 15)
plt.xticks(rotation = 0)
plt.ylabel("Amount of sentiments")
plt.show()
```

Model 2 – Dataset Loading & Balancing



```
#  Text cleaning
import re
import string

def Text_cleaning(Text):

    Text = Text.lower()                          # 1: lower case
    punc = str.maketrans('', '', string.punctuation)
    Text = Text.translate(punc)                  # 2: Remove punctuations

    Text = re.sub(r'\d+', '', Text)              # 3: Remove numbers
    Text = re.sub(r'https?://\S+|www\.\S+', '', Text)   # 4: Remove links
    Text = re.sub('\n', '', Text)                # 5: Delete new lines

    return Text
```

```
!pip install nltk
```

```
Requirement already satisfied: nltk in /usr/local/lib/python3.12/dist-packages (3.9.1)
Requirement already satisfied: click in /usr/local/lib/python3.12/dist-packages (from nltk) (8.2.1)
Requirement already satisfied: joblib in /usr/local/lib/python3.12/dist-packages (from nltk) (1.5.2)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.12/dist-packages (from nltk) (2024.11.6)
Requirement already satisfied: tqdm in /usr/local/lib/python3.12/dist-packages (from nltk) (4.67.1)
```

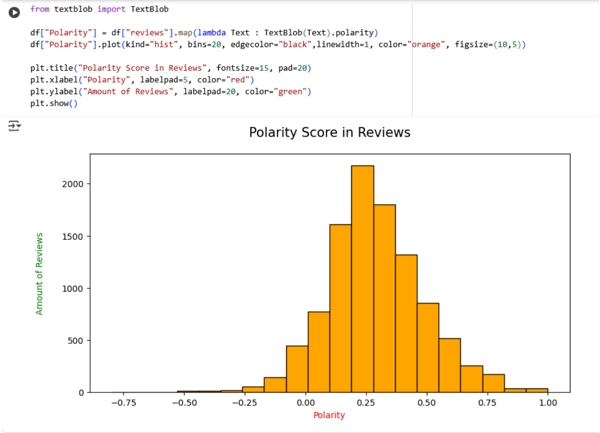
```
# Text Preprocessing

import nltk
import nltk.corpus
from nltk.corpus import stopwords # Import stopwords here
nltk.download("punkt")
nltk.download("stopwords")
nltk.download("wordnet")
nltk.download("punkt_tab")
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...
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
```

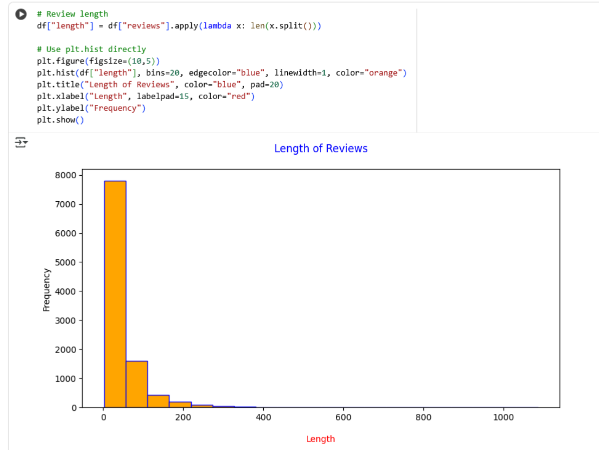
```
# Text Preprocessing function

from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize

def Text_Preprocessing(Text):
    # Tokenization
    tokens = word_tokenize(Text)

    # Remove stop words
    stop_words = set(stopwords.words('english'))
    tokens = [word for word in tokens if word not in stop_words]

    # Lemmatization
    lemmatizer = WordNetLemmatizer()
    tokens = [lemmatizer.lemmatize(word) for word in tokens]

    # Join tokens back into a string
    return " ".join(tokens)
```

```
df["reviews"] = df["reviews"].apply(lambda Text : Text_cleaning(Text))
df["reviews"] = df["reviews"].apply(lambda Text : Text_Preprocessing(Text))
df.head()
```

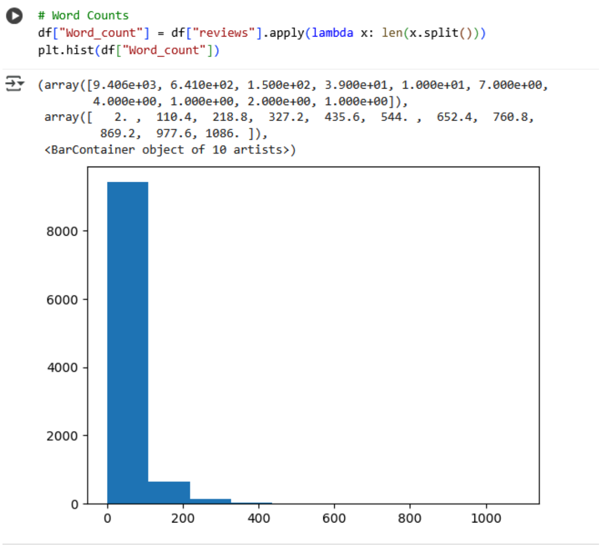
	reviewerID	asin	reviewerName	helpful	overall	unixReviewTime	reviewTime	reviews	sentiment
0	A2IBPI20UZIR0U	1384719342	cassandra tu "Yeah, well, that's just like, u...	[0, 0]	5.0	1393545600	02 28, 2014	much write exactly supposed filter pop sound r...	Positive
1	A14VAT5EAX3D9S	1384719342	Jake	[13, 14]	5.0	1363392000	03 16, 2013	product exactly quite affordablei realized dou...	Positive



```
from textblob import TextBlob

df["Polarity"] = df["reviews"].map(lambda Text : TextBlob(Text).polarity)
df["Polarity"].plot(kind="hist", bins=20, edgecolor="black",linewidth=1, color="orange", figsize=(10,5))

plt.title("Polarity Score in Reviews", fontsize=15, pad=20)
plt.xlabel("Polarity", labelpad=5, color="red")
plt.ylabel("Amount of Reviews", labelpad=20, color="green")
plt.show()
```

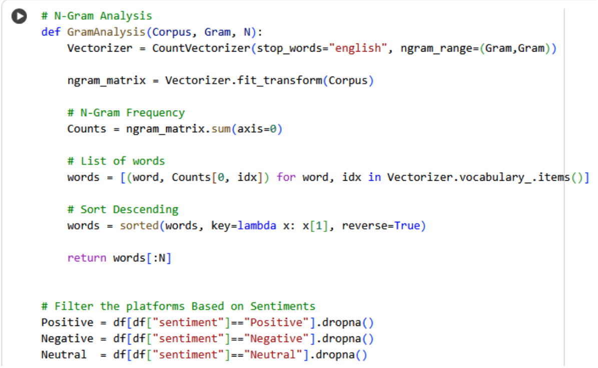


```
# Review length
df["length"] = df["reviews"].apply(lambda x: len(x.split()))

# Use plt.hist directly
plt.figure(figsize=(10,5))
plt.hist(df["length"], bins=20, edgecolor="blue", linewidth=1, color="orange")
plt.title("Length of Reviews", color="blue", pad=20)
plt.xlabel("Length", labelpad=15, color="red")
plt.ylabel("Frequency")
plt.show()
```



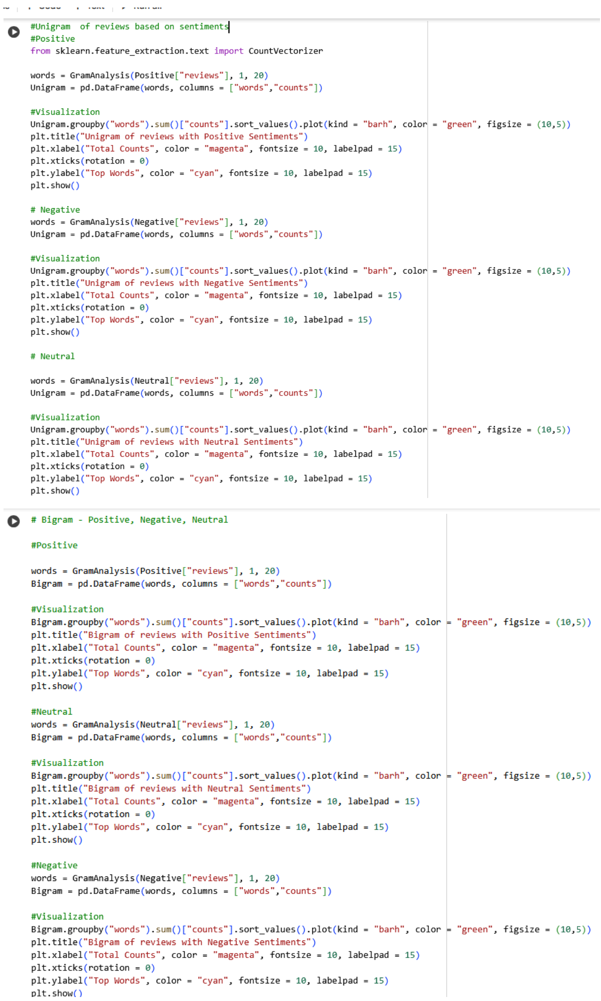
```
# Word Counts
df["Word_count"] = df["reviews"].apply(lambda x: len(x.split()))
plt.hist(df["Word_count"])
```

```
(array([9.406e+03, 6.410e+02, 1.500e+02, 3.900e+01, 1.000e+01, 7.000e+00,
        4.000e+00, 1.000e+00, 2.000e+00, 1.000e+00]),
 array([   2. ,  110.4,  218.8,  327.2,  435.6,  544. ,  652.4,  760.8,
         869.2,  977.6, 1086. ]),
 <BarContainer object of 10 artists>)
```



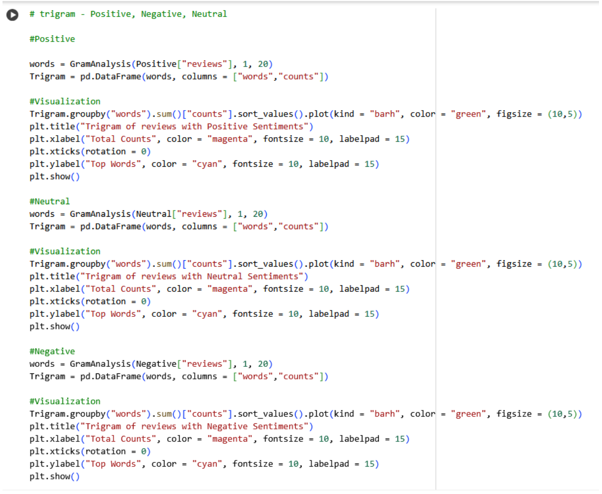
```python
# N-Gram Analysis
def GramAnalysis(Corpus, Gram, N):
    Vectorizer = CountVectorizer(stop_words="english", ngram_range=(Gram,Gram))

    ngram_matrix = Vectorizer.fit_transform(Corpus)

    # N-Gram Frequency
    Counts = ngram_matrix.sum(axis=0)

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

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

    return words[:N]


# Filter the platforms Based on Sentiments
Positive = df[df["sentiment"]=="Positive"].dropna()
Negative = df[df["sentiment"]=="Negative"].dropna()
Neutral  = df[df["sentiment"]=="Neutral"].dropna()
```

Model 3 – Feature Extraction (TF-IDF)



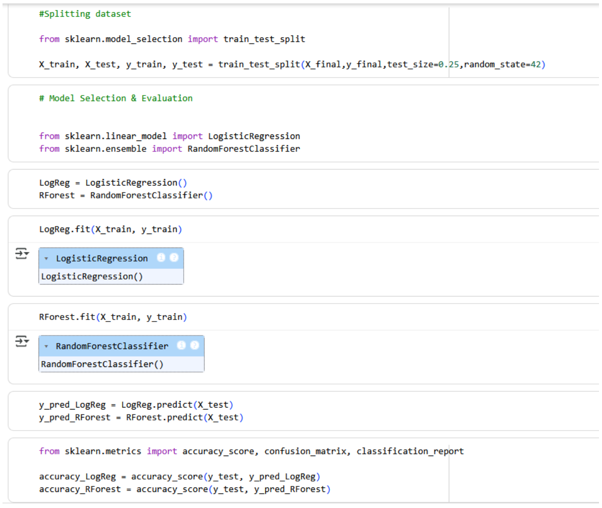
```python
#Unigram  of reviews based on sentiments
#Positive
from sklearn.feature_extraction.text import CountVectorizer

words = GramAnalysis(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")
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()

# Negative
words = GramAnalysis(Negative["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 Negative Sentiments")
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()

# Neutral

words = GramAnalysis(Neutral["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 Neutral Sentiments")
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()
```

```python
# Bigram - Positive, Negative, Neutral

#Positive

words = GramAnalysis(Positive["reviews"], 1, 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")
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()

#Neutral
words = GramAnalysis(Neutral["reviews"], 1, 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 Neutral Sentiments")
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()

#Negative
words = GramAnalysis(Negative["reviews"], 1, 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 Negative Sentiments")
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()
```



```
# trigram - Positive, Negative, Neutral

#Positive

words = GramAnalysis(Positive["reviews"], 1, 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")
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()

#Neutral
words = GramAnalysis(Neutral["reviews"], 1, 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 Neutral Sentiments")
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()

#Negative
words = GramAnalysis(Negative["reviews"], 1, 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 Negative Sentiments")
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()
```

Model 4 – Training the Models



```
#Splitting dataset

from sklearn.model_selection import train_test_split

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

```
# Model Selection & Evaluation


from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
```

```
LogReg = LogisticRegression()
RForest = RandomForestClassifier()
```

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

```
LogisticRegression
LogisticRegression()
```

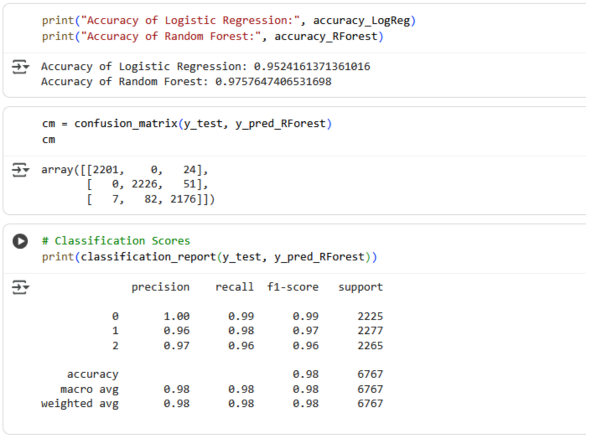
```
RForest.fit(X_train, y_train)
```

```
RandomForestClassifier
RandomForestClassifier()
```

```
y_pred_LogReg = LogReg.predict(X_test)
y_pred_RForest = RForest.predict(X_test)
```

```
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

accuracy_LogReg = accuracy_score(y_test, y_pred_LogReg)
accuracy_RForest = accuracy_score(y_test, y_pred_RForest)
```

Model 5 – Evaluation on Test Set



```
print("Accuracy of Logistic Regression:", accuracy_LogReg)
print("Accuracy of Random Forest:", accuracy_RForest)
```

```
Accuracy of Logistic Regression: 0.9524161371361016
Accuracy of Random Forest: 0.9757647406531698
```

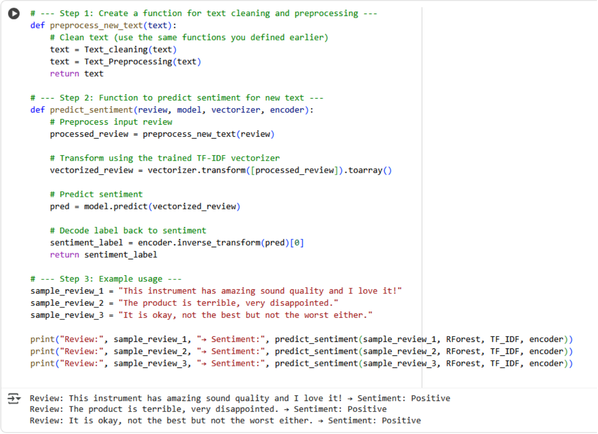
```
cm = confusion_matrix(y_test, y_pred_RForest)
cm
```

```
array([[2201,    0,   24],
       [   0, 2226,   51],
       [   7,   82, 2176]])
```

```
# Classification Scores
print(classification_report(y_test, y_pred_RForest))
```

```
              precision    recall  f1-score   support

           0       1.00      0.99      0.99      2225
           1       0.96      0.98      0.97      2277
           2       0.97      0.96      0.96      2265

    accuracy                           0.98      6767
   macro avg       0.98      0.98      0.98      6767
weighted avg       0.98      0.98      0.98      6767
```

Model 6 – Individual Review Predictions



```python
# --- Step 1: Create a function for text cleaning and preprocessing ---
def preprocess_new_text(text):
    # Clean text (use the same functions you defined earlier)
    text = Text_cleaning(text)
    text = Text_Preprocessing(text)
    return text

# --- Step 2: Function to predict sentiment for new text ---
def predict_sentiment(review, model, vectorizer, encoder):
    # Preprocess input review
    processed_review = preprocess_new_text(review)

    # Transform using the trained TF-IDF vectorizer
    vectorized_review = vectorizer.transform([processed_review]).toarray()

    # Predict sentiment
    pred = model.predict(vectorized_review)

    # Decode label back to sentiment
    sentiment_label = encoder.inverse_transform(pred)[0]
    return sentiment_label

# --- Step 3: Example usage ---
sample_review_1 = "This instrument has amazing sound quality and I love it!"
sample_review_2 = "The product is terrible, very disappointed."
sample_review_3 = "It is okay, not the best but not the worst either."

print("Review:", sample_review_1, "→ Sentiment:", predict_sentiment(sample_review_1, RForest, TF_IDF, encoder))
print("Review:", sample_review_2, "→ Sentiment:", predict_sentiment(sample_review_2, RForest, TF_IDF, encoder))
print("Review:", sample_review_3, "→ Sentiment:", predict_sentiment(sample_review_3, RForest, TF_IDF, encoder))
```

```
Review: This instrument has amazing sound quality and I love it! → Sentiment: Positive
Review: The product is terrible, very disappointed. → Sentiment: Positive
Review: It is okay, not the best but not the worst either. → Sentiment: Positive
```