In [1]:
```
import pandas as pd

# Read the CSV file into a DataFrame
df = pd.read_csv('your_file.csv')
```

In [2]:
```
data2 = df.copy()
```

In [3]:
```
%%time
from sklearn.preprocessing import LabelEncoder

# Initialize the LabelEncoder
label_encoder = LabelEncoder()

# Fit and transform the 'sentiment' column
data2['sentiment_encoded'] = label_encoder.fit_transform(data2['Sentiment'])

# Display the DataFrame with the encoded sentiment column
data2.head()
```

```
CPU times: total: 453 ms
Wall time: 546 ms
```

Out[3]:

	Product Name	Brand Name	Price	Rating	Reviews	Review Votes	Sentiment	Tokenized	Without_Stopwords
0	"clear clean esn" sprint epic 4g galaxy sph-d7...	samsung	199.99	5	i feel so lucky to have found this used (phone...	1.0	positive	['i', 'feel', 'so', 'lucky', 'to', 'have', 'fo...	['feel', 'lucky', 'found', 'used', '(', 'phone...
1	"clear clean esn" sprint epic 4g galaxy sph-d7...	samsung	199.99	4	nice phone, nice up grade from my pantach revu...	0.0	positive	['nice', 'phone', ',', 'nice', 'up', 'grade', ...	['nice', 'phone', ',', 'nice', 'grade', 'panta...
2	"clear clean esn" sprint epic 4g galaxy sph-d7...	samsung	199.99	5	very pleased	0.0	positive	['very', 'pleased']	['pleased']
3	"clear clean esn" sprint epic 4g galaxy sph-d7...	samsung	199.99	4	it works good but it goes slow sometimes but i...	0.0	positive	['it', 'works', 'good', 'but', 'it', 'goes', '...	['works', 'good', 'goes', 'slow', 'sometimes',...
4	"clear clean esn" sprint epic 4g galaxy sph-d7...	samsung	199.99	4	great phone to replace my lost phone. the only...	0.0	positive	['great', 'phone', 'to', 'replace', 'my', 'los...	['great', 'phone', 'replace', 'lost', 'phone',...

In [4]:

```
# Check the distribution of the neutral , positive and negative reviews
data2.Sentiment.value_counts()
```

Out[4]:

```
Sentiment
positive    236886
negative     84902
neutral      27682
Name: count, dtype: int64
```

In [5]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 349470 entries, 0 to 349469
Data columns (total 11 columns):
 #   Column               Non-Null Count   Dtype  
---  ------               --------------   -----  
 0   Product Name         349470 non-null  object 
 1   Brand Name           294786 non-null  object 
 2   Price                349470 non-null  float64
 3   Rating               349470 non-null  int64  
 4   Reviews              349470 non-null  object 
 5   Review Votes         349470 non-null  float64
 6   Sentiment            349470 non-null  object 
 7   Tokenized            349470 non-null  object 
 8   Without_Stopwords    349470 non-null  object 
 9   Without_Punctuation  349470 non-null  object 
 10  Lemmatized           349470 non-null  object 
dtypes: float64(2), int64(1), object(8)
memory usage: 29.3+ MB
```

In [6]:

```
%%time
# Assuming data2 is your DataFrame
neg = data2.loc[data2.Sentiment == 'negative']
pos = data2.loc[data2.Sentiment == 'positive'].sample(n=len(neg), random_state=42)
# Count the number of negative reviews
nue = data2.Sentiment.value_counts()['negative']

# Select all available 'neutral' reviews if there are fewer of them than negative revi
if nue >= data2.Sentiment.value_counts().get('neutral', 0):
    neutral = data2.loc[data2.Sentiment == 'neutral']
else:
    neutral = data2.loc[data2.Sentiment == 'neutral'].sample(n=nue, random_state=42)
```

```
CPU times: total: 188 ms
Wall time: 182 ms
```

In [7]:

```
import nltk
nltk.download('wordnet')
```

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

Out[7]:

```
True
```

In [8]:

```
import nltk
from nltk.corpus import stopwords as sw
import string
import matplotlib.pyplot as plt
%matplotlib inline

from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer, TfidfVe

lemmatizer = nltk.WordNetLemmatizer()
stopwords = sw.words('english')
stopwords = stopwords + ['not_' + w for w in stopwords]

# transform punctuation to blanks
trans_punct = str.maketrans(string.punctuation,' '*len(string.punctuation))

# pad punctuation with blanks
```

```
pad_punct = str.maketrans({key: " {0} ".format(key) for key in string.punctuation})
# remove "_" from string.punctuation
invalidChars = str(string.punctuation.replace("_", ""))
```

In [9]:
```
data2.head()
```

Out[9]:

	Product Name	Brand Name	Price	Rating	Reviews	Review Votes	Sentiment	Tokenized	Without_Stopwords
0	"clear clean esn" sprint epic 4g galaxy sph-d7...	samsung	199.99	5	i feel so lucky to have found this used (phone...	1.0	positive	['i', 'feel', 'so', 'lucky', 'to', 'have', 'fo...	['feel', 'lucky', 'found', 'used', '(', 'phone...
1	"clear clean esn" sprint epic 4g galaxy sph-d7...	samsung	199.99	4	nice phone, nice up grade from my pantach revu...	0.0	positive	['nice', 'phone', ',', 'nice', 'up', 'grade', ...	['nice', 'phone', ',', 'nice', 'grade', 'panta...
2	"clear clean esn" sprint epic 4g galaxy sph-d7...	samsung	199.99	5	very pleased	0.0	positive	['very', 'pleased']	['pleased']
3	"clear clean esn" sprint epic 4g galaxy sph-d7...	samsung	199.99	4	it works good but it goes slow sometimes but i...	0.0	positive	['it', 'works', 'good', 'but', 'it', 'goes', '...	['works', 'good', 'goes', 'slow', 'sometimes',...
4	"clear clean esn" sprint epic 4g galaxy sph-d7...	samsung	199.99	4	great phone to replace my lost phone. the only...	0.0	positive	['great', 'phone', 'to', 'replace', 'my', 'los...	['great', 'phone', 'replace', 'lost', 'phone',...

In [10]:
```
from sklearn.model_selection import train_test_split

# Split the data into training (60%) and temporary data (40%)
X_train_temp, X_temp, Y_train_temp, Y_temp = train_test_split(data2['Lemmatized'], dat
```

```
# Split the temporary data into testing (50%) and validation (50%)
X_test, X_validation, Y_test, Y_validation = train_test_split(X_temp, Y_temp, test_si

print("Train:", X_train_temp.shape, Y_train_temp.shape)
print("Test:", X_test.shape, Y_test.shape)
print("Validation:", X_validation.shape, Y_validation.shape)
```

```
Train: (209682,) (209682,)
Test: (69894,) (69894,)
Validation: (69894,) (69894,)
```

In [11]:
```
#Using TF*IDF Vectorizer
```

In [12]:
```
%%time
from sklearn.feature_extraction.text import CountVectorizer

# Initialize the CountVectorizer
count_vectorizer = CountVectorizer(analyzer = 'word',ngram_range=(3,3), stop_words='er

# Fit and transform the CountVectorizer on the training data
X_train_count = count_vectorizer.fit_transform(X_train_temp)

# Transform the testing and validation data
X_test_count = count_vectorizer.transform(X_test)
X_validation_count = count_vectorizer.transform(X_validation)

# Print the shapes of the CountVectorized data
print("Train CountVectorized:", X_train_count.shape)
print("Test CountVectorized:", X_test_count.shape)
print("Validation CountVectorized:", X_validation_count.shape)
```

```
Train CountVectorized: (209682, 1864593)
Test CountVectorized: (69894, 1864593)
Validation CountVectorized: (69894, 1864593)
CPU times: total: 11.4 s
Wall time: 15.4 s
```

In [13]:
```
##model
```

In [14]:
```
%%time
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Create a Logistic Regression model
logistic_reg = LogisticRegression(random_state=30)

# Train the model on the training data
logistic_reg.fit(X_train_count, Y_train_temp)

# Predictions on the testing and validation data
Y_test_pred = logistic_reg.predict(X_test_count)
Y_validation_pred = logistic_reg.predict(X_validation_count)

# Calculate evaluation metrics for the testing set
accuracy_test = accuracy_score(Y_test, Y_test_pred)
precision_test = precision_score(Y_test, Y_test_pred, average='weighted')
recall_test = recall_score(Y_test, Y_test_pred, average='weighted')
f1_score_test = f1_score(Y_test, Y_test_pred, average='weighted')

# Calculate evaluation metrics for the validation set
```

```
accuracy_validation = accuracy_score(Y_validation, Y_validation_pred)
precision_validation = precision_score(Y_validation, Y_validation_pred, average='weigh
recall_validation = recall_score(Y_validation, Y_validation_pred, average='weighted')
f1_score_validation = f1_score(Y_validation, Y_validation_pred, average='weighted')

# Print the evaluation metrics
print("Testing Set Metrics:")
print("Accuracy:", accuracy_test)
print("Precision:", precision_test)
print("Recall:", recall_test)
print("F1 Score:", f1_score_test)

print("\nValidation Set Metrics:")
print("Accuracy:", accuracy_validation)
print("Precision:", precision_validation)
print("Recall:", recall_validation)
print("F1 Score:", f1_score_validation)
```

```
C:\Users\SACHIN\anaconda3\Lib\site-packages\sklearn\linear_model\_logistic.py:460: Co
nvergenceWarning: 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
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Testing Set Metrics:
Accuracy: 0.8522619967379175
Precision: 0.8687038083379822
Recall: 0.8522619967379175
F1 Score: 0.8379590326220326

Validation Set Metrics:
Accuracy: 0.8522763041176639
Precision: 0.8680748972094098
Recall: 0.8522763041176639
F1 Score: 0.8382678722065365
CPU times: total: 2min 39s
Wall time: 1min 29s
```

In [15]:

```
%%time
from sklearn.ensemble import RandomForestClassifier

# Create a Random Forest model
random_forest = RandomForestClassifier(random_state=30)

# Train the model on the training data
random_forest.fit(X_train_count, Y_train_temp)

# Predictions on the testing and validation data
Y_test_pred_rf = random_forest.predict(X_test_count)
Y_validation_pred_rf = random_forest.predict(X_validation_count)

# Calculate evaluation metrics for the testing set
accuracy_test_rf = accuracy_score(Y_test, Y_test_pred_rf)
precision_test_rf = precision_score(Y_test, Y_test_pred_rf, average='weighted')
recall_test_rf = recall_score(Y_test, Y_test_pred_rf, average='weighted')
f1_score_test_rf = f1_score(Y_test, Y_test_pred_rf, average='weighted')
```

```
# Calculate evaluation metrics for the validation set
accuracy_validation_rf = accuracy_score(Y_validation, Y_validation_pred_rf)
precision_validation_rf = precision_score(Y_validation, Y_validation_pred_rf, average=
recall_validation_rf = recall_score(Y_validation, Y_validation_pred_rf, average='weigh
f1_score_validation_rf = f1_score(Y_validation, Y_validation_pred_rf, average='weighte

# Print the evaluation metrics for Random Forest
print("Random Forest Testing Set Metrics:")
print("Accuracy:", accuracy_test_rf)
print("Precision:", precision_test_rf)
print("Recall:", recall_test_rf)
print("F1 Score:", f1_score_test_rf)

print("\nRandom Forest Validation Set Metrics:")
print("Accuracy:", accuracy_validation_rf)
print("Precision:", precision_validation_rf)
print("Recall:", recall_validation_rf)
print("F1 Score:", f1_score_validation_rf)
```

```
Random Forest Testing Set Metrics:
Accuracy: 0.8573840386871548
Precision: 0.8640368430052175
Recall: 0.8573840386871548
F1 Score: 0.8463574699218742

Random Forest Validation Set Metrics:
Accuracy: 0.8570692763327324
Precision: 0.8634292149595955
Recall: 0.8570692763327324
F1 Score: 0.8464514202100215
CPU times: total: 6h 50min 24s
Wall time: 8h 57min 24s
```

In [16]:

```
%%time
from sklearn.svm import SVC

# Create an SVM model
svm = SVC(random_state=30)

# Train the model on the training data
svm.fit(X_train_count, Y_train_temp)

# Predictions on the testing and validation data
Y_test_pred_svm = svm.predict(X_test_count)
Y_validation_pred_svm = svm.predict(X_validation_count)

# Calculate evaluation metrics for the testing set
accuracy_test_svm = accuracy_score(Y_test, Y_test_pred_svm)
precision_test_svm = precision_score(Y_test, Y_test_pred_svm, average='weighted')
recall_test_svm = recall_score(Y_test, Y_test_pred_svm, average='weighted')
f1_score_test_svm = f1_score(Y_test, Y_test_pred_svm, average='weighted')

# Calculate evaluation metrics for the validation set
accuracy_validation_svm = accuracy_score(Y_validation, Y_validation_pred_svm)
precision_validation_svm = precision_score(Y_validation, Y_validation_pred_svm, averag
recall_validation_svm = recall_score(Y_validation, Y_validation_pred_svm, average='wei
f1_score_validation_svm = f1_score(Y_validation, Y_validation_pred_svm, average='weigh

# Print the evaluation metrics for SVM
print("SVM Testing Set Metrics:")
```

```
print("Accuracy:", accuracy_test_svm)
print("Precision:", precision_test_svm)
print("Recall:", recall_test_svm)
print("F1 Score:", f1_score_test_svm)

print("\nSVM Validation Set Metrics:")
print("Accuracy:", accuracy_validation_svm)
print("Precision:", precision_validation_svm)
print("Recall:", recall_validation_svm)
print("F1 Score:", f1_score_validation_svm)
```

```
SVM Testing Set Metrics:
Accuracy: 0.814275903511031
Precision: 0.8467155320380438
Recall: 0.814275903511031
F1 Score: 0.7878158008900779

SVM Validation Set Metrics:
Accuracy: 0.8151486536755659
Precision: 0.8465403349184478
Recall: 0.8151486536755659
F1 Score: 0.7895114262489186
CPU times: total: 4h 49min 30s
Wall time: 6h 17min 6s
```

In [17]:

```
%%time
from sklearn.naive_bayes import MultinomialNB

# Create a Naive Bayes model
naive_bayes = MultinomialNB()

# Train the model on the training data
naive_bayes.fit(X_train_count, Y_train_temp)

# Predictions on the testing and validation data
Y_test_pred_nb = naive_bayes.predict(X_test_count)
Y_validation_pred_nb = naive_bayes.predict(X_validation_count)

# Calculate evaluation metrics for the testing set
accuracy_test_nb = accuracy_score(Y_test, Y_test_pred_nb)
precision_test_nb = precision_score(Y_test, Y_test_pred_nb, average='weighted')
recall_test_nb = recall_score(Y_test, Y_test_pred_nb, average='weighted')
f1_score_test_nb = f1_score(Y_test, Y_test_pred_nb, average='weighted')

# Calculate evaluation metrics for the validation set
accuracy_validation_nb = accuracy_score(Y_validation, Y_validation_pred_nb)
precision_validation_nb = precision_score(Y_validation, Y_validation_pred_nb, average=
recall_validation_nb = recall_score(Y_validation, Y_validation_pred_nb, average='weigh
f1_score_validation_nb = f1_score(Y_validation, Y_validation_pred_nb, average='weighte

# Print the evaluation metrics for Naive Bayes
print("Naive Bayes Testing Set Metrics:")
print("Accuracy:", accuracy_test_nb)
print("Precision:", precision_test_nb)
print("Recall:", recall_test_nb)
print("F1 Score:", f1_score_test_nb)

print("\nNaive Bayes Validation Set Metrics:")
print("Accuracy:", accuracy_validation_nb)
print("Precision:", precision_validation_nb)
```

```
print("Recall:", recall_validation_nb)
print("F1 Score:", f1_score_validation_nb)
```

```
Naive Bayes Testing Set Metrics:
Accuracy: 0.8729361604715712
Precision: 0.8803426148158386
Recall: 0.8729361604715712
F1 Score: 0.8621802110842035

Naive Bayes Validation Set Metrics:
Accuracy: 0.8731221564082754
Precision: 0.879750262448651
Recall: 0.8731221564082754
F1 Score: 0.8628237745395287
CPU times: total: 359 ms
Wall time: 384 ms
```

In [ ]: