

In [4]: `import pandas as pd`

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

In [5]: `data2 = df.copy()`

In [6]: `%%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: 15.6 ms

Wall time: 51.1 ms

	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 [8]: 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 [9]: %%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 reviews
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: 62.5 ms
Wall time: 181 ms
```

```
In [10]: 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[10]: True
```

```
In [11]: 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, TfidfVectorizer

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 [12]: data2.head()

Out[12]:

	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', 'found', 'this', 'used', 'phone...']	['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', 'up', 'grade', 'from', 'my', 'pantach', 'revu...']	['nice', 'phone', 'up', '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', 'slow', 'sometimes', 'but', 'i...']	['works', 'good', 'goes', 'slow', 'sometimes', 'but', 'i...']
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', 'lost', 'phone', 'the', 'only...']	['great', 'phone', 'replace', 'lost', 'phone', 'the', 'only...']

In [13]: 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'], data2['Sentiment'],
```

```
# 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_size=0.5)

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 [14]: *#Using TF*IDF Vectorizer*

```
In [15]: %%time
from sklearn.feature_extraction.text import CountVectorizer

# Initialize the CountVectorizer
count_vectorizer = CountVectorizer(analyzer = 'word', ngram_range=(2,2), stop_words='english')

# 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, 865080)
Test CountVectorized: (69894, 865080)
Validation CountVectorized: (69894, 865080)
CPU times: total: 8.64 s
Wall time: 11.7 s
```

In [16]: *##model*

```
In [17]: %%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='weighted')
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)

```

Testing Set Metrics:

Accuracy: 0.8937820127621827
Precision: 0.8913634745565027
Recall: 0.8937820127621827
F1 Score: 0.8866785375225699

Validation Set Metrics:

Accuracy: 0.8954130540532806
Precision: 0.8931282599173996
Recall: 0.8954130540532806
F1 Score: 0.8888735932670034
CPU times: total: 1min 41s
Wall time: 47 s

C:\Users\SACHIN\anaconda3\Lib\site-packages\sklearn\linear_model_logistic.py:460: ConvergenceWarning: 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(

```

In [18]: %%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='weighted')
recall_validation_rf = recall_score(Y_validation, Y_validation_pred_rf, average='weighted')
f1_score_validation_rf = f1_score(Y_validation, Y_validation_pred_rf, average='weighted')

# 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: 0.8946261481672246

Precision: 0.8932346097627896

Recall: 0.8946261481672246

F1 Score: 0.8892652738526636

Random Forest Validation Set Metrics:

Accuracy: 0.8951841359773371

Precision: 0.8935724950611814

Recall: 0.8951841359773371

F1 Score: 0.8903245241824945

CPU times: total: 5h 25min 41s

Wall time: 7h 9min 20s

```

In [19]: %%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, average='weighted')
recall_validation_svm = recall_score(Y_validation, Y_validation_pred_svm, average='weighted')
f1_score_validation_svm = f1_score(Y_validation, Y_validation_pred_svm, average='weighted')

# 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: 0.8678141185223338
Precision: 0.8740291479677388
Recall: 0.8678141185223338
F1 Score: 0.8537789685544649

SVM Validation Set Metrics:

Accuracy: 0.8685724096488969
Precision: 0.8746843643320363
Recall: 0.8685724096488969
F1 Score: 0.8552084367736367
CPU times: total: 4h 14min
Wall time: 5h 29min 36s

```

In [20]: %%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='weighted')
recall_validation_nb = recall_score(Y_validation, Y_validation_pred_nb, average='weighted')
f1_score_validation_nb = f1_score(Y_validation, Y_validation_pred_nb, average='weighted')

# 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: 0.8836094657624403

Precision: 0.8878195741183387

Recall: 0.8836094657624403

F1 Score: 0.8701169504281894

Naive Bayes Validation Set Metrics:

Accuracy: 0.8835379288637079

Precision: 0.8879814020562641

Recall: 0.8835379288637079

F1 Score: 0.8705085051489697

CPU times: total: 62.5 ms

Wall time: 427 ms

In []: