```
### English Poem Comprehensive Dataset (EPCD)
```

Description

The English dataset is scraped from many different web resources. It consists of 199,002 verses, each of them is labeled with one of these four meters: Iambic, Trochee, Dactyl and Anapaestic. The Iambic class dominates the dataset; they are 186,809 Iambic verses, 5418 Trochee verses, 5378 Anapaestic verses, 1397 Dactyl verses.

Steps Included :

1. Checking Of Null Values
2. Removal Of StopWords
3. Removal Of Rare Words (Optional)
4. Cleansing Of Dataset
5. Stemming Using Porter Stemmer
6. Lemmatization (if Required)
7. Use Of Word Cloud
8. Finding the Frequency Of Words
9. Finding the Frequency Of Bi-Gram Words
10. Finding the Frequency Of Tri-Gram Words
11. Adding the Review Length and Word Count
12. Adding the Polarity
13. Rating Vs Polarity
14. Removing the Neutral Ratings
15. Use Of Count Vectorizer with Logistic Regression
16. Use Of TF-IDF Vectorizer with Logistic Regression
17. Use Of Ramdom Forest Classifier
18. Model Fitting
19. Model Evaluation

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
print('setup Completed^___^')
```

```
setup Completed^___^
```

```
###! pip install --upgrade pandas
```

```
np.version.version
```

```
'1.21.6'
```

```
import pandas as pd
import numpy as np
import seaborn as sn
import matplotlib.pyplot as plt
import re
import nltk
from collections import Counter
from sklearn.feature_extraction.text import TfidfVectorizer,CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB,MultinomialNB
from sklearn.svm import SVC
from sklearn import metrics
```

```
##!mkdir ~/.kaggle
```

```
###!cp /kaggle.json ~/.kaggle/
```

```
###! pip install kaggle
```

```
###! kaggle datasets download -d mohamedkhaledelsafty/english-poem-comprehensive-dataset-apcd
```

```
##! unzip /content/english-poem-comprehensive-dataset-apcd.zip
```

```
plt.style.use('dark_background')
```

```
full_english = pd.read_csv("/content/Full English PCD.csv")
```

```
english_poem = pd.read_csv("/content/Down-sampled English PCD.csv")
```

```
print(full_english.columns, full_english.columns)
```

```
Index(['Unnamed: 0', 'Verse', 'Meter', 'char_count'], dtype='object') Index(['Unnamed: 0', 'Verse', 'Meter', 'char_count'], dtype='object')
```

Checking Of Null Values

```
train = full_english.sample(frac=0.8,random_state=0)
test = full_english.drop(train.index)
```

```
print(train.shape, test.shape)
```

```
(159202, 4) (39800, 4)
```

```
train.to_csv("/content/train.csv")
```

```
test.to_csv("/content/test.csv")
```

```
train.isnull().sum()
```

```
Unnamed: 0    0
Verse         0
Meter         0
char_count    0
dtype: int64
```

```
train.rename(columns={"Unnamed: 0" : "index"}, inplace=True)
```

```
test.rename(columns={"Unnamed: 0" : "index"}, inplace=True)
```

```
train['Verse'] = train['Verse'].astype(str)
test['Verse'] = test['Verse'].astype(str)
```

```
c = train.Meter.astype('category')

d = dict(enumerate(c.cat.categories))
```

```
print(d)
```

```
{0: 'anapaestic', 1: 'dactyl', 2: 'iambic', 3: 'trochaic'}
```

```
train['Meter'] = train.Meter.astype('category').cat.codes
```

```
test['Meter'] = test.Meter.astype('category').cat.codes
```

Cleansing The Text - Making all text lowercase, remove text in square brackets,remove links,remove punctuation

```
print('the column data types is:',train['Verse'].dtypes)
```

```
the column data types is: object
```

```
print('the column data types is:',test['Verse'].dtypes)
```

```
the column data types is: object
```

```
train['Verse'] = train['Verse'].astype(str)
test['Verse'] = test['Verse'].astype(str)
```

```
import string
import re
def clean_text(text):
    '''Make text lowercase, remove text in square brackets,remove links,remove punctuation
    and remove words containing numbers.'''
    text = text.lower()
    text = re.sub('\[.*?\]', '', text)
    text = re.sub('https?://\S+|www\.\S+', '', text)
    text = re.sub('<.*?>+', '', text)
    text = re.sub('[%s]' % re.escape(string.punctuation), '', text)
    text = re.sub('\n', '', text)
    text = re.sub('\w*\d\w*', '', text)
    return text
```

```
train['Cleaned_Verse'] = train['Verse'].apply(lambda x: clean_text(x))
```

```
test['Cleaned_Verse'] = test['Verse'].apply(lambda x: clean_text(x))
```

Removing StopWords

```
###! pip install nltk
```

```
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
stop = stopwords.words('english')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

```
train['Cleaned_Verse'] = train['Cleaned_Verse'].apply(lambda x: " ".join(x for x in x.split() if x not in stop))
train['Cleaned_Verse'].head()
```

```
69602                        untainted vice art
133404                stung envy spleen diseasd
123426                     plenty wantond cheek
28956     deck foster son fit need daring toil
```

```
30517          blushing sisterb saw pace along
Name: Cleaned_Verse, dtype: object
```

```
test['Cleaned_Verse'] = test['Cleaned_Verse'].apply(lambda x: " ".join(x for x in x.split() if x not in stop))
test['Cleaned_Verse'].head()
```

```
2             mean regardless yon midnight bell
10    bids heavns bright guard paraclete remove
21                   anguish muses horror broods
27                         taught heart glow god
30            fond soul impassiond rapt unveild
Name: Cleaned_Verse, dtype: object
```

Remove the Rare Words

```
freq = pd.Series(' '.join(train['Cleaned_Verse']).split()).value_counts()
less_freq = list(freq[freq == 1].index)
```

```
train['Cleaned_Verse'] = train['Cleaned_Verse'].apply(lambda x: " ".join(x for x in x.split() if x not in less_freq))
train['Cleaned_Verse'].head(2)
```

```
69602     untainted vice art
133404     stung envy spleen
Name: Cleaned_Verse, dtype: object
```

```
freq = pd.Series(' '.join(train['Cleaned_Verse']).split()).value_counts()
less_freq = list(freq[freq == 1].index)
```

```
test['Cleaned_Verse'] = test['Cleaned_Verse'].apply(lambda x: " ".join(x for x in x.split() if x not in less_freq))
test['Cleaned_Verse'].head(2)
```

```
2             mean regardless yon midnight bell
10    bids heavns bright guard paraclete remove
Name: Cleaned_Verse, dtype: object
```

```
freq = pd.Series(' '.join(test['Cleaned_Verse']).split()).value_counts()
less_freq = list(freq[freq == 1].index)
```

```
from textblob import TextBlob, Word, Blobber
from nltk.stem import PorterStemmer
st = PorterStemmer()
```

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

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
True
```

```
from nltk.stem import WordNetLemmatizer
wordnet_lemmatizer = WordNetLemmatizer()
```

```
train['Cleaned_Verse'] = train['Cleaned_Verse'].apply(lambda x: " ".join([st.stem(word) for word in x.split()]))
```

```
test['Cleaned_Verse'] = test['Cleaned_Verse'].apply(lambda x: " ".join([st.stem(word) for word in x.split()]))
```

Adding the length of the review and the word count of each Verse

```
train['Cleaned_Verse_len'] = train['Cleaned_Verse'].astype(str).apply(len)
train['word_count'] = train['Cleaned_Verse'].apply(lambda x: len(str(x).split()))
```

```
test['Cleaned_Verse_len'] = test['Cleaned_Verse'].astype(str).apply(len)
test['word_count'] = test['Cleaned_Verse'].apply(lambda x: len(str(x).split()))
```

Adding Polarity

Add one more feature called polarity. **Polarity** shows the sentiment of a piece of text. It counts the negative and positive words and determines the polarity. The value ranges from -1 to 1 where -1 represents the negative sentiment, 0 represents neutral and 1 represent positive sentiment.

```
train['polarity'] = train['Cleaned_Verse'].map(lambda text: TextBlob(text).sentiment.polarity)
train.head(2)
```

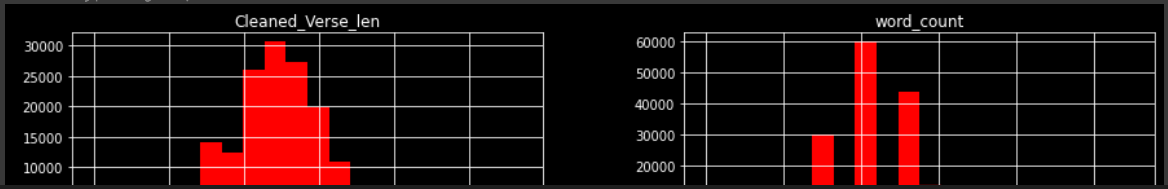
	index	Verse	Meter	char_count	Cleaned_Verse	Cleaned_Verse_len	word_count	polarity
69602	70451	and you untainted by the vice of art	2	6	untaint vice art	16	3	0.0
133404	136080	nor stung with envy nor with spleen diseas'd	2	6	stung envi spleen	17	3	0.0

```
test['polarity'] = test['Cleaned_Verse'].map(lambda text: TextBlob(text).sentiment.polarity)
test.head(2)
```

	index	Verse	Meter	char_count	Cleaned_Verse	Cleaned_Verse_len	word_count	polarity
2	2	what mean regardless of yon midnight bell	2	6	mean regardless yon midnight bell	33	5	-0.3125
10	10	bids heav'n's bright guard from paraclete remove	2	6	bid heavn bright guard paraclet remov	37	6	0.7000

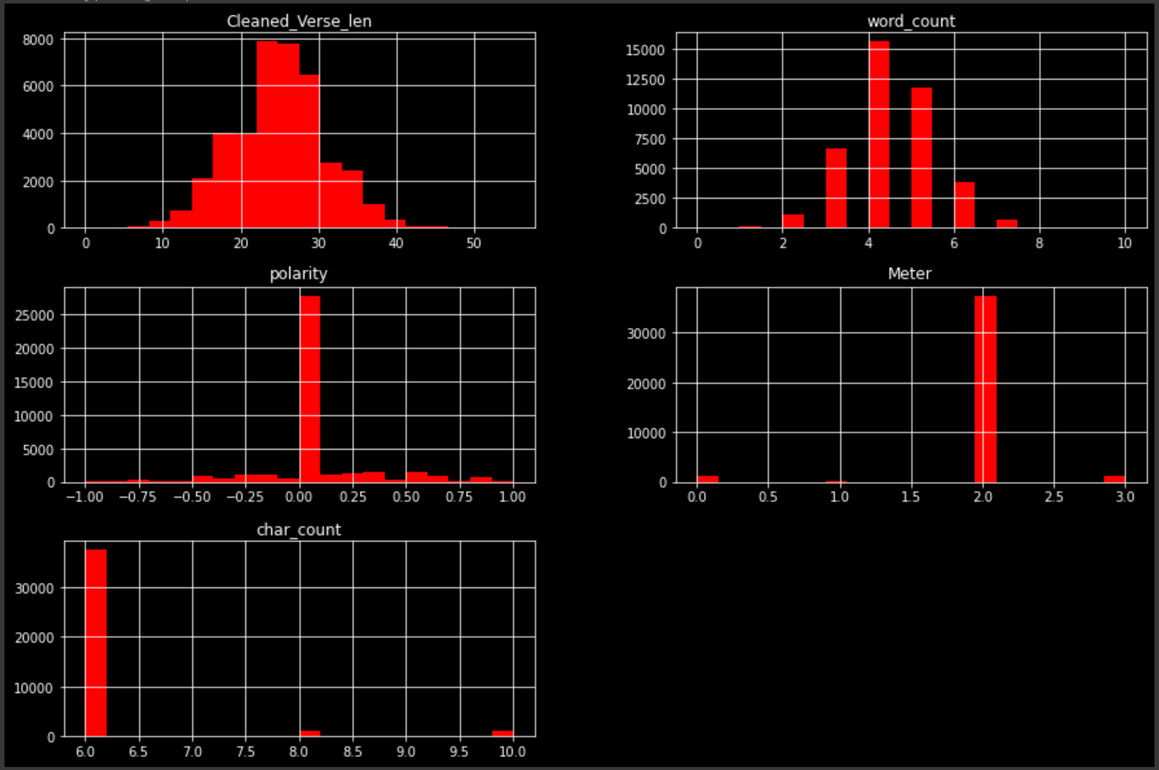
```
train[['Cleaned_Verse_len',·'word_count','polarity',·'Meter',·'char_count']].hist(bins=20,·figsize=(15,·10),·color='red')
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f27e0d9f250>,
         <matplotlib.axes._subplots.AxesSubplot object at 0x7f27e193e6a0>],
        [<matplotlib.axes._subplots.AxesSubplot object at 0x7f27e1877ac0>,
         <matplotlib.axes._subplots.AxesSubplot object at 0x7f27e1663eb0>],
        [<matplotlib.axes._subplots.AxesSubplot object at 0x7f27e181f2b0>,
         <matplotlib.axes._subplots.AxesSubplot object at 0x7f27e1821700>]],
       dtype=object)
```



```
test[['Cleaned_Verse_len', 'word_count','polarity', 'Meter', 'char_count']].hist(bins=20, figsize=(15, 10), color='red')
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f27e18242b0>,
         <matplotlib.axes._subplots.AxesSubplot object at 0x7f27e11f0940>],
        [<matplotlib.axes._subplots.AxesSubplot object at 0x7f27e0e5bd90>,
         <matplotlib.axes._subplots.AxesSubplot object at 0x7f27e179e1c0>],
        [<matplotlib.axes._subplots.AxesSubplot object at 0x7f27e17825b0>,
         <matplotlib.axes._subplots.AxesSubplot object at 0x7f27e128ca00>]],
       dtype=object)
```



```
train.columns
```

```
Index(['index', 'Verse', 'Meter', 'char_count', 'Cleaned_Verse',
       'Cleaned_Verse_len', 'word_count', 'polarity'],
      dtype='object')
```

```
condition_pol = train.groupby('Meter')['polarity'].agg([np.mean])
condition_pol.columns = ['polarity']
condition_pol = condition_pol.sort_values('polarity', ascending=False)
condition_pol = condition_pol.head(30)
condition_pol
```

	polarity
Meter	
0	0.067479
3	0.058498
2	0.047953
1	0.038299

```
condition_pol = test.groupby('Meter')['polarity'].agg([np.mean])
condition_pol.columns = ['polarity']
condition_pol = condition_pol.sort_values('polarity', ascending=False)
condition_pol = condition_pol.head(30)
condition_pol
```

	polarity
Meter	
1	0.080843
0	0.070277
2	0.047639
3	0.041828

WordCloud:

Wordcloud is a common and beautiful visualization for text data to plot the frequency of words. You may need to install wordcloud if you do not have it already, using this command:

```
####! pip install wordcloud
```

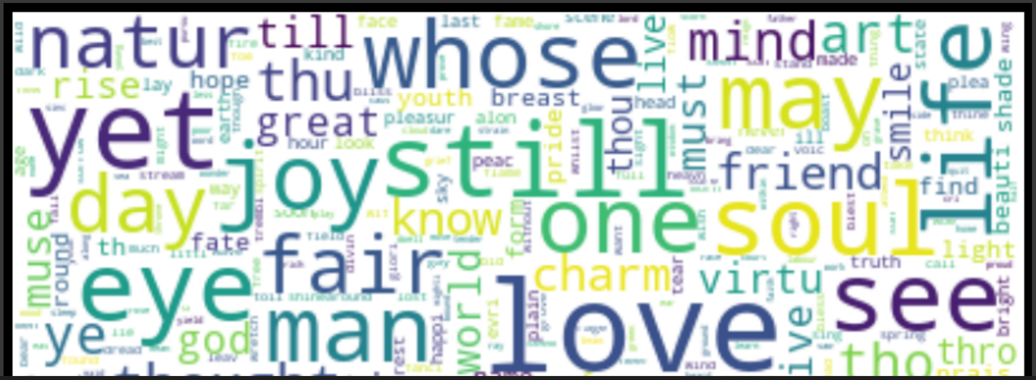
```
train.columns
```

```
Index(['index', 'Verse', 'Meter', 'char_count', 'Cleaned_Verse',
       'Cleaned_Verse_len', 'word_count', 'polarity'],
      dtype='object')
```

```
text_train = " ".join(review for review in train.Cleaned_Verse)
```

```
text_test = " ".join(review for review in test.Cleaned_Verse)
```

```
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
stopwords = set(STOPWORDS)
stopwords = stopwords.union(["ha", "thi", "now", "onli", "im", "becaus", "wa", "will", "even", "go", "realli", "didnt", "abl"])
wordcl = WordCloud(stopwords = stopwords, background_color='white', max_font_size = 50, max_words = 5000).generate(text_train)
plt.figure(figsize=(14, 12))
plt.imshow(wordcl, interpolation='bilinear')
plt.axis('off')
plt.show()
```



```
from sklearn.feature_extraction.text import CountVectorizer
```

```
def get_top_n_words(corpus, n=None):
    vec=CountVectorizer().fit(corpus)
    bag_of_words = vec.transform(corpus)
    sum_words = bag_of_words.sum(axis=0)
    words_freq = [(word, sum_words[0, idx]) for word, idx in vec.vocabulary_.items()]
    words_freq =sorted(words_freq, key = lambda x: x[1], reverse=True)
    return words_freq[:n]
common_words = get_top_n_words(train['Cleaned_Verse'], 20)
df1 = pd.DataFrame(common_words, columns = ['Cleaned_Verse', 'count'])
df1.head()
```

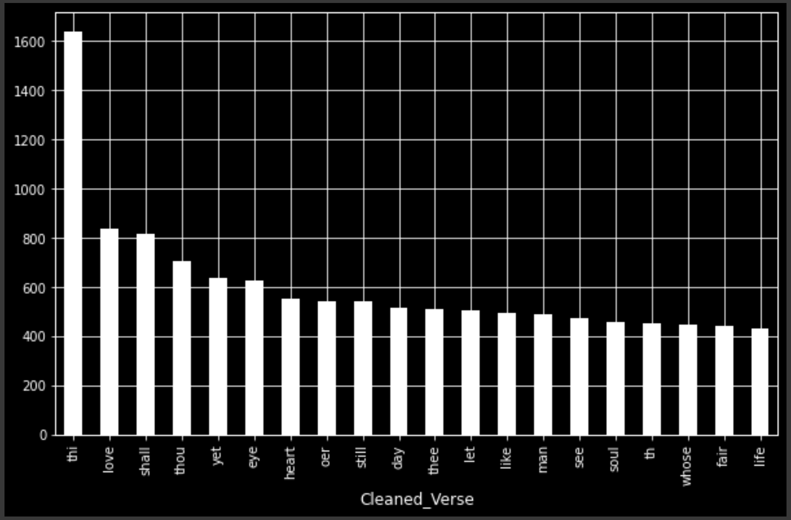
	Cleaned_Verse	count
0	thi	6397
1	love	3373
2	shall	3146
3	yet	2649
4	eye	2516

```
plt.style.use("dark_background")
df1.groupby('Cleaned_Verse').sum()['count'].sort_values(ascending=False).plot(kind='bar',color='white',figsize = (10, 6))
xlabel = 'Top Words'
ylabel = 'Count'
title = 'BarChart represent the Top Words Frequency'
plt.show()
```

```
def get_top_n_words(corpus, n=None):
    vec=CountVectorizer().fit(corpus)
    bag_of_words = vec.transform(corpus)
    sum_words = bag_of_words.sum(axis=0)
    words_freq = [(word, sum_words[0, idx]) for word, idx in vec.vocabulary_.items()]
    words_freq =sorted(words_freq, key = lambda x: x[1], reverse=True)
    return words_freq[:n]
common_words = get_top_n_words(test['Cleaned_Verse'], 20)
df2 = pd.DataFrame(common_words, columns = ['Cleaned_Verse', 'count'])
df2.head()
```

	Cleaned_Verse	count
0	thi	1637
1	love	840
2	shall	817
3	thou	704
4	yet	635

```
df2.groupby('Cleaned_Verse').sum()['count'].sort_values(ascending=False).plot(kind='bar',color='white',figsize = (10, 6))
xlabel = 'Top Words'
ylabel = 'Count'
title = 'BarChart represent the Top Words Frequency'
plt.show()
```

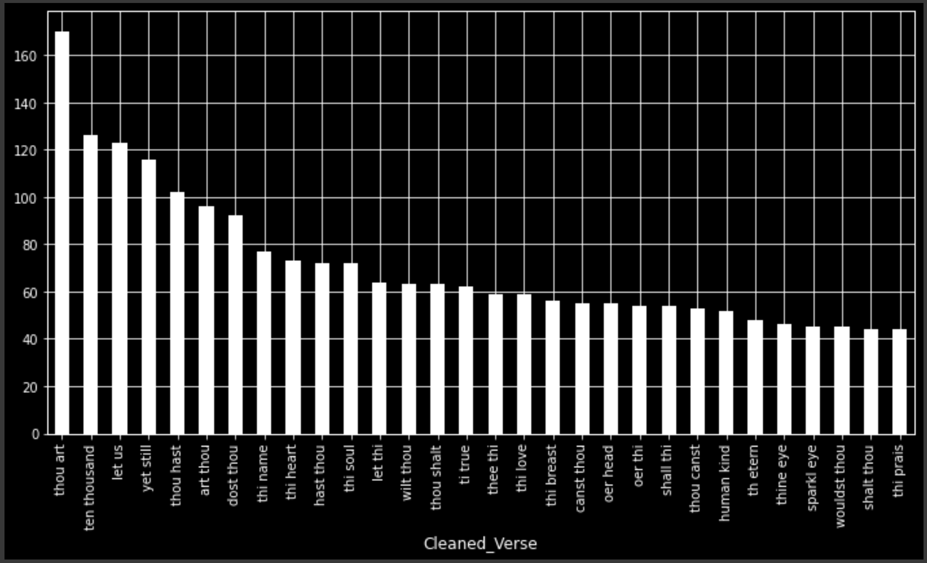


Bi-Grams

```
def get_top_n_bigram(corpus, n=None):
    vec = CountVectorizer(ngram_range=(2,2)).fit(corpus)
    bag_of_words = vec.transform(corpus)
    sum_words = bag_of_words.sum(axis=0)
    words_freq = [(word, sum_words[0, idx]) for word, idx in vec.vocabulary_.items()]
    words_freq =sorted(words_freq, key = lambda x: x[1], reverse=True)
    return words_freq[:n]
common_words2 = get_top_n_bigram(train['Cleaned_Verse'], 30)
df3 = pd.DataFrame(common_words2, columns=['Cleaned_Verse', "Count"])
df3.head()
```

	Cleaned_Verse	Count
0	thou art	170
1	ten thousand	126
2	let us	123
3	yet still	116
4	thou hast	102

```
df3.groupby('Cleaned_Verse').sum()['Count'].sort_values(ascending=False).plot(kind='bar',figsize=(12,6), color='white')
xlabel = "Bigram Words"
ylabel = "Count"
title = "Bar chart of Bigrams Frequency"
plt.show()
```

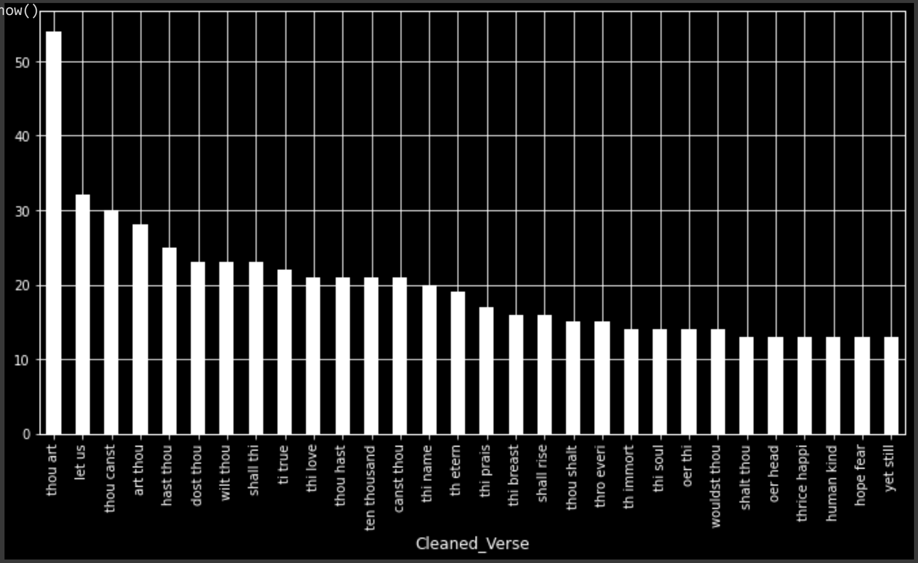


```
def get_top_n_bigram(corpus, n=None):
    vec = CountVectorizer(ngram_range=(2,2)).fit(corpus)
    bag_of_words = vec.transform(corpus)
    sum_words = bag_of_words.sum(axis=0)
    words_freq = [(word, sum_words[0, idx]) for word, idx in vec.vocabulary_.items()]
    words_freq =sorted(words_freq, key = lambda x: x[1], reverse=True)
    return words_freq[:n]
common_words3 = get_top_n_bigram(test['Cleaned_Verse'], 30)
df4 = pd.DataFrame(common_words3, columns=['Cleaned_Verse', "Count"])
df4.head()
```

	Cleaned_Verse	Count
0	thou art	54
1	let us	32
2	thou canst	30
3	art thou	28
4	hast thou	25

```
df4.groupby('Cleaned_Verse').sum()['Count'].sort_values(ascending=False).plot(kind='bar',figsize=(12,6), color='white')
xlabel = "Bigram Words"
ylabel = "Count"
```

```
title = "Bar chart of Bigrams Frequency"
plt.show()
```



```
train.columns
```

```
Index(['index', 'Verse', 'Meter', 'char_count', 'Cleaned_Verse',
       'Cleaned_Verse_len', 'word_count', 'polarity'],
      dtype='object')
```

```
X_train = train["Cleaned_Verse"]
```

```
y_train = train["Meter"]
```

```
X_test = test["Cleaned_Verse"]
```

```
y_test = test["Meter"]
```

```
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
```

```
(159202,) (159202,)
(39800,) (39800,)
```

Count Vectorizer

```
from sklearn.feature_extraction.text import CountVectorizer
# fit the countvectorizer to the training data:
vect = CountVectorizer().fit(X_train)
```

```
len(vect.get_feature_names()[:2000])
#len(vect.get_feature_names())
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function get_feature_names is deprecated; get_feature_names is deprecated in 1.0 and will be removed in 1.2. Please u
  warnings.warn(msg, category=FutureWarning)
2000
```

```
X_train_vectorized = vect.transform(X_train)
```

```
from sklearn.linear_model import LogisticRegression

# Train the model
model = LogisticRegression()
model.fit(X_train_vectorized, y_train)
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/linear_model/_logistic.py:814: 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(
LogisticRegression()
```

```
from sklearn.metrics import accuracy_score

# Predict the transformed test documents
predictions = model.predict(vect.transform(X_test))
print('Acurracy: ', accuracy_score(y_test, predictions))
```

```
Acurracy:  0.9407286432160804
```

TF-IDF Vectorizer

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
vect = TfidfVectorizer(min_df=5).fit(X_train)
len(vect.get_feature_names())
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function get_feature_names is deprecated; get_feature_names is deprecated in 1.0 and will be removed in 1.2. Please
  warnings.warn(msg, category=FutureWarning)
8931
```

```
X_train_vectorized = vect.transform(X_train)

model = LogisticRegression()
model.fit(X_train_vectorized, y_train)

predictions = model.predict(vect.transform(X_test))

print('Accuracy: ', accuracy_score(y_test, predictions))
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/linear_model/_logistic.py:814: 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(
Accuracy:  0.9397989949748744
```

```
feature_names = np.array(vect.get_feature_names())

sorted_tfidf_index = X_train_vectorized.max(0).toarray()[0].argsort()

print('Smallest tfidf:\n{}\n'.format(feature_names[sorted_tfidf_index[:10]]))
print('Largest tfidf: \n{}'.format(feature_names[sorted_tfidf_index[:-11:-1]]))
```

```
Smallest tfidf:
['casa' 'pr' 'basil' 'notari' 'forgd' 'talon' 'udder' 'raw' 'somer'
 'lybia']
```

```
Largest tfidf:
['dabbl' 'rather' 'street' 'turn' 'turnd' 'flew' 'avaric' 'fli' 'ill'
 'seat']
/usr/local/lib/python3.8/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function get_feature_names is deprecated; get_feature_names is deprecated in 1.0 and will be removed in 1.2. Please
  warnings.warn(msg, category=FutureWarning)
```

Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier
```

```
model = RandomForestClassifier()
model.fit(X_train_vectorized, y_train)
```

```
RandomForestClassifier()
```

```
predictions = model.predict(vect.transform(X_test))

print('Accuracy: ', accuracy_score(y_test, predictions))
```

```
Accuracy:  0.9353015075376885
```

```
predictions = pd.DataFrame(predictions)
```

Gradient Boosting Classifier

```
from sklearn.ensemble import GradientBoostingClassifier
```

```
model_gb = GradientBoostingClassifier()
model_gb.fit(X_train_vectorized, y_train)
```

```
GradientBoostingClassifier()
```

```
predictions_gb = model_gb.predict(vect.transform(X_test))

print('Accuracy: ', accuracy_score(y_test, predictions))
```

```
Accuracy:  0.9353015075376885
```

```
predictions_gb.shape
```

```
(39800,)
```

```
testvalue = pd.read_csv("/content/test.csv")
```

```
testvalue.shape
```

```
(39800, 5)
```

```
output = pd.concat([test, predictions], axis = 1)
```

```

output.rename(columns = { 0 :'Predict'}, inplace = True)
```

```
output.head(2)
```

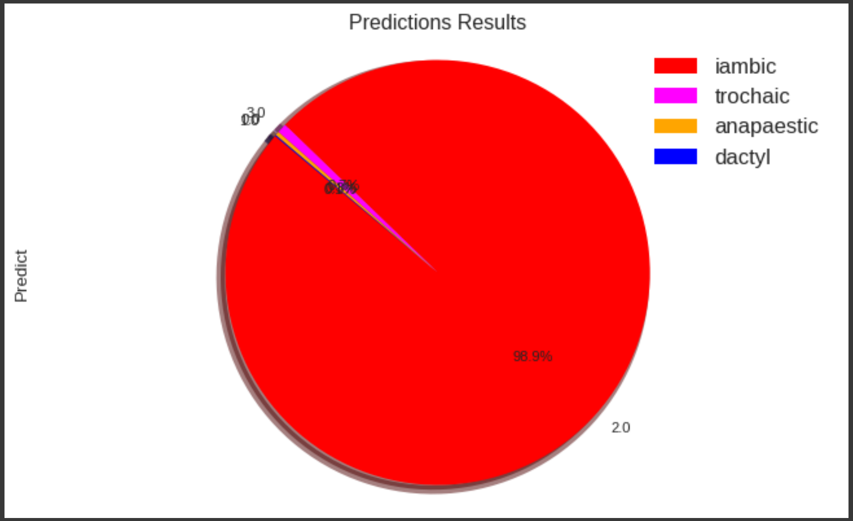
	index	Verse	Meter	char_count	Cleaned_Verse	Cleaned_Verse_len	word_count	polarity	Predict
2	2.0	what mean regardless of yon midnight bell	2.0	6.0	mean regardless yon midnight bell	33.0	5.0	-0.3125	2.0
10	10.0	bids heav'n's bright guard from paraclete remove	2.0	6.0	bid heavn bright guard paraclet remov	37.0	6.0	0.7000	2.0

```
output.Predict.value_counts()
```

```
2.0    39357
3.0      290
0.0      121
1.0       32
Name: Predict, dtype: int64
```

```
bw = output["Predict"].value_counts()
```

```
plt.figure(figsize=(10,6), dpi=80, facecolor='white')
plt.style.use('seaborn-white')
explode = (0.1, 0.1, 0.2, 0.2
           )
labels = ['iambic', 'trochaic', 'anapaestic',  'dactyl']
colors = ['red', 'magenta', 'orange', 'blue']
bw.plot.pie(shadow = True, colors=colors, autopct='%1.1f%%', wedgeprops = {'linewidth' : 6}, startangle=140)
plt.title('Predictions Results')
plt.legend(labels, loc="best", fontsize=15)
plt.axis('equal')
plt.show()
```



✓ 0s completed at 9:25 PM