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

```
dataFrame=pd.read_csv("/content/twitter_training.csv",names=["id","company","kind","tweet"])
dataFrame.head()
```

	id	company	kind	tweet
0	2401	Borderlands	Positive	im getting on borderlands and i will murder yo...
1	2401	Borderlands	Positive	I am coming to the borders and I will kill you...
2	2401	Borderlands	Positive	im getting on borderlands and i will kill you ...
3	2401	Borderlands	Positive	im coming on borderlands and i will murder you...
4	2401	Borderlands	Positive	im getting on borderlands 2 and i will murder ...

```
del dataFrame["id"] # Deleting ID column as it has no use in sentiment analysis
```

```
dataFrame.isnull().sum() #Finding the null entries in each column
```

```
company      0
kind         0
tweet      686
dtype: int64
```

```
dataFrame=dataFrame.dropna()
```

```
dataFrame.isnull().sum() #All the null values are removed
```

```
company    0
kind       0
tweet      0
dtype: int64
```

Counting the sentiment of each kind in the dataFrame

```
dataFrame_count = pd.DataFrame(dataFrame['kind'].value_counts()).reset_index()
dataFrame_count.columns = ['kind', 'Count']
dataFrame_count
```

	kind	Count
0	Negative	22358
1	Positive	20655

Removing Punctuation in Tweet column by creating a new column named new_Tweet

```
dataFrame['new_Tweet']=dataFrame['tweet'].str.replace('[^a-zA-Z0-9]',' ')
dataFrame.head()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: FutureWarning: The defa
  """Entry point for launching an IPython kernel.
```

	company	kind	tweet	
0	Borderlands	Positive	im getting on borderlands and i will murder yo...	im getting on borderlands
1	Borderlands	Positive	I am coming to the borders and I will kill you...	I am coming to the bord
2	Borderlands	Positive	im getting on borderlands and i will kill you ...	im getting on borderla
3	Borderlands	Positive	im coming on borderlands and i will murder you...	im coming on borderlands
4	Borderlands	Positive	im getting on borderlands 2 and i will murder ...	im getting on borderland

Removing short words like "of", "in" , "on" from the tweets as they are not helpful in sentiment analysis.

```
dataFrame['new_Tweet'] = dataFrame['new_Tweet'].apply(lambda row: ' '.join([word for word in
dataFrame.head()

# In this code we have removed all 2 letter words from tweets and replaced them with a space
```

	company	kind	tweet	
0	Borderlands	Positive	im getting on borderlands and i will murder yo...	getting borderlands and w
1	Borderlands	Positive	I am coming to the borders and I will kill you...	coming the borders a
2	Borderlands	Positive	im getting on borderlands and i will kill you ...	getting borderlands a
3	Borderlands	Positive	im coming on borderlands and i will murder you...	coming borderlands and w
4	Borderlands	Positive	im getting on borderlands 2 and i will murder ...	getting borderlands and w

```
dataFrame['new_Tweet']=dataFrame['new_Tweet'].str.lower()
```

```
dataFrame.head()
```

	company	kind	tweet	
0	Borderlands	Positive	im getting on borderlands and i will murder yo...	getting borderlands and w
1	Borderlands	Positive	I am coming to the borders and I will kill you...	coming the borders a
2	Borderlands	Positive	im getting on borderlands and i will kill you ...	getting borderlands a
3	Borderlands	Positive	im coming on borderlands and i will murder you...	coming borderlands and w
4	Borderlands	Positive	im getting on borderlands 2 and i will murder ...	getting borderlands and w

In the next few cells we will be removing stop words from the tweets

```
import nltk                                 #importing libraries for stop word removal
nltk.download('punkt')
nltk.download('stopwords')
from nltk.corpus import stopwords
from nltk import word_tokenize

stop_words = stopwords.words('english')
```

```
[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.
```

The below function takes a paragraph, breaks it into words, checks if the word is a stop word, removes if it is a stop word and combine the remaining words into a sentence again.

```
def remove_stopwords(twt):

    twt_tokenized = word_tokenize(twt)
    twt_new = " ".join([i for i in twt_tokenized  if i not in stop_words])
    return twt_new
```

```
dataFrame['new_Tweet'] = [remove_stopwords(t) for t in dataFrame['new_Tweet']]
```

```
dataFrame.head()
```

company	kind	tweet	new_Tweet

Lemmatinzation of the tweets:

In lemmatinzation we convert each word in the tweet to its base root word as many words used provide same meaning but in different verbal form. To avoid this we do lemmatization.

```
nltk.download('wordnet')      #Libraries required for lemmatization
nltk.download('omw-1.4')
nltk.download('averaged_perceptron_tagger')
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
```

```
# function to convert nltk tag to wordnet tag
lemmatizer = WordNetLemmatizer()



def nltk_tag_to_wordnet_tag(nltk_tag):    #Function that finds the parts of speech tag
    if nltk_tag.startswith('J'):
        return wordnet.ADJ
    elif nltk_tag.startswith('V'):
        return wordnet.VERB
    elif nltk_tag.startswith('N'):
        return wordnet.NOUN
    elif nltk_tag.startswith('R'):
        return wordnet.ADV
    else:
        return None


def lemmatize_sentence(sentence):     #Function for lemmatization
  # word tokenize -> pos tag (detailed) -> wordnet tag (shallow pos) -> lemmatizer -> root wo
    #tokenizes the sentence and finds the POS tag for each token
    nltk_tagged = nltk.pos_tag(nltk.word_tokenize(sentence))
    #tuple of (token, wordnet_tag)
    wordnet_tagged = map(lambda x: (x[0], nltk_tag_to_wordnet_tag(x[1])), nltk_tagged)
    lemmatized_sentence = []
    for word, tag in wordnet_tagged:
        if tag is None:
          lemmatized_sentence.append(word)
        else:
            #uses the tag to lemmatize the token
            lemmatized_sentence.append(lemmatizer.lemmatize(word, tag))
    return " ".join(lemmatized_sentence)
```

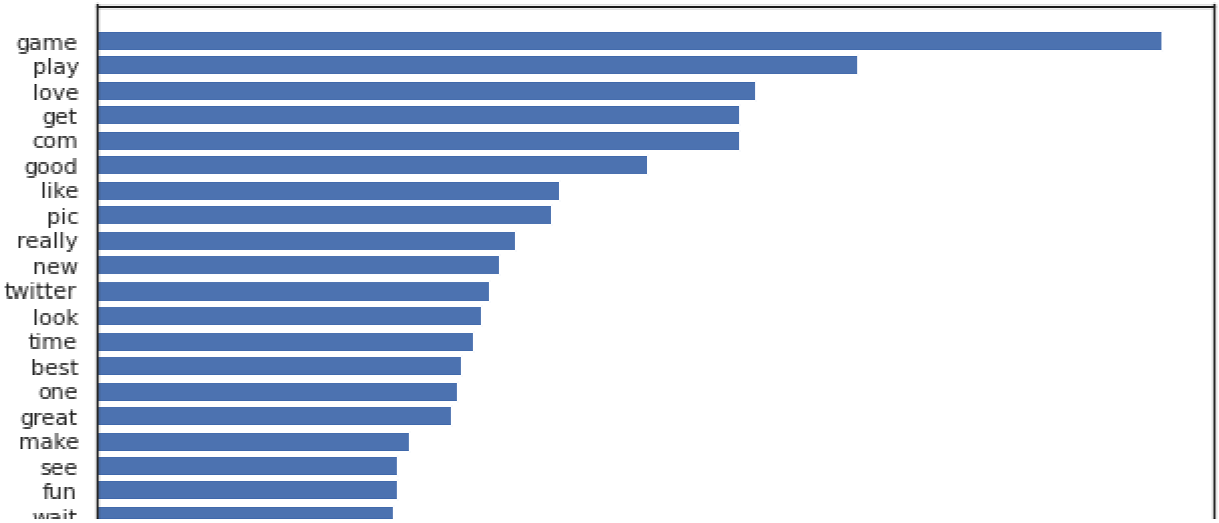
```
dataFrame['new_Tweet'] = dataFrame['new_Tweet'].apply(lambda x: lemmatize_sentence(x))
```

```
dataFrame.head()   #Data Frame after Lemmatization
```

	company	kind	tweet	new_Tweet
0	Borderlands	Positive	im getting on borderlands and i will murder yo...	get borderland murder
1	Borderlands	Positive	I am coming to the borders and I will kill you...	come border kill
2	Borderlands	Positive	im getting on borderlands and i will kill you ...	get borderland kill
3	Borderlands	Positive	im coming on borderlands and i will murder you...	come borderland murder
4	Borderlands	Positive	im getting on borderlands 2 and i will murder ...	get borderland murder

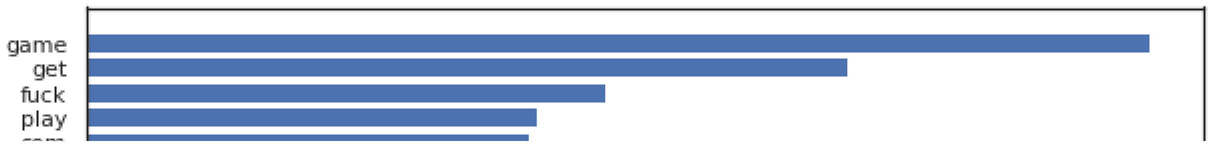
Plotting top 20 words in positive kind of sentences frequency

```
import matplotlib.pyplot as plt
import seaborn as sns
sns.set(style = 'white')
from nltk import FreqDist #function to find the frequent words in the data

# Subset positive review dataset
pos_Words = dataFrame.loc[dataFrame['kind'] == 'Positive',:]

#Extracts words into list and count frequency
pos_Words_1 = ' '.join([text for text in pos_Words['new_Tweet']])
pos_Words_1 = pos_Words_1.split()
words_df = FreqDist(pos_Words_1)

# Extracting words and frequency from words_df object
words_df = pd.DataFrame({'word':list(words_df.keys()), 'count':list(words_df.values())})
words_df
# Subsets top 20 words by frequency
words_df = words_df.nlargest(columns="count", n = 20)

words_df.sort_values('count', inplace = True)

# Plotting 20 frequent positive words
plt.figure(figsize=(10,5))
ax = plt.barh(words_df['word'], width = words_df['count'])
plt.show()
```



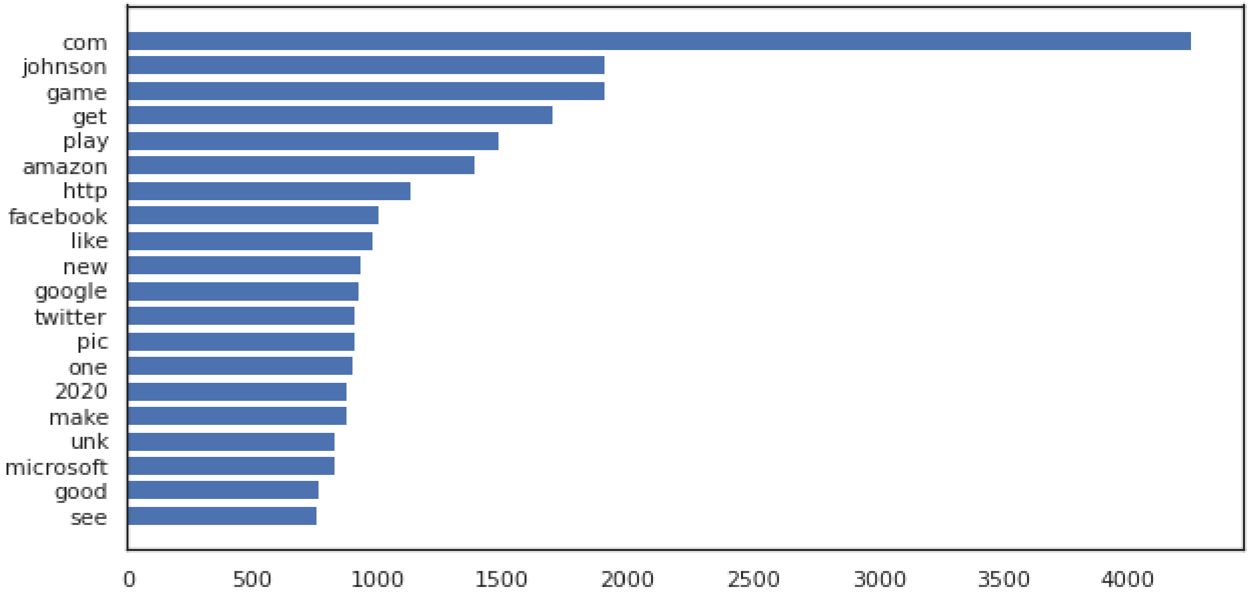
Plotting top 20 words in negative kind of sentences frequency

```
# Subset negative review dataset
neg_Words = dataFrame.loc[dataFrame['kind'] == 'Negative',:]

#Extracts words into list and count frequency
neg_Words_1 = ' '.join([text for text in neg_Words['new_Tweet']])
neg_Words_1 = neg_Words_1.split()
words_df = FreqDist(neg_Words_1)

# Extracting words and frequency from words_df object
words_df = pd.DataFrame({'word':list(words_df.keys()), 'count':list(words_df.values())})
words_df
# Subsets top 20 words by frequency
words_df = words_df.nlargest(columns="count", n = 20)

words_df.sort_values('count', inplace = True)

# Plotting 20 frequent negative words
plt.figure(figsize=(10,5))
ax = plt.barh(words_df['word'], width = words_df['count'])
plt.show()
```



Plotting top 20 words in neutral kind of sentences frequency

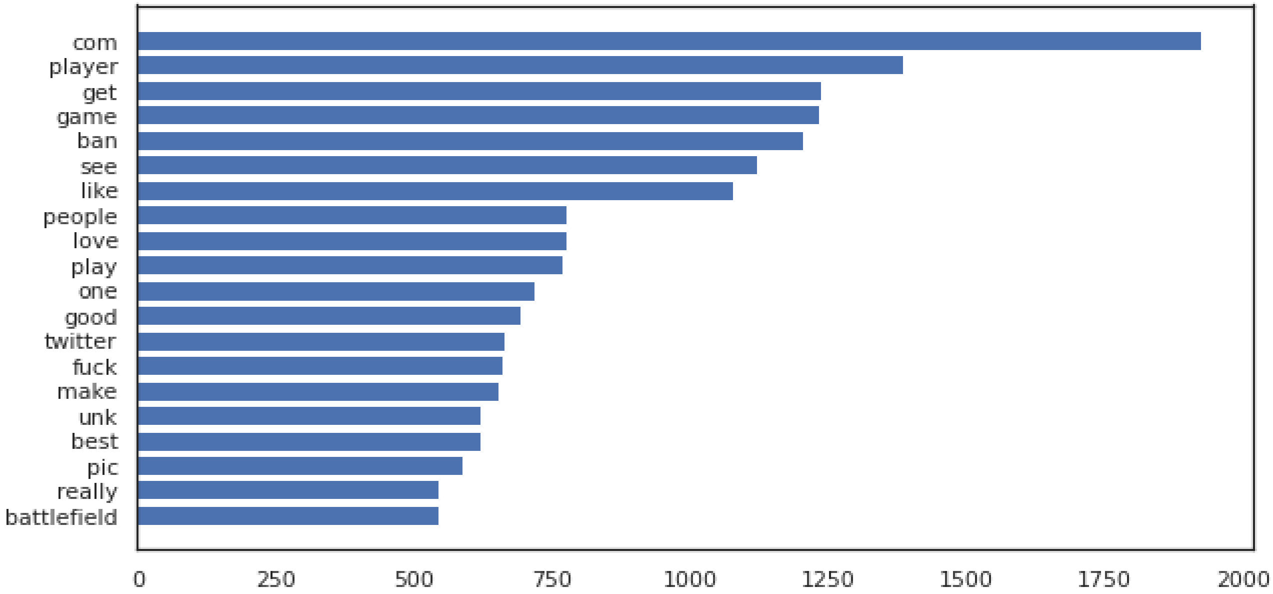
```
# Subset Neutral review dataset
neu_Words = dataFrame.loc[dataFrame['kind'] == 'Neutral',:]

#Extracts words into list and count frequency
neu_Words_1 = ' '.join([text for text in neu_Words['new_Tweet']])
neu_Words_1 = neu_Words_1.split()
words_df = FreqDist(neu_Words_1)

# Extracting words and frequency from words_df object
words_df = pd.DataFrame({'word':list(words_df.keys()), 'count':list(words_df.values())})
words_df
# Subsets top 20 words by frequency
words_df = words_df.nlargest(columns="count", n = 20)

words_df.sort_values('count', inplace = True)

# Plotting 20 frequent neutral words
plt.figure(figsize=(10,5))
ax = plt.barh(words_df['word'], width = words_df['count'])
plt.show()
```



Plotting top 20 words in Irrelevant kind of sentences frequency

```
# Subset Irrelavent review dataset
irr_Words = dataFrame.loc[dataFrame['kind'] == 'Irrelevant',:]
```

```
#Extracts words into list and count frequency
irr_Words_1 = ' '.join([text for text in irr_Words['new_Tweet']])
irr_Words_1 = irr_Words_1.split()
words_df = FreqDist(irr_Words_1)

# Extracting words and frequency from words_df object
words_df = pd.DataFrame({'word':list(words_df.keys()), 'count':list(words_df.values())})
words_df
# Subsets top 20 words by frequency
words_df = words_df.nlargest(columns="count", n = 20)

words_df.sort_values('count', inplace = True)

# Plotting 20 frequent negative words
plt.figure(figsize=(10,5))
ax = plt.barh(words_df['word'], width = words_df['count'])
plt.show()
```



In TF-IDF, we calculate the TF-IDF score of each word in accordance with the dataset. After that we store the vlaues into a vector.

```
#Applying TF-IDF vectorizer

from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer(max_features=2500)
```

Encoding:

In this step we are mapping positive to 1, negative to 0, neutral to 2 and irrelevant to 3.

```
dataFrame['kind']=dataFrame['kind'].map({'Positive':1, 'Neutral':2, 'Negative':0, 'Irrelevant
```

Decision Tree

```
X = tfidf.fit_transform(dataFrame['new_Tweet']).toarray()
y = dataFrame['kind'].values
featureNames = tfidf.get_feature_names()

# Splitting the dataset into train and test
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=30)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning:
  warnings.warn(msg, category=FutureWarning)
```

Fitting model

```
from sklearn.tree import DecisionTreeClassifier

dt = DecisionTreeClassifier()
dt.fit(X_train,y_train)

y_pred = dt.predict(X_test)
```

```
from sklearn.metrics import roc_auc_score
roc_auc_score(y_test, dt.predict_proba(X_test),multi_class="ovo")
```

```
0.9641414141414141
```

```
X = tfidf.fit_transform(dataFrame['new_Tweet']).toarray()
y = dataFrame["kind"].values
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
from sklearn.ensemble import RandomForestClassifier
rf_model = RandomForestClassifier(n_estimators=50, max_features="auto")
rf_model.fit(X_train, y_train)
```

```
RandomForestClassifier(n_estimators=50)
```

```
predictions = rf_model.predict(X_test)
predictions
```

```
    array([0, 1, 0, ..., 2, 2, 0])
```

```
from sklearn.metrics import roc_auc_score
roc_auc_score(y_test, rf_model.predict_proba(X_test),multi_class="ovo")
```

```
    0.9700408416100331
```

```
featureImportance = pd.DataFrame({i : j for i,j in zip(rf_model.feature_importances_,featureN
featureImportance.sort_values(by='Importance',ascending=False)
```

	Importance	word
457	0.012058	com
1321	0.009460	love
880	0.006747	fix
719	0.006685	eamaddennfl
931	0.006650	fuck
...	...	...
277	0.000011	blogspot
2345	0.000009	url
134	0.000009	anna
1216	0.000005	kamuy
1030	0.000001	hakusho

2500 rows × 2 columns

Colab paid products - Cancel contracts here

✓ 0s completed at 21:17 ● ×