# EMOTION CLASSIFICATION NLP

## INTRODUCTION

## About Dataset

Identifying emotions has become an integral part of many NLP and data science projects. With the help of this dataset, one can train and build various robust models and perform emotional analysis.Emotional classification using Natural Language Processing (NLP) involves the task of identifying and categorizing emotions expressed in text data. This area of NLP is crucial for various applications such as sentiment analysis, customer feedback analysis, and mental health monitoring.Emotional classification using NLP continues to evolve with advancements in machine learning and deep learning techniques, making it a powerful tool for understanding and analyzing human emotions expressed through text.There are several Kaggle datasets that focus on emotional classification using NLP techniques. These datasets typically provide text data along with labels indicating the emotions expressed in the text.Kaggle datasets for emotional classification using NLP provide valuable resources for researchers and data enthusiasts to develop, test, and benchmark models that can effectively understand and classify emotions expressed in textual data. These datasets contribute significantly to advancing research and applications in sentiment analysis and emotional AI.

***AIM:Classified based on 4 emotions - joy, sadness, anger and fear.***

## Text Preprocessing

Before analyzing emotions, the text data usually undergoes preprocessing steps like tokenization (breaking text into words or phrases), removing stopwords, and possibly stemming or lemmatization to normalize the text.

***1.STOP WORDS***

In Natural Language Processing (NLP), stop words refer to commonly used words that typically do not carry significant meaning on their own and are often filtered out during text preprocessing. These words include articles (e.g., "the", "a", "an"), prepositions (e.g., "in", "on", "at"), conjunctions (e.g., "and", "but", "or"), and some common verbs (e.g., "is", "have", "do").Stop words in NLP serve the purpose of reducing noise and focusing on meaningful content words in text data. Their removal is a standard preprocessing step that enhances the efficiency and effectiveness of various NLP applications and tasks.

***2.TOKENIZATION***

Tokenization in Natural Language Processing (NLP) is the process of breaking down a text into smaller units called tokens. These tokens could be words, phrases, or other meaningful elements. Tokenization is a fundamental step in most NLP tasks because it allows machines to understand and process human language.Tokenization is a foundational step in NLP that involves dividing text into meaningful units (tokens) for further analysis and processing. It plays a crucial role in extracting features from text data and enabling machines to understand and work with human language effectively.

Types of tokenization are: Word Tokenization-This is the most common form of tokenization, where each word in a sentence is considered a token.

Sentence Tokenization-In some cases, the task might require breaking down text into sentences first. Sentence tokenization involves splitting text into individual sentences.

Subword Tokenization-Subword tokenization breaks words into smaller units called subword tokens. This is particularly useful for languages with complex morphology or for handling rare words.

### *3.STEMMING*

Stemming in Natural Language Processing (NLP) is the process of reducing words to their base or root form. The goal of stemming is to reduce inflected (or derived) words to a common base form to normalize variations of words and improve text analysis.Stemming is a valuable technique in NLP for reducing words to their base forms to improve text analysis and processing. While stemming algorithms like Porter Stemmer and Snowball Stemmer are widely used, they have limitations that need to be considered based on specific NLP tasks and language characteristics.

Porter Stemmer: Developed by Martin Porter in 1980, the Porter stemming algorithm is one of the most widely used stemming algorithms. It uses a set of rules for suffix stripping to reduce words to their stems. While simple and fast, it may not always produce the most linguistically correct stems.

### *4.LEMMATIZATION*

Lemmatization in Natural Language Processing (NLP) is the process of reducing words to their base or canonical form, which is called the lemma. Unlike stemming, which reduces words to a root form by removing suffixes, lemmatization considers the context and meaning of a word to ensure that the root form is a meaningful word.

### *5.WORD CLOUD*

A word cloud in NLP is a visual representation of text data where the size of each word indicates its frequency or importance in the dataset.It's a popular technique used to quickly and intuitively summarize textual information, highlighting the most prominent terms.Word clouds provide a quick and visually appealing way to grasp the most significant terms in a text corpus. They are useful for exploratory data analysis, identifying themes, and gaining insights into the content of textual data. Adjust the preprocessing and customization according to your specific requirements and dataset characteristics for optimal results.

#### *6.TF-IDF (Term Frequency-Inverse Document Frequency)*

TF-IDF (Term Frequency-Inverse Document Frequency) is a statistical measure used in natural language processing (NLP) to evaluate how important a word is to a document in a collection (corpus).TF-IDF is a fundamental tool in NLP for extracting meaningful information from text data, especially in tasks where understanding the relative importance of terms within documents and across a corpus is crucial.

#### *7.POS TAG*

POS Tagging, or Part-of-Speech Tagging, is a fundamental task in natural language processing (NLP) that involves labeling words in a text with their corresponding part-of-speech categories (such as nouns, verbs, adjectives, etc.). Common POS Tags: POS tags are usually represented using standard sets such as the Penn Treebank tagset, which includes tags like:

NN: Noun, singular or mass VB: Verb, base form JJ: Adjective PRP: Personal pronoun IN: Preposition or subordinating conjunction DT: Determiner CC: Coordinating conjunction

POS tagging is essential for many NLP tasks, enabling machines to understand the grammatical structure and meaning of text, making it a crucial component in various applications from information extraction to machine translation.

## MODEL CREATION

Naive Bayes is a simple probabilistic classifier based on applying Bayes' theorem with strong (naive) independence assumptions between the features. It is widely used in machine learning, particularly for classification tasks in natural language processing (NLP).

#### *Naive Independence Assumption:*

Naive Bayes assumes that the presence of a particular feature in a class is independent of the presence of any other feature, given the class label. This is a strong assumption and hence the term "naive". Despite its simplicity, Naive Bayes often performs surprisingly well in practice, especially for text classification tasks.

#### *Types of Naive Bayes Classifiers:*

Multinomial Naive Bayes: Suitable for classification with discrete features (e.g., word counts for text classification).

Gaussian Naive Bayes: Assumes features follow a normal distribution. It is used for continuous features.

Bernoulli Naive Bayes: Similar to Multinomial NB but considers binary features (e.g., presence or absence of a term in a document).

#### *Application in NLP:*

In NLP, Naive Bayes classifiers are commonly used for:

Text Classification: Such as spam detection, sentiment analysis, and topic categorization.

Document Classification: Sorting documents into predefined categories based on their content.

Language Detection: Identifying the language of a given text.

Naive Bayes remains a popular choice in NLP due to its simplicity, efficiency, and often surprisingly good performance, especially on text data with high-dimensional feature spaces like word frequencies or presence/absence of words.

In [2]:
```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt
from wordcloud import WordCloud
```

In [3]:
```
df1=pd.read_csv('emotion-labels-test.csv')
df2=pd.read_csv('emotion-labels-train.csv')
df3=pd.read_csv('emotion-labels-val.csv')
df=pd.concat([df2,df3],axis=0)
```

In [4]:
```
df1
```

Out[4]:

| | text | label |
|---|---|---|
| **0** | You must be knowing #blithe means (adj.) Happ... | joy |
| **1** | Old saying 'A #smile shared is one gained for ... | joy |
| **2** | Bridget Jones' Baby was bloody hilarious 😅 #Br... | joy |
| **3** | @Elaminova sparkling water makes your life spa... | joy |
| **4** | I'm tired of everybody telling me to chill out... | joy |
| **...** | ... | ... |
| **3137** | Why does Candice constantly pout #GBBO 💄😒 | sadness |
| **3138** | @redBus_in #unhappy with #redbus CC, when I ta... | sadness |
| **3139** | @AceOperative789 no pull him afew weeks ago, s... | sadness |
| **3140** | I'm buying art supplies and I'm debating how s... | sadness |
| **3141** | @sainsburys Could you ask your Chafford Hundre... | sadness |

3142 rows × 2 columns

In [5]:
```
df
```

Out[5]:

| | text | label |
|---|---|---|
| **0** | Just got back from seeing @GaryDelaney in Burs... | joy |
| **1** | Oh dear an evening of absolute hilarity I don'... | joy |
| **2** | Been waiting all week for this game ❤️❤️❤️ #ch... | joy |
| **3** | @gardiner_love : Thank you so much, Gloria! Yo... | joy |
| **4** | I feel so blessed to work with the family that... | joy |
| **...** | ... | ... |
| **342** | Common app just randomly logged me out as I wa... | sadness |
| **343** | I'd rather laugh with the rarest genius, in be... | sadness |
| **344** | If you #invest in my new #film I will stop ask... | sadness |
| **345** | Just watched Django Unchained, Other people ma... | sadness |
| **346** | @KeithOlbermann depressing how despicable Trum... | sadness |

3960 rows × 2 columns

## PREPROCESSING

In [6]:
```
df1.head()
```

Out[6]:

| | text | label |
|---|---|---|
| **0** | You must be knowing #blithe means (adj.) Happ... | joy |
| **1** | Old saying 'A #smile shared is one gained for ... | joy |
| **2** | Bridget Jones' Baby was bloody hilarious 😅 #Br... | joy |
| **3** | @Elaminova sparkling water makes your life spa... | joy |
| **4** | I'm tired of everybody telling me to chill out... | joy |

In [7]:
```
df.head()
```

Out[7]:

| | text | label |
|---|---|---|
| **0** | Just got back from seeing @GaryDelaney in Burs... | joy |
| **1** | Oh dear an evening of absolute hilarity I don'... | joy |
| **2** | Been waiting all week for this game ❤️❤️❤️ #ch... | joy |
| **3** | @gardiner_love : Thank you so much, Gloria! Yo... | joy |
| **4** | I feel so blessed to work with the family that... | joy |

In [8]:
```
df1.tail()
```

Out[8]:

| | text | label |
|---|---|---|
| **3137** | Why does Candice constantly pout #GBBO 💄😒 | sadness |
| **3138** | @redBus_in #unhappy with #redbus CC, when I ta... | sadness |
| **3139** | @AceOperative789 no pull him afew weeks ago, s... | sadness |
| **3140** | I'm buying art supplies and I'm debating how s... | sadness |
| **3141** | @sainsburys Could you ask your Chafford Hundre... | sadness |

In [9]:
```
df.tail()
```

Out[9]:

| | text | label |
|---|---|---|
| **342** | Common app just randomly logged me out as I wa... | sadness |
| **343** | I'd rather laugh with the rarest genius, in be... | sadness |
| **344** | If you #invest in my new #film I will stop ask... | sadness |
| **345** | Just watched Django Unchained, Other people ma... | sadness |
| **346** | @KeithOlbermann depressing how despicable Trum... | sadness |

In [10]:
```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3142 entries, 0 to 3141
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   text    3142 non-null   object
 1   label   3142 non-null   object
dtypes: object(2)
memory usage: 49.2+ KB
```

In [11]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3960 entries, 0 to 346
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   text    3960 non-null   object
 1   label   3960 non-null   object
dtypes: object(2)
memory usage: 92.8+ KB
```

In [12]:
```
df1.shape
```

Out[12]:
```
(3142, 2)
```

In [13]:
```
df.shape
```

Out[13]:
```
(3960, 2)
```

In [14]:
```
df1.label.value_counts()
```

Out[14]:
```
fear       995
anger      760
joy        714
sadness    673
Name: label, dtype: int64
```

In [15]:
```
df.label.value_counts()
```

Out[15]:
```
fear       1257
anger       941
joy         902
sadness     860
Name: label, dtype: int64
```

In [16]:
```
df1.dtypes
```

Out[16]:
```
text     object
label    object
dtype: object
```

In [17]:
```
df.dtypes
```

Out[17]:
```
text     object
label    object
dtype: object
```

In [18]:
```
df1.nunique()
```

Out[18]:
```
text     3099
label       4
dtype: int64
```

In [19]:
```
df.nunique()
```

Out[19]:
```
text     3900
label       4
dtype: int64
```

In [20]:
```
df1.isna().sum()
```

Out[20]:
```
text     0
label    0
dtype: int64
```

In [21]:
```
df.isna().sum()
```

Out[21]:
```
text     0
label    0
dtype: int64
```

## LABEL ENCODING

In [23]:
```
le_data=LabelEncoder()
model1=le_data.fit_transform(df1['label'])
df1['label']=model1
```

In [24]:
```
model1
```

Out[24]:
```
array([2, 2, 2, ..., 3, 3, 3])
```

In [25]:
```
le_data=LabelEncoder()
model=le_data.fit_transform(df['label'])
df['label']=model
```

In [26]:
```
model
```

Out[26]:
```
array([2, 2, 2, ..., 3, 3, 3])
```

In [27]:
```
df1
```

Out[27]:

| | text | label |
|---|---|---|
| **0** | You must be knowing #blithe means (adj.) Happ... | 2 |
| **1** | Old saying 'A #smile shared is one gained for ... | 2 |
| **2** | Bridget Jones' Baby was bloody hilarious 😅 #Br... | 2 |
| **3** | @Elaminova sparkling water makes your life spa... | 2 |
| **4** | I'm tired of everybody telling me to chill out... | 2 |
| **...** | ... | ... |
| **3137** | Why does Candice constantly pout #GBBO 💄😒 | 3 |
| **3138** | @redBus_in #unhappy with #redbus CC, when I ta... | 3 |
| **3139** | @AceOperative789 no pull him afew weeks ago, s... | 3 |
| **3140** | I'm buying art supplies and I'm debating how s... | 3 |
| **3141** | @sainsburys Could you ask your Chafford Hundre... | 3 |

3142 rows × 2 columns

In [28]:
```
df
```

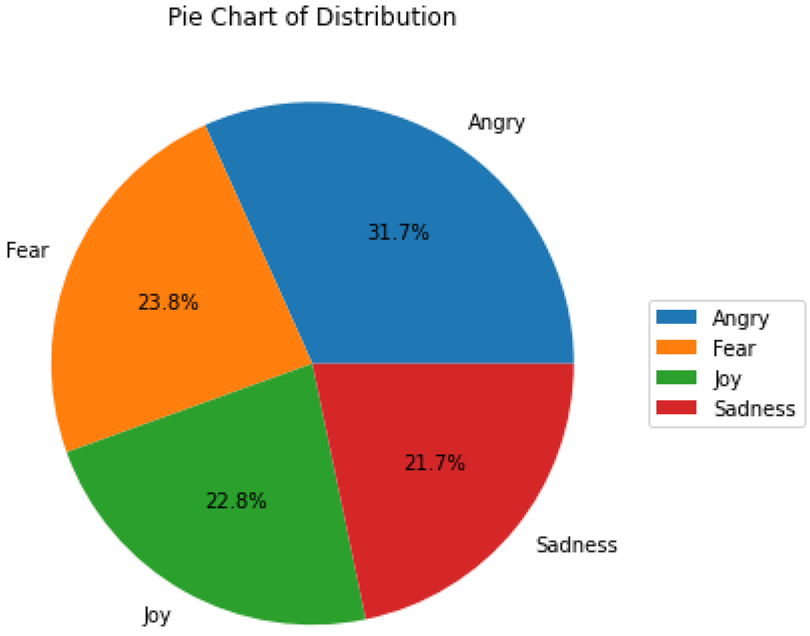
Out[28]:

|  | text | label |
|---|---|---|
| **0** | Just got back from seeing @GaryDelaney in Burs... | 2 |
| **1** | Oh dear an evening of absolute hilarity I don'... | 2 |
| **2** | Been waiting all week for this game ❤️❤️❤️ #ch... | 2 |
| **3** | @gardiner_love : Thank you so much, Gloria! Yo... | 2 |
| **4** | I feel so blessed to work with the family that... | 2 |
| **...** | ... | ... |
| **342** | Common app just randomly logged me out as I wa... | 3 |
| **343** | I'd rather laugh with the rarest genius, in be... | 3 |
| **344** | If you #invest in my new #film I will stop ask... | 3 |
| **345** | Just watched Django Unchained, Other people ma... | 3 |
| **346** | @KeithOlbermann depressing how despicable Trum... | 3 |

3960 rows × 2 columns

Label 0 indicates that the message is Angry, Label 1 signifies that the message is Fear,Label 2 indicates that the message is Joy and label 3 indicates that the message is Sadness

In [30]:
```
category_counts = df['label'].value_counts()
# Pie chart
plt.figure(figsize=(8,6))
labels=['Angry','Fear','Joy','Sadness']
plt.pie(category_counts,labels=labels,autopct='%1.1f%%')
plt.title('Pie Chart of Distribution')
# # Add Legend
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
plt.show()
```



In [31]:
```
pip install nltk
```

```
Requirement already satisfied: nltk in c:\users\user\anaconda3\lib\site-pac
kages (3.7)
Requirement already satisfied: joblib in c:\users\user\anaconda3\lib\site-p
ackages (from nltk) (1.3.2)
Requirement already satisfied: click in c:\users\user\anaconda3\lib\site-pa
ckages (from nltk) (8.0.4)
Requirement already satisfied: tqdm in c:\users\user\anaconda3\lib\site-pac
kages (from nltk) (4.64.0)
Requirement already satisfied: regex>=2021.8.3 in c:\users\user\anaconda3\l
ib\site-packages (from nltk) (2022.3.15)
Requirement already satisfied: colorama in c:\users\user\anaconda3\lib\site
-packages (from click->nltk) (0.4.4)
Note: you may need to restart the kernel to use updated packages.
```

In [32]:
```
import nltk
```

In [33]:
```
nltk.download('all')
```

```
[nltk_data] Downloading collection 'all'
[nltk_data]    |
[nltk_data]    | Downloading package abc to
[nltk_data]    |     C:\Users\user\AppData\Roaming\nltk_data...
[nltk_data]    |   Package abc is already up-to-date!
[nltk_data]    | Downloading package alpino to
[nltk_data]    |     C:\Users\user\AppData\Roaming\nltk_data...
[nltk_data]    |   Package alpino is already up-to-date!
[nltk_data]    | Downloading package averaged_perceptron_tagger to
[nltk_data]    |     C:\Users\user\AppData\Roaming\nltk_data...
[nltk_data]    |   Package averaged_perceptron_tagger is already up-
[nltk_data]    |       to-date!
[nltk_data]    | Downloading package averaged_perceptron_tagger_ru to
[nltk_data]    |     C:\Users\user\AppData\Roaming\nltk_data...
[nltk_data]    |   Package averaged_perceptron_tagger_ru is already
[nltk_data]    |       up-to-date!
[nltk_data]    | Downloading package basque_grammars to
[nltk_data]    |     C:\Users\user\AppData\Roaming\nltk_data...
[nltk_data]    |   Package basque_grammars is already up-to-date!
[nltk_data]    | Downloading package bcp47 to
```

## REMOVING STOPWORDS,TOKENIZATION,STEMMING,LEMMATIZATION

In [105]:

```
import re
import nltk
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('averaged_perceptron_tagger')
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer,WordNetLemmatizer
from nltk.tokenize import word_tokenize
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\user\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\user\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\user\AppData\Roaming\nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

### *LEMMATIZATION*

In [93]:

```
# TEST DATA
```

In [153]:

```
lemmatizer=WordNetLemmatizer()
```

In [154]:

```
k=df1['text']
```

In [155]:

```
corpus1=[]
```

In [156]:

```
corpus2=[] #for storing pos tag
```

In [157]:

```
for i in k:
    review=re.sub('[^a-zA-Z]', ' ',i)
    review=review.lower()
    review=nltk.word_tokenize(review)
    review2=nltk.pos_tag(review) #pos tag
    review=[lemmatizer.lemmatize(word) for word in review if not word in set
    review=" ".join(review)
    corpus1.append(review)
    corpus2.append(review2)
```

In [158]:
```
corpus1
```

```
lazy amp pointless',
 'got official jrjyp happy birthday jin young princejinyoungday happyjiny
oungday got birthday',
 'got official jrjyp happy birthday jin young princejinyoungday happyjiny
oungday got happy birthday',
 'race advance extra achievement individual individual charles towne n op
timism inspire',
 'race advance extra achievement individual individual charles towne n in
spire',
 'watching football match without commentary something rejoice found tran
smission city match like today joyful',
 'twd come soon happy',
 'twd come soon',
 'taudeltaphidk thank obama cut elated back home',
 'ddogsscout oh almost odd cheerfulness big bos offer muzzle flash blindi
ng accidental guy became best friend',
 'gemma simmons bright spot premiere far agentsofshield',
 'beautiful day lord made rejoice glad',
 'watch amazing live ly broadcast kelli peterson lively musically come wa
tch',
```

In [160]:
```
corpus2
```

Out[160]:
```
[[('you', 'PRP'),
  ('must', 'MD'),
  ('be', 'VB'),
  ('knowing', 'VBG'),
  ('blithe', 'JJ'),
  ('means', 'NNS'),
  ('adj', 'VBP'),
  ('happy', 'JJ'),
  ('cheerful', 'NN')],
 [('old', 'JJ'),
  ('saying', 'VBG'),
  ('a', 'DT'),
  ('smile', 'NN'),
  ('shared', 'VBN'),
  ('is', 'VBZ'),
  ('one', 'CD'),
  ('gained', 'VBN'),
  ('for', 'IN'),
  ('another', 'DT'),
```

In [100]:
```
# CONCATENATE DATA
```

In [182]:
```
k=df['text']
```

In [183]:
```
corpus=[]
```

In [184]:
```
corpus2=[] # for storing pos taq
```

In [185]:
```
for i in k:
    review=re.sub('[^a-zA-Z]', ' ',i)
    review=review.lower()
    review=nltk.word_tokenize(review)
    review2=nltk.pos_tag(review) #pos tag
    review=[lemmatizer.lemmatize(word) for word in review if not word in set
    review=" ".join(review)
    corpus.append(review)
    corpus2.append(review2)
```

In [186]:
```
corpus
```

Out[186]:
```
['got back seeing garydelaney burslem amazing face still hurt laughing mu
ch hilarious',
 'oh dear evening absolute hilarity think laughed much long time',
 'waiting week game cheer friday',
 'gardiner love thank much gloria sweet thoughtful made day joyful love',
 'feel blessed work family nanny nothing love amp appreciation make smil
e',
 'today reached subscriber yt goodday thankful',
 'singaholic good morning love happy first day fall let make awesome autu
mnmemories annabailey laughter smile',
 'bridgetjonesbaby best thing seen age funny missed bridget love teammar
k',
 'got back seeing garydelaney burslem amazing face still hurt laughing mu
ch',
 'indymn thought holiday could get cheerful met thenicebot',
 'still happy na blast',
 'meant happy happy',
 'yeah paul glorious bb',
 'morning started amazing hopefully whole day going want go n greatday',
```

In [187]:
```
corpus2
```

Out[187]:
```
[[('just', 'RB'),
  ('got', 'VBN'),
  ('back', 'RB'),
  ('from', 'IN'),
  ('seeing', 'VBG'),
  ('garydelaney', 'NN'),
  ('in', 'IN'),
  ('burslem', 'NN'),
  ('amazing', 'NN'),
  ('face', 'NN'),
  ('still', 'RB'),
  ('hurts', 'VBZ'),
  ('from', 'IN'),
  ('laughing', 'VBG'),
  ('so', 'RB'),
  ('much', 'RB'),
  ('hilarious', 'JJ')],
 [('oh', 'UH'),
  ('dear', 'VBP'),
```

***STEMMING***

In [46]:
```
# TEST DATA
```

In [188]:
```
ps=PorterStemmer()
```

In [194]:
```
k=df1['text']
```

In [195]:
```
corpus1=[]
```

In [197]:
```
for i in k:
    review=re.sub('[^a-zA-Z]', ' ',i)
    review=review.lower()
    review=nltk.word_tokenize(review)
    review2=nltk.pos_tag(review) #pos tag
    review=[ps.stem(word) for word in review if not word in set(stopwords.wo
    review=" ".join(review)
    corpus1.append(review)
```

In [198]:
```
corpus1
```

```
 got offici jrjyp happi birthday jin young princejinyoungday happyjinyou
ngday got birthday',
 'got offici jrjyp happi birthday jin young princejinyoungday happyjinyou
ngday got happi birthday',
 'race advanc extra achiev individu individu charl town n optim inspir',
 'race advanc extra achiev individu individu charl town n inspir',
 'watch footbal match without commentari someth rejoic found transmiss ci
ti match like today joy',
 'twd come soon happi',
 'twd come soon',
 'taudeltaphidk thank obama cut elat back home',
 'ddogsscout oh almost odd cheer big boss offer muzzl flash blind acciden
t guy becam best friend',
 'gemma simmon bright spot premier far agentsofshield',
 'beauti day lord made rejoic glad',
 'watch amaz live ly broadcast kelli peterson live music come watch',
 'sometim like talk sad time want distract friend laughter shop eat n n m
hchat',
 'oi thewiggymess absolut fuck kill min later im still cri laughter grind
ah grindah hahahahahahaha',
```

In [52]:
```
# CONCATENATE DATA
```

In [53]:
```
k=df['text']
```

In [54]:
```
corpus=[]
```

In [145]:

```
for i in k:
    review=re.sub('[^a-zA-Z]', ' ',i)
    review=review.lower()
    review=nltk.word_tokenize(review)
    review=[ps.stem(word)for word in review if not word in set(stopwords.wor
    review=" ".join(review)
    corpus.append(review)
```

In [146]:

```
corpus
```

Out[146]:

```
['just got back from seeing  garydelaney in burslem  amazing   face still
hurts from laughing so much  hilarious',
 'oh dear an evening of absolute hilarity i don t think i have laughed so
much in a long time',
 'been waiting all week for this game         cheer  friday',
 ' gardiner love   thank you so much  gloria  you re so sweet  and though
tful  you just made my day more joyful  i love you too',
 'i feel so blessed to work with the family that i nanny for    nothing b
ut love  amp  appreciation  makes me smile',
 'today i reached      subscribers on yt      goodday   thankful',
 ' singaholic    good morning  love  happy first day of fall  let s make
some awesome  autumnmemories  annabailey  laughter  smile',
 ' bridgetjonesbaby is the best thing i ve seen in ages  so funny  i ve m
issed bridget   love   teammark',
 'just got back from seeing  garydelaney in burslem  amazing   face still
hurts from laughing so much',
 ' indymn i thought the holidays could not get any more cheerful  and the
n i met you   thenicebot',
 'i m just still   so happy   na blast',
```

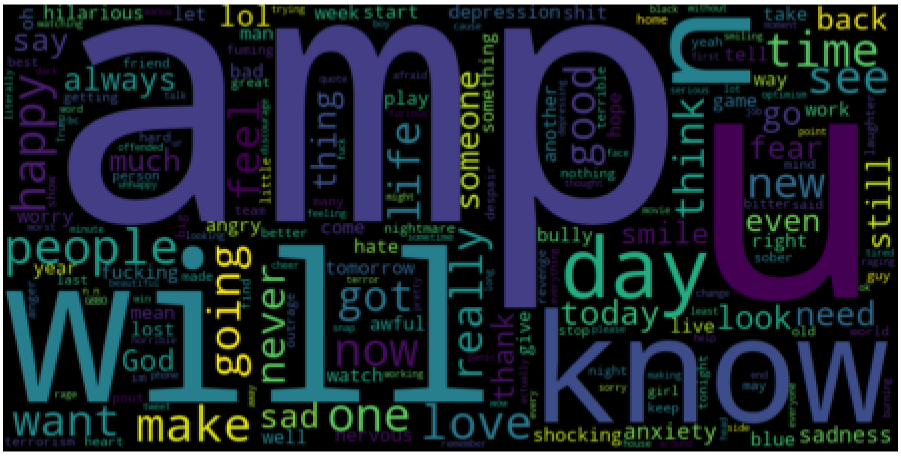
## CREATING WORD CLOUD

In [57]:

```
# TEST DATA
```

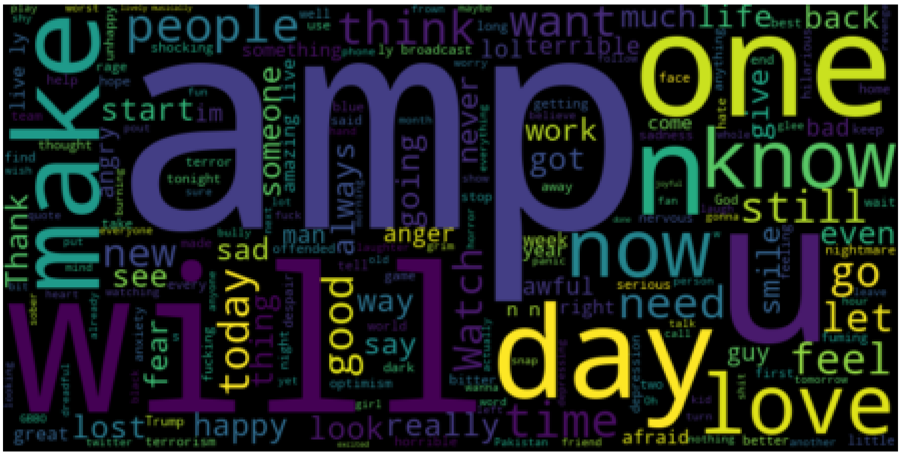
In [58]:

```
consolidated=' '.join(word for word in df1['text'])
wordCloud=WordCloud(width=800,height=400,random_state=21)
plt.figure(figsize=(8,6))
plt.imshow(wordCloud.generate(consolidated),interpolation='bilinear')
plt.axis('off')
plt.show()
```



```
In [59]: # CONCATENATE DATA
```

```
In [60]: consolidated=' '.join(word for word in df['text'])
         wordCloud=WordCloud(width=800,height=400,random_state=21)
         plt.figure(figsize=(8,6))
         plt.imshow(wordCloud.generate(consolidated),interpolation='bilinear')
         plt.axis('off')
         plt.show()
```



## VECTORIZATION

```
In [61]: from sklearn.feature_extraction.text import TfidfVectorizer
```

```
In [62]: tf=TfidfVectorizer()
         test=tf.fit_transform(df1['text'])
```

In [63]:
```
print(test)
```

```
  (0, 1449)	0.355356444129857
  (0, 3509)	0.23643392607058197
  (0, 233)	0.4578778126432787
  (0, 5019)	0.347518599668242
  (0, 984)	0.39770519574507834
  (0, 4450)	0.3697265222151111
  (0, 796)	0.19497942250360817
  (0, 5358)	0.3645062429637138
  (0, 9008)	0.14714152194601798
  (1, 6990)	0.39331436686573684
  (1, 8989)	0.39331436686573684
  (1, 2042)	0.20879188194193882
  (1, 465)	0.2573270631819589
  (1, 3026)	0.14938788140121761
  (1, 3163)	0.3742378911013824
  (1, 5760)	0.21167329518629158
  (1, 4083)	0.1239521564756093
  (1, 7128)	0.39331436686573684
  (1, 7331)	0.2409397980908342
  (1, 6956)	0.2849814725093943
  (1, 5747)	0.25356118359697577
  (2, 1111)	0.430794353444767
  (2, 3644)	0.28786687641504943
  (2, 992)	0.3932545180588725
  (2, 8642)	0.21509217198255673
  :	:
  (3140, 4104)	0.12577249579245192
  (3140, 425)	0.11010378821733718
  (3140, 8155)	0.10035036829116648
  (3140, 4083)	0.1180536924450097
  (3141, 1391)	0.29376446449286747
  (3141, 6916)	0.29376446449286747
  (3141, 4651)	0.26940714077505945
  (3141, 1303)	0.22296038924337885
  (3141, 6967)	0.24504981705725135
  (3141, 8326)	0.21856213260182947
  (3141, 5545)	0.24974213558333663
  (3141, 2011)	0.19979437249739382
  (3141, 5919)	0.26940714077505945
  (3141, 7625)	0.24504981705725135
  (3141, 3788)	0.27951634350002375
  (3141, 8046)	0.1867793671548767
  (3141, 7973)	0.20230545407872008
  (3141, 1812)	0.17860497546902526
  (3141, 934)	0.22798898009767848
  (3141, 580)	0.23080169606440756
  (3141, 5756)	0.1102253228416863
  (3141, 8014)	0.1659389952003304
  (3141, 8155)	0.0786960494958865
  (3141, 9016)	0.12842932448883232
  (3141, 9008)	0.09440280617574491
```

In [72]:
```
tf=TfidfVectorizer()
train=tf.fit_transform(df['text'])
```

```
In [73]: print(train)
```

```
  (0, 4395)     0.23310220666239753
  (0, 6325)     0.2096946440109472
  (0, 8822)     0.14629826308879437
  (0, 5389)     0.2771078565050305
  (0, 4563)     0.31884765304442064
  (0, 9083)     0.19954891519955026
  (0, 3306)     0.23811453671780058
  (0, 545)      0.2102437563981057
  (0, 1529)     0.34834496075581506
  (0, 4681)     0.12005886461039733
  (0, 3839)     0.34834496075581506
  (0, 8459)     0.2745279381633564
  (0, 3730)     0.34634453049410785
  (0, 902)      0.20808858097143562
  (0, 4028)     0.20756641957919614
  (0, 5117)     0.14935746867881194
  (1, 9697)     0.21246670897702713
  (1, 5649)     0.2673375446098878
  (1, 5388)     0.38386797302156317
  (1, 4282)     0.1630131686676567
  (1, 9617)     0.2162540365571959
  (1, 2807)     0.18396834608956925
  (1, 4396)     0.302523346202296
  (1, 270)      0.3323482265281695
  (1, 6814)     0.12431630003641325
  :     :
  (3958, 7170)  0.21608912460302723
  (3958, 2550)  0.30367522611781933
  (3958, 1544)  0.17157177881550423
  (3958, 4681)  0.13730328397689218
  (3958, 5117)  0.17081013553330512
  (3959, 4659)  0.29364435922771004
  (3959, 7764)  0.29364435922771004
  (3959, 1602)  0.29364435922771004
  (3959, 7369)  0.29364435922771004
  (3959, 2607)  0.29364435922771004
  (3959, 5192)  0.29364435922771004
  (3959, 2583)  0.1999623185070352
  (3959, 1173)  0.2797855427929576
  (3959, 2705)  0.2699525560477336
  (3959, 3221)  0.26232549898555746
  (3959, 6615)  0.14199598344455408
  (3959, 1939)  0.23240193643300475
  (3959, 9885)  0.1887781658368413
  (3959, 4509)  0.14942138772738692
  (3959, 1604)  0.12560415304166458
  (3959, 1041)  0.11263317995191867
  (3959, 6868)  0.11145444083442256
  (3959, 565)   0.14429642645217358
  (3959, 10537) 0.1226092395178388
  (3959, 8822)  0.11750461054224379
```

```
In [74]: y=df['label']
```

In [75]:
```
y
```

Out[75]:
```
0      2
1      2
2      2
3      2
4      2
      ..
342    3
343    3
344    3
345    3
346    3
Name: label, Length: 3960, dtype: int32
```

## MODEL CREATION

In [107]:
```
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score,classification_report
```

In [77]:
```
NB_model=MultinomialNB()
```

In [78]:
```
NB_model.fit(train,df['label'])
```

Out[78]:
```
MultinomialNB()
```

In [79]:
```
y_predict=NB_model.predict(train)
```

In [80]:
```
print(accuracy_score(df['label'],y_predict))
```

```
0.9209595959595596
```

In [81]:
```
print(classification_report(df['label'] , y_predict))
```

```
              precision    recall  f1-score   support

           0       0.97      0.94      0.95       941
           1       0.83      1.00      0.90      1257
           2       1.00      0.90      0.95       902
           3       0.98      0.82      0.89       860

    accuracy                           0.92      3960
   macro avg       0.94      0.91      0.92      3960
weighted avg       0.93      0.92      0.92      3960
```