Introduction:

Data is generated every day on a social networking site. Millions or billions of data are generated in a millisecond, not only in textual form, but also in emojis, symbols, numbers. Whenever we have to find out about people's feelings through comment about that particular thing, we can do data sentiment analysis using machine learning and NLP.

See the given below: -

```
Review  LikedWow... Loved this place.   1Crust is
not good.       0Not tasty and the texture was just
nasty.  0Stopped by during the late May bank
holiday off Rick Steve recommendation and loved it.
1The selection on the menu was great and so were
the prices.     1Now I am getting angry and I want
my damn pho.    0Honeslty it didn't taste THAT
fresh.) 0The potatoes were like rubber and you
could tell they had been made up ahead of time
being kept under a warmer.      0The fries were
great too.      1A great touch. 1Service was very
prompt. 1Would not go back.     0The cashier had no
```

In this article, our problem is the sentiment analysis on YouTube dataset. I discuss important library files, preprocessing steps, feature extraction methods, starting with some basic techniques which will lead into advanced Natural Language Processing techniques.

Let’s get started!

Table of Contents:

4. Basic Text Pre-processing of text data
    - Punctuation removal
    - Stopwords removal
    - Frequent words removal
    - Spelling correction
    - Tokenization
    - Stemming
    - Lemmatization
5. Advance Text Processing
    - N-grams
    - Term Frequency
    - Inverse Document Frequency
    - Term Frequency-Inverse Document Frequency (TF-IDF)
    - Sentiment Analysis

1. Importing Library files: -

First of all, import the important library files, which is used in entire sentimental analysis. Like NLTK, pandas, numpy, sklearn, textblob etc.

In above, pandas are used for importing CSV files (used in next step), nltk is use for natural language processing, TextBlob is built on the shoulders of NLTK and Pattern, sklearn is use for feature extraction.

Importing packages

In [1]:

```
import pandas as pd
import numpy as np
import nltk
from textblob import TextBlob
from nltk.stem import PorterStemmer
from textblob import Word
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer
```

2. Importing Datasets: -

Before starting, let's read the file from the dataset in order to perform different tasks on it. In the entire article, we will use the YouTube sentiment dataset from the Kaggle platform. (link is : https://www.kaggle.com/datasnaek/youtube#UScomments.csv)

Importing YouTube comments data

In [2]:
```
comm = pd.read_csv('UScomments.csv',error_bad_lines=False)#opening the file UScomments
comm.head()
```

```
b'Skipping line 41589: expected 4 fields, saw 11\nSkipping line 51628: expected 4 fields, saw 7\nSkipping line 114465: expected
4 fields, saw 5\n'
b'Skipping line 142496: expected 4 fields, saw 8\nSkipping line 189732: expected 4 fields, saw 6\nSkipping line 245218: expecte
d 4 fields, saw 7\n'
b'Skipping line 388430: expected 4 fields, saw 5\n'
C:\ProgramData\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py:3049: DtypeWarning: Columns (2,3) have mixed types.
Specify dtype option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

Out[2]:

	video_id	comment_text	likes	replies
0	XpVt6Z1Gjjo	Logan Paul it's yo big day !!!!!	4	0
1	XpVt6Z1Gjjo	I've been following you from the start of your...	3	0
2	XpVt6Z1Gjjo	Say hi to Kong and maverick for me	3	0
3	XpVt6Z1Gjjo	MY FAN . attendance	3	0
4	XpVt6Z1Gjjo	trending 😉	3	0

Here, **comm** is *dataframe* name, **pd.read_csv( )** is **pandas** function which is used in import the entire dataset. **UScomments.csv** is file name, which is kept in same file path. **error_bad_lines=False**,(here, false means *bad lines* will be dropped from the DataFrame). **comm.head()** is use for display the first five row of data frame.

3. Basic feature extraction using text data: -

➢ Check Null Values or Not

In [3]:
```
comm.isnull().sum()
```

Out[3]:
```
video_id         0
comment_text    25
likes            0
replies          0
dtype: int64
```

Here*, comment_text* have 25 null values. Need to replace the null values.

- **Replace Null Values**

null values are replaced by **Hello** (here, we can any word use for replacing the null values)

In [4]:
```
comm['comment_text'].fillna('Hello',inplace=True)
```

- **Check again null values removed or not**

In [5]:
```
comm.isnull().sum()
```

Out[5]:
```
video_id        0
comment_text    0
likes           0
replies         0
dtype: int64
```

Now, we can see that, no null values are available, means all set!

- **Number of words: -**

One of the most basic features we can extract is the number of words in each comment

In [6]:
```
comm['word_count']=comm['comment_text'].apply(lambda x: len(str(x).split(" ")))
comm[['comment_text', 'word_count']].head()
```

Out[6]:

	comment_text	word_count
0	Logan Paul it's yo big day ‼️‼️‼️	7
1	I've been following you from the start of your...	17
2	Say hi to Kong and maverick for me	8
3	MY FAN . attendance	4
4	trending 😉	2

- **Number of characters: -**

This feature is also based on the previous feature intuition. Here, we calculate the number of characters in each *comment_text*. This is done by calculating the length of the *comment_text*.

In [7]:

```
comm['char_count']=comm['comment_text'].str.len()
comm[['comment_text','char_count']].head()
```

Out[7]:

	comment_text	char_count
0	Logan Paul it's yo big day !!!!!	33
1	I've been following you from the start of your...	87
2	Say hi to Kong and maverick for me	34
3	MY FAN . attendance	19
4	trending 😉	10

➢ **Average word length: -**

We can also extract another feature which will calculate the average word length of each *comment_text*. It can also potentially help us in improving our model. Here, we simply take the sum of the length of all the words and divide it by the total length of the *comment_text*.

In [8]:

```
def avg_word(sentence):
    l = 0
    words = sentence.split()
    for word in words:
        l =l + len(word)

    if len(words) == 0:
        return 0
    else:
        return int(l/len(words))


comm['avg_word']=comm['comment_text'].apply(lambda x:avg_word(x))
comm[['comment_text', 'avg_word']].head()
```

Out[8]:

	comment_text	avg_word
0	Logan Paul it's yo big day !!!!!	3
1	I've been following you from the start of your...	4
2	Say hi to Kong and maverick for me	3
3	MY FAN . attendance	4
4	trending 😉	4

➢ **Number of stopwords:-**

Normally, solving an NLP problem, the first thing we do is to remove the stopwords. But sometimes calculating the number of stopwords can also give us some extra information which we might have been losing before. we already imported stopwords from NLTK (above).

In [9]:

```python

stop = stopwords.words('english')
comm['stopwords']=comm['comment_text'].apply(lambda x:len([x for x in x.split() if x in stop]))
comm[['comment_text', 'stopwords']].head()
```

Out[9]:

	comment_text	stopwords
0	Logan Paul it's yo big day !!!!!	1
1	I've been following you from the start of your...	9
2	Say hi to Kong and maverick for me	4
3	MY FAN . attendance	0
4	trending 😉	0

➢ **Number of special characters: -**

It is most interesting feature of NLP, which we can extract from a *comment_text* is calculating the number of hashtags or mentions present in it. This also helps in extracting extra information from our text data.

(Here, we make use of the ‘startswith’ function because hashtags (or mentions) always appear at the beginning of a word.)

In [10]:

```python
comm['hastags']=comm['comment_text'].apply(lambda x:len([x for x in x.split() if x.startswith('#')]))
comm[['comment_text', 'hastags']].head()
```

Out[10]:

	comment_text	hastags
0	Logan Paul it's yo big day !!!!!	0
1	I've been following you from the start of your...	0
2	Say hi to Kong and maverick for me	0
3	MY FAN . attendance	0
4	trending 😉	0

4. Basic Text Pre-processing of text data

➢ **Punctuation removal: -**

it doesn’t add any extra information while treating text data. Therefore, it will help us reduce t size of the training data.

In [11]:

```python
comm['comment_text']=comm['comment_text'].str.replace('[^\w\s]','')
comm['comment_text'].head()
```

Out[11]:

```
0                         Logan Paul its yo big day
1    Ive been following you from the start of your ...
2                   Say hi to Kong and maverick for me
3                                   MY FAN  attendance
4                                             trending
Name: comment_text, dtype: object
```

in above output, all the punctuation, including '#' '@',emoji, has been removed from the training data.

- **Stopwords removal: -**

We know, stop words (or commonly occurring words) should be removed from the text data

In [12]:
```
stop = stopwords.words('english')
comm['comment_text'] =comm['comment_text'].apply(lambda x:" ".join(x for x in x.split() if x not in stop))
comm['comment_text'].head()
```

Out[12]:
```
0                         Logan Paul yo big day
1    Ive following start vine channel seen 365 vlogs
2                          Say hi Kong maverick
3                             MY FAN attendance
4                                      trending
Name: comment_text, dtype: object
```

- **Frequent words removal: -**

We can also remove commonly occurring words from our text data First, here, check the 10 most frequently occurring words in our text data then take call to remove

In [13]:
```
freq =pd.Series(' '.join(comm['comment_text']).split()).value_counts()[:10]
freq
```

Out[13]:
```
I         217386
like       61940
love       44664
video      34985
Im         33755
The        31518
This       30998
one        29740
dont       26992
people     26614
dtype: int64
```

Now, remove those words as their presence will not of any use in classification of our text data.

- **Spelling correction: -**

spelling correction is a useful pre-processing step because this also will help us in reducing multiple copies of words. (with help of **textblob library**)

In [16]:
```
from textblob import TextBlob
comm['comment_text'][:10].apply(lambda x:str(TextBlob(x).correct()))
```

Out[16]:
```
0                              Began Paul to big day
1          Ve following start vine channel seen 365 logs
2                              May hi Long maverick
3                                 of FAN attendance
4                                          treading
5                                1 treading AYYEEEEE
6                                       end though
7                                        1 treading
8                           Happy year vlogaversary
9      You shit brother may single handed ruined YouT...
Name: comment_text, dtype: object
```

(We Know, sometime some people use mistake spelling, or use shortcuts also.)

- **Tokenization: -**

  Tokenization is use for dividing the text into a sequence of words or sentences.

```
In [17]: com=TextBlob(comm['comment_text'][1]).words # here tokenize the 1 index value
         com
Out[17]: WordList(['Ive', 'following', 'start', 'vine', 'channel', 'seen', '365', 'vlogs'])
```

In above output, used only index value of 1, you can use any index.

- **Stemming: -**

  Stemming is use for removal of suffices, like “ing”, “ional”, “ly”, “s”, etc. by a simple rule-based approach. we use **PorterStemmer** from the NLTK library**.**

```
In [18]: st = PorterStemmer()
         comm['comment_text'][:10].apply(lambda x: " ".join([st.stem(word) for word in x.split()]))
Out[18]: 0                           logan paul yo big day
         1        ive follow start vine channel seen 365 vlog
         2                           say hi kong maverick
         3                                  MY fan attend
         4                                          trend
         5                                1 trend ayyeeee
         6                                    end though
         7                                        1 trend
         8                         happi year vlogaversari
         9     you shit brother may singl handedli ruin youtu...
         Name: comment_text, dtype: object
```

  In above output, we can see **following** is **follow** and **trending** is **trend.**

- **Lemmatization: -**

  Lemmatization converts the word into its root word, it is more effective feature

```
In [19]: comm['comment_text'] =comm['comment_text'].apply(lambda x: " ".join([Word(word).lemmatize() for word in x.split()]))
         comm['comment_text'].head()
Out[19]: 0                      Logan Paul yo big day
         1    Ive following start vine channel seen 365 vlogs
         2                       Say hi Kong maverick
         3                           MY FAN attendance
         4                                    trending
         Name: comment_text, dtype: object
```

Now, Our Data is cleaned, all pre-processing steps are completed

5. Advance Text Processing: -

Now, we can move on to extracting features using NLP techniques.

- **N-grams :-**

N-grams are the combination of multiple words used together. Ngrams with **N=1** are called **unigrams**. Similarly, **bigrams** (N=2), **trigrams** (N=3) and so on can also be used.

Unigrams do not usually contain as much information as compared to bigrams and trigrams.

So, extract bigrams from our *comment_text* using the ngrams function of the textblob library.

```
In [20]:  1  TextBlob(comm['comment_text'][0]).ngrams(2)

Out[20]: [WordList(['Logan', 'Paul']),
          WordList(['Paul', 'yo']),
          WordList(['yo', 'big']),
          WordList(['big', 'day'])]
```

In above output, our sentence is **Logan Paul yo big day,** we can see the output of the that sentence

- **Term Frequency: -**

  Term frequency is simply the ratio of the count of a word present in a sentence, to the length of the sentence.

  Therefore, we can generalize term frequency as:

  TF = (Number of times term T appears in the particular row) / (number of terms in that row)

  Below, tried to show the term frequency table of a *comment_text*.

```
In [21]:  1  tf1= (comm['comment_text'][1:2]).apply(lambda x: pd.value_counts(x.split(" "))).sum(axis = 0).reset_index()
          2  tf1.columns =['words','tf']
          3  tf1
```

Out[21]:

	words	tf
0	365	1
1	vlogs	1
2	following	1
3	seen	1
4	channel	1
5	vine	1
6	start	1
7	Ive	1

- **Inverse Document Frequency: -**

The intuition behind inverse document frequency (IDF) is that a word is not of much use to if it's appearing in all the documents.

IDF = log(N/n)

where**, N** is the **total number of rows** and **n** is the **number of rows** in which the word was present**.**

So, calculate IDF for the same comment_text for which we calculated the term frequency.

In [22]:
```
for i, word in enumerate(tf1['words']):
    tf1.loc[i, 'idf']=np.log(comm.shape[0]/(len(comm[comm['comment_text'].str.contains(word)])))
tf1
```

Out[22]:

	words	tf	idf
0	365	1	8.755126
1	vlogs	1	7.247995
2	following	1	7.278957
3	seen	1	4.930682
4	channel	1	4.555100
5	vine	1	7.447537
6	start	1	4.614178
7	lve	1	4.409535

Here, more the value of IDF, means that more unique is the word.

- **Term Frequency-Inverse Document Frequency (TF-IDF): -**

TF-IDF is the multiplication of the TF and IDF which we calculated above

In [23]:
```
tf1['tfidf'] = tf1['tf']*tf1['idf']
tf1
```

Out[23]:

	words	tf	idf	tfidf
0	365	1	8.755126	8.755126
1	vlogs	1	7.247995	7.247995
2	following	1	7.278957	7.278957
3	seen	1	4.930682	4.930682
4	channel	1	4.555100	4.555100
5	vine	1	7.447537	7.447537
6	start	1	4.614178	4.614178
7	lve	1	4.409535	4.409535

- **Feature Extraction: -**

We don't have to calculate TF and IDF every time beforehand and then multiply it to obtain TF-IDF. so, sklearn has a separate function to directly obtain it use of :

from sklearn.feature_extraction.text import TfidfVectorizer (library)

In [24]:
```
tfidf = TfidfVectorizer(max_features=1000, lowercase=True, analyzer='word',
                        stop_words ='english',ngram_range=(1,1))
train_vect = tfidf.fit_transform(comm['comment_text'])
train_vect
```

Out[24]:
```
<691400x1000 sparse matrix of type '<class 'numpy.float64'>'
	with 2404336 stored elements in Compressed Sparse Row format>
```

- **Sentiment Analysis: -**

our problem was to detect the sentiment of the comment_text. So, before applying any ML/DL models, check the sentiment of the comment_text.

In [25]:
```
comm['comment_text'][:5].apply(lambda x:TextBlob(x).sentiment)
```

Out[25]:
```
0    (0.0, 0.1)
1    (0.0, 0.1)
2    (0.0, 0.0)
3    (0.0, 0.0)
4    (0.0, 0.0)
Name: comment_text, dtype: object
```

In [26]:
```
comm['sentiment']=comm['comment_text'].apply(lambda x:TextBlob(x).sentiment[0])
comm[['comment_text','sentiment']]
```

Out[26]:

	comment_text	sentiment
0	Logan Paul yo big day	0.000000
1	Ive following start vine channel seen 365 vlogs	0.000000
2	Say hi Kong maverick	0.000000
3	MY FAN attendance	0.000000
4	trending	0.000000
5	1 trending AYYEEEEE	0.000000
6	end though	0.000000
7	1 trending	0.000000
8	Happy year vlogaversary	0.800000
9	You shit brother may single handedly ruined Yo...	-0.135714
10	There mini Logan Paul	0.000000
11	Dear Logan really wanna get Merch money We eve...	0.200000

here, classification of sentiment is three type 0, 1, -1. now, we can apply the model