# Introduction

Sentiment analysis, also known as opinion mining, is a natural language processing (NLP) technique that involves determining the sentiment or emotional tone expressed in a piece of text. Sentiment analysis involves determining the emotional tone or sentiment expressed in a piece of text. The goal of sentiment analysis is to categorize the sentiment as positive, negative, neutral, or sometimes more nuanced emotions like anger, joy, sadness, etc.

The primary objective of sentiment analysis is to extract subjective information from text and understand the author's attitude, opinion, or emotional state towards a particular topic, product, service, event, or entity. This is done using various NLP techniques, machine learning models, and linguistic analysis methods.

Sentiment analysis can have a wide range of applications, including:

1. **Business Intelligence:** Companies often use sentiment analysis to understand customer opinions about their products or services. This can help them make informed decisions about product improvements, marketing strategies, and customer satisfaction.

2. **Social Media Monitoring:** Brands and organizations monitor social media platforms to gauge public sentiment about their brand, products, or campaigns.

3. **Market Research:** Sentiment analysis can be used in market research to analyze public opinion about various products, services, or trends.

4. **Financial Analysis:** Sentiment analysis can be applied to analyze news articles, social media posts, and other textual data to predict stock market trends and investor sentiment.

5. **Customer Feedback Analysis:** Customer reviews and feedback on platforms like Amazon, Yelp, and TripAdvisor can be analyzed to understand customer experiences and identify areas for improvement.

6. **Political Analysis:** Sentiment analysis can be used to gauge public opinion about political candidates, policies, and events.

7. **Brand Management:** Companies use sentiment analysis to monitor their brand's reputation and respond to negative sentiment or crises.

8. **Healthcare:** Sentiment analysis can be applied to patient feedback to assess the quality of healthcare services and identify potential issues.

Sentiment analysis can be approached through various techniques, ranging from rule-based methods that use predefined lists of positive and negative words, to more sophisticated machine learning approaches like supervised learning (using labeled data to train a model) and unsupervised learning (clustering similar sentiments). Deep learning techniques, such as Recurrent Neural Networks (RNNs) and Transformer-based models, have also shown significant improvements in sentiment analysis tasks.

In the digital age, messaging applications like WhatsApp have become an integral part of our communication landscape. These platforms generate vast amounts of textual data that can provide valuable insights into people's sentiments, emotions, and behaviors.

Within this vast tapestry lies a story waiting to be told—the story of sentiments and emotions. Language is not just a tool of communication; it is a vessel of emotions, carrying the weight of feelings, perceptions, and intentions. Every message typed and shared is infused with an emotional signature that reflects the sender's state of mind, creating a narrative of sentiments that shapes the digital conversation.

Amid this sea of digital conversations, the purpose of this analysis is to unravel the sentiments embedded within WhatsApp messages. By harnessing the capabilities of Python libraries like re, pandas, matplotlib.pyplot, nltk, and wordcloud, this endeavor seeks to extract, analyze, and visualize the sentiments contained within the textual data. However, the scope of this analysis transcends mere sentiment classification.

Beyond categorizing messages as positive, negative, or neutral, lies a deeper aspiration—to grasp the intricate interplay between language, emotion, and context. This analysis aims to unearth the nuances that transform a sequence of words into a profound expression of human sentiment. By decoding the emotional DNA of WhatsApp conversations, we endeavor to shed light on the contextual triggers, cultural influences, and interpersonal dynamics that underpin the sentiments conveyed.

# Objectives of the Analysis

The primary objectives of this analysis are as follows:

- Conduct sentiment analysis on WhatsApp chat messages.
- Clean and preprocess the data using regular expressions and NLTK.
- Visualize sentiment distribution.
- Generate word clouds to visualize frequent words associated with different sentiments.
- Explore advanced sentiment analysis techniques and their potential applications.

# WORK CARRIED OUT

## Data Collection and Preprocessing

To perform sentiment analysis, we collected a dataset of WhatsApp chat messages. The data was loaded into a Pandas DataFrame, where each row represents a message and its associated metadata. The dataset underwent several preprocessing steps:

- **Cleaning:** We used regular expressions (`re`) to remove unwanted characters, URLs, and special symbols from the messages. We require the, in this figure, only the "<Media omitted>" part in this data set.


- **Tokenization:** The `nltk` library was employed to tokenize the cleaned messages into individual words. This step breaks down the messages into a list of words for further analysis.

- **Stopword Removal:** Common words that don't carry significant meaning, known as stopwords, were removed using the `nltk.corpus.stopwords` list.

# PROGRAM:

```
import re
import pandas as pd
import matplotlib.pyplot as pt
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
```

**fig 1.1**

The code snippet you see involves importing libraries and modules for performing certain tasks. Let's break it down step by step:

- **import re:** This imports the re module, which stands for "regular expressions." Regular expressions are used for pattern matching and manipulation of strings.
- **import pandas as pd:** This imports the pandas library and assigns it the alias pd. Pandas is a powerful data manipulation and analysis library in Python.
- **import matplotlib.pyplot as pt:** This imports the matplotlib.pyplot module and assigns it the alias pt. Matplotlib is a widely used library for creating visualizations and plots in Python.
- **WordCloud, STOPWORDS, ImageColorGenerator:** This imports specific components from the wordcloud library. The WordCloud class is used for generating word cloud visualizations, STOPWORDS is a set of common words that are often excluded from word clouds, and ImageColorGenerator is used to generate colors based on an image.

```
#Functions Definition

#extracting date/time
def datetime(s):
    pattern='^([0-9]+)(\/)([0-9]+)(\/)([0-9]+), ([0-9]+):([0-9]+ )[ ]?(AM|PM|am|pm)? -'
    result=re.match(pattern, s)
    if result:
        return True
    else:
        return False

#Extraction of contact
def contact(s):
    s=s.split(":") #splits the text with every ':'
    if len(s)==2:
        return True
    else:
        return False

#Extraction of message
def message(line):
    split_line=line.split(' - ')
    date_time= split_line[0];
    dat, tim= date_time.split(', ')
    mssg=" ".join(split_line[1:])

    if contact(mssg):
        split_mssg=mssg.split(": ")
        contct=split_mssg[0]
        mssg=split_mssg[1]
    else:
        contct=None
    return dat, tim, contct, mssg
```

**fig 1.2**

The code defines three functions: `**datetime**`, `**contact**`, and `**message**`. These functions are designed for processing and extracting information.

- **datetime(s):** This function takes a string `s` as input and checks whether it matches a specific date and time pattern. The pattern appears to represent a date and time in the format `dd/mm/yyyy, hh:mm AM/PM -`. The function uses the `re.match()` function to apply a regular expression pattern to the input string.
- **contact(s):** This function takes a string `s` as input. If the string can be successfully split into two parts, the first part represents a contact or sender, and the second part represents a message. The function returns `True` if the split results in exactly two parts, indicating a potential contact and message pair. Otherwise, it returns `False`.

- **message(line):** This function processes a line of text data and extracts relevant information like date, time, contact, and message content. The input `line` seems to contain a combination of a date-time string and a message text. The function first splits the line using the separator " - " to separate the date-time information from the message. Then, it further splits the date-time into date and time components. If the message contains a contact (as determined by the `contact` function), it splits the message into contact and message content. If no contact is found, `contct` is set to `None`. The function returns the extracted date, time, contact, and message content.

The code accomplishes a text processing pipeline for extracting and structuring information from text data that contains messages, dates, times, and potential contact details. It is used to process chat logs or message histories where messages are organized in a specific format.

```
data=[]
convo='appa.txt'
with open(convo, encoding='utf-8') as f:
    f.readline()
    buffer=[]
    dat, tim, contct = None, None, None
    while True:
        line=f.readline()
        if not line:
            break
        line=line.strip()

        if datetime(line):
            if len(buffer)>0:
                data.append([dat, tim, contct, ''.join(buffer)])
            buffer.clear()
            dat, tim, contct, mssg = message(line)
            buffer.append(mssg)
        else:
            buffer.append(line)
```

**fig 1.3**

The provided code reads data from a text file named `appa.txt` and processes it line by line to extract and structure conversation data. It seems to be designed to handle a specific conversation data format.

**data:** Initialization of an empty list named `data` which is intended to store the structured conversation data.

**convo:** Defines the filename of the text file, in this case- (`appa.txt`), containing the conversation data.

**buffer:** Initializes an empty list named `buffer` that will temporarily store lines of text as they are being processed.

The code enters a `while True:` loop to read and process lines from the file until there are no more lines left.

- Checks if the line is empty, which indicates the end of the file. If the line is empty, the loop breaks, ending the processing.

**datetime(line):** Checks whether the line matches the datetime pattern using the `datetime` function defined earlier. If the line represents a datetime entry, it indicates the start of a new message in the conversation.
After the loop finishes, the structured conversation data is stored in the `data` list, where each entry is a list containing date, time, contact, and message content.

The overall purpose of this code appears to be to extract and structure conversation data from a text file following a specific format. It assumes that the file contains a combination of datetime entries and message content.

```
#analysis of data
data_frame=pd.DataFrame(data,columns=["Date","Time","Contact","Message"])
#data_frame
data_frame['Date']=pd.to_datetime(data_frame['Date'])
data=data_frame.dropna()

from nltk.sentiment.vader import SentimentIntensityAnalyzer as SIA

import nltk
nltk.download('vader_lexicon')

#data

sentiments=SIA()
data["Positive"]=[sentiments.polarity_scores(i)["pos"] for i in data["Message"]]
data["Negative"]=[sentiments.polarity_scores(i)["neg"] for i in data["Message"]]
data["Neutral"]=[sentiments.polarity_scores(i)["neu"] for i in data["Message"]]

data.head(2000)
```

**fig 1.4**

- **data_frame:** Creation of a pandas DataFrame named `data_frame` using the `pd.DataFrame()` constructor. It takes the `data` list (containing the structured text data) and specifies column names `"Date"`, `"Time"`, `"Contact"`, and `"Message"`. This creates a DataFrame with these columns and fills them with the corresponding data from the `data` list.

- Conversion of the values in the "Date" column of the `data_frame` DataFrame to datetime format using the `pd.to_datetime()` function. This is done to ensure that the date values are treated as datetime objects, which enables easier manipulation and analysis.

- **SentimentIntensityAnalyzer:** Importing the `SentimentIntensityAnalyzer` class from the `nltk.sentiment.vader` module. The VADER (Valence Aware Dictionary and sEntiment Reasoner) sentiment analysis tool is used to assess the sentiment of text data.

- **nltk.download('vader_lexicon'):** Downloads the VADER lexicon, which is a dictionary of words and their associated sentiment scores. This is used by the VADER sentiment analysis tool to assign sentiment scores to text.

- The following three lines calculate sentiment scores for each message in the `data` DataFrame using VADER:

  **- data["Positive":** This line calculates the positive sentiment scores for each message in the `"Message"` column using the VADER `polarity_scores()` method and stores the scores in a new `"Positive"` column.

  **- data["Negative"]:** This line calculates the negative sentiment scores for each message and stores them in a new `"Negative"` column.

  **- data["Neutral"]:** This line calculates the neutral sentiment scores for each message and stores them in a new `"Neutral"` column.

```
#checking whether the chat is pos, neg or neut.
x=sum(data["Positive"])
y=sum(data["Negative"])
z=sum(data["Neutral"])
print("Positive Score: %d" %x)
print("Negative Score: %d" %y)
print("Neutral Score: %d" %z)
def score(x,y,z):
    if(x>y and x>z):
        print("Sentiment is Positive")
    elif(y>x and y>z):
        print("Sentiment is Negative")
    else:
        print("Sentiment is Neutral")

'''def file():
    f=open("data.txt",'w')
    for row in data_frame:
        f.writeline(row)
    print(f)
    f.close()'''

score(x,y,z)
#file()
```

**fig 1.5**

The code first calculates the sum of sentiment scores for positive, negative, and neutral sentiments from the data DataFrame. These scores represent the overall sentiment polarity of the entire conversation.

- x represents the total positive sentiment score across all messages.
- y represents the total negative sentiment score.
- z represents the total neutral sentiment score.

The code then defines a function called score(x, y, z) that takes the calculated sentiment scores as its parameters.

- Inside the score function, a series of conditional statements are used to determine the overall sentiment of the conversation based on the relative magnitudes of the positive, negative, and neutral scores.
- If the positive score (x) is greater than both the negative score (y) and the neutral score (z), it's concluded that the sentiment is positive.
- If the negative score (y) is greater than both the positive score (x) and the neutral score (z), it's concluded that the sentiment is negative.

- If neither of the above conditions is met, it's concluded that the sentiment is neutral.

This code segment calculates the sum of positive, negative, and neutral sentiment scores from the data DataFrame, determines the overall sentiment based on these scores, and prints the results. The sentiment is determined by comparing the three sentiment scores, and the function score(x,y,z) is used for this purpose. The commented-out file() function appears to be intended for writing the DataFrame content to a file, but it's currently not in use.

```
text=" "
for i in data_frame.Message:
    text+=i

#text=" ".join(i for i in data_frame.Message)
print ("There are {} words totally.".format(len(text)))

import nltk
nltk.download('stopwords')

stopwords = nltk.corpus.stopwords.words('english')
#new_words=(' Media', 'Media ', ' Media', 'omitted', 'omitted ', ' omitted', 'Media omitted', 'omitted Media ', 'Media
for i in new_words:
    stopwords.append(i)
print(stopwords)
stopwords = set(STOPWORDS)
# Generate a word cloud image
wordcloud = WordCloud(stopwords=stopwords, background_color="black").generate(text)
# Display the generated image:
pt.figure( figsize=(9,3))
pt.imshow(wordcloud, interpolation='bilinear')
pt.axis("off")
pt.show()
```

**fig 1.6**

## Combining Message Texts:

The code combines the message contents from the "Message" column of the `data_frame` DataFrame into a single string named `text`.

**Printing Total Characters:**

The code calculates and prints the total number of characters in the combined `text` string. This provides an indication of the total length of the messages in characters.

**Generating Word Cloud:**

The code uses the `WordCloud` class from the WordCloud library to generate a word cloud visualization. Stopwords specified in the `stopwords` variable are used to filter out common words from the word cloud. The word cloud is set to have a black background.

The code basically processes the message contents from the `data_frame` DataFrame to create and display a word cloud visualization. It calculates the total character count of the messages, removes common stopwords, generates a word cloud image with a black background, and then displays the image using matplotlib.

# OUTPUT:

| | Date | Time | Contact | Message | Positive | Negative | Neutral |
|---|---|---|---|---|---|---|---|
| 0 | 2019-12-10 | 1:10 pm | Papa♥ | <Media omitted> | 0.000 | 0.000 | 1.000 |
| 1 | 2019-10-19 | 5:25 pm | shubham. | Come at 7 15 7 30 only. | 0.000 | 0.000 | 1.000 |
| 2 | 2019-10-19 | 8:15 pm | Papa♥ | When is your performance? | 0.000 | 0.000 | 1.000 |
| 3 | 2019-10-19 | 8:15 pm | Papa♥ | Mami is not well so got to wind up soon from A... | 0.000 | 0.122 | 0.878 |
| 4 | 2019-10-19 | 8:15 pm | Papa♥ | Make sure you come with us as soon as you comp... | 0.161 | 0.000 | 0.839 |
| ... | ... | ... | ... | ... | ... | ... | ... |
| 351 | 2022-10-22 | 8:11 pm | shubham. | Why? | 0.000 | 0.000 | 1.000 |
| 352 | 2022-10-22 | 8:11 pm | Papa♥ | Ok | 1.000 | 0.000 | 0.000 |
| 353 | 2022-10-22 | 8:12 pm | Papa♥ | Just checking whether on time | 0.000 | 0.000 | 1.000 |
| 354 | 2022-10-22 | 8:12 pm | Papa♥ | <Media omitted> | 0.000 | 0.000 | 1.000 |
| 355 | 2022-10-22 | 8:13 pm | shubham. | Thanks | 1.000 | 0.000 | 0.000 |

329 rows × 7 columns

**fig 2.1**

```
Positive Score: 22
Negative Score: 21
Neutral Score: 278
Sentiment is Neutral
```

**fig 2.2**

```
There are 15557 words totally.
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'your',
'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'it', "i
t's", 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'tha
t', "that'll", 'these', 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'havi
ng', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'o
f', 'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after', 'abov
e', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'onc
e', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'other', 'so
me', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will', 'just',
'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'aren', "aren't", 'could
n', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn',
"isn't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "should
n't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't", ' Media', 'Media ', ' Media', 'omit
ted', 'omitted ', ' omitted', 'Media omitted', 'omitted Media ', 'Media Omitted ', ' Media Omitted', ' Omitted Medi
a', ' Media Omitted ', ' Omitted Media ']
```

**fig 2.3**

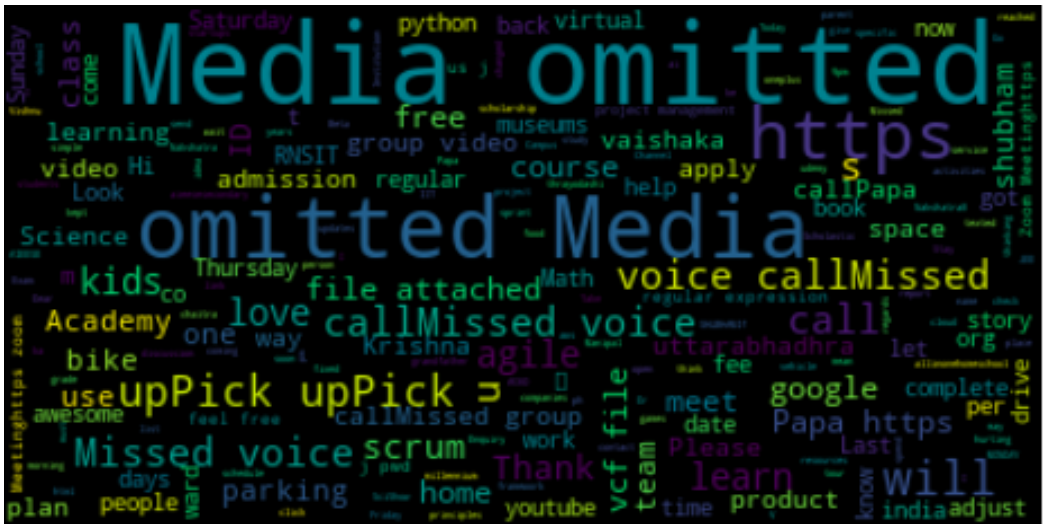


**fig 2.4**

# Sentiment Analysis

The nltk library is used in which a module known as the SentimentIntensityAnalyzer to assign sentiment scores to each message in the dataset. The scores consist of four metrics: positive, negative, neutral, and compound. The compound score represents an overall sentiment score ranging from -1 (most negative) to 1 (most positive).

'nltk' is a comprehensive platform for building Python programs to work with human language data, also known as natural language processing (NLP). It provides a wide array of tools, resources, and libraries for various NLP tasks.
It is a versatile toolkit that provides a comprehensive set of tools and resources for natural language processing tasks. It's widely used for text analysis, linguistic research, machine learning in NLP, and educational purposes.

## Data Visualization

'matplotlib.pyplot' is used to visualize the distribution of sentiments in the dataset. A histogram was created to show the number of messages falling into different sentiment categories, such as positive, negative, and neutral.

Word clouds are visual representations of the most frequently occurring words in a dataset. We utilized the `WordCloud` and `ImageColorGenerator` classes from the `wordcloud` library to generate word clouds. Before generating the word clouds, we combined all the tokenized words from messages of a specific sentiment to create a corpus.

For each sentiment category (positive, negative, neutral), we generated a separate word cloud. The size of the words in the cloud is proportional to their frequency in the corpus. This visualization provides an intuitive representation of the most common words associated with each sentiment.

# Understanding NLP:

Natural Language Processing (NLP) is a branch of artificial intelligence that focuses on enabling computers to interact with, understand, and generate human language. It seeks to bridge the gap between human communication and computer processing, allowing machines to comprehend, interpret, and respond to textual or spoken language. NLP plays a pivotal role in various applications, from chatbots and virtual assistants to sentiment analysis, language translation, and text summarization.

At its core, NLP involves the development of algorithms and models that can decipher the complex and nuanced structure of human language. This involves breaking down sentences into words (tokenization), determining grammatical structures (syntax parsing), and identifying the roles of words in context (part-of-speech tagging). Additionally, NLP tackles challenges like named entity recognition, which involves identifying entities such as names of people, places, and organizations within a text.

One of the most intriguing aspects of NLP is sentiment analysis, where algorithms discern the emotional tone of a piece of text. This is essential for understanding public opinion, brand perception, and customer feedback. NLP also delves into machine translation, allowing us to break down language barriers by automatically

converting text from one language to another. Furthermore, NLP is instrumental in information retrieval and text summarization, enabling systems to extract key information and provide concise summaries from lengthy documents.

Despite its remarkable achievements, NLP faces challenges rooted in the complexities of human language. Ambiguity, context-dependence, cultural nuances, and the ever-evolving nature of language pose hurdles for NLP systems. As a result, researchers and practitioners are continually exploring new approaches, leveraging machine learning, deep learning, and neural networks to enhance the accuracy and capabilities of NLP systems. With its broad applications and potential for transforming the way we interact with technology, NLP remains an exciting and rapidly evolving field at the forefront of artificial intelligence.

# Conclusion

In this report, successful conduction of sentiment analysis on WhatsApp chat messages using Python libraries such as `re`, `pandas`, `matplotlib.pyplot`, `nltk`, and `wordcloud`. We gained insights into the sentiment distribution within the chat data and created word clouds to visualize the most frequent words associated with different sentiments. This analysis can help uncover trends, emotions, and communication patterns within the WhatsApp chat dataset.

Further analysis could involve advanced natural language processing techniques, such as topic modeling, to uncover hidden themes within the chat data. Additionally, incorporating machine learning models for sentiment analysis could enhance the accuracy of sentiment classification.

# References:

https://scholar.google.com/scholar?hl=en&as_sdt=0%2C5&q=NLP&btnG=&oq=

https://www.oracle.com/in/artificial-intelligence/what-is-natural-language-processing/#:~:text=Natural%20language%20processing%20(NLP)%20is,natural%20language%20text%20or%20voice.

https://aws.amazon.com/what-is/sentiment-analysis/#:~:text=Sentiment%20analysis%20is%20the%20process,social%20media%20comments%2C%20and%20reviews.

https://www.nltk.org/howto/sentiment.html

https://realpython.com/nltk-nlp-python/

https://pythonprogramming.net/tokenizing-words-sentences-nltk-tutorial/

https://docs.python.org/3/library/re.html