WHATSAPP ANALYZER AND LANGUAGE TRANSLATOR USING PYTHON

PROJECT PHASE II REPORT

Submitted by

ABISHEK P	**(714019104003)**
ARUN KUMAR V	**(714019104016)**
KABILESH KUMAR S	**(714019104043)**
LOGESH S	**(714019104052)**

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

SRI SHAKTHI INSTITUTE OF ENGINEERING

AND TECHNOLOGY (AUTONOMOUS),

COIMBATORE 641 062

Autonomous Institution,

Accredited by NAAC with "A" Grade

DECEMBER 2022

BONAFIDE CERTIFICATE

Certified that this project report **"WhatsApp Chat Analyzer and Language Translation Using Python"** is the Bonafide work of **"Abishek P (714019104003), Arun Kumar V (714019104016), Kabilesh Kumar S (714019104043), Logesh S (714019104052)"**, who carried out the work under my supervision. Certified further that to the best of my knowledge the work reported herein does not from part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE

Dr.K.E.KANNAMMAL

HEAD OF THE DEPARTMENT

Professor and Head,

Department of CSE,

Sri Shakthi Institute of

Engineering and Technology,

Coimbatore- 641 062.

SIGNATURE

Ms.M.MOHANA PRIYA

SUPERVISOR

Assistant Professor,

Department of CSE,

Sri Shakthi Institute of

Engineering and Technology,

Coimbatore- 641 062.

Submitted for the project work viva-voce Examination held on……

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

First and foremost, I would like to thank God Almighty for giving me the strength. Without his blessings this achievement would not have been possible.

We express our deepest gratitude to our **Chairman Dr.S.Thangavelu** for his continuous encouragement and support throughout our course of study.

We are thankful to our **Secretary Er.T.Dheepan** for his unwavering support during the entire course of this project work.

We are also thankful to our **Joint Secretary Mr.T.Sheelan** for his support during the entire course of this project work.

We are highly indebted to **Principal Dr.A.R.Ravi Kumar** for his support during the tenure of the project.

We are deeply indebted to our **Head of the Department**, Computer Scienceand Engineering, **Dr.K.E.Kannammal**, for providing us with the necessary facilities.

It's a great pleasure to thank our **Project Guide Ms.M.MohanaPriya** for her valuable technical suggestions and continuous guidance throughout this project work.

We are also thankful to our **Project Coordinator Mr.R.Karthiban** and **Mrs.M.Banupriya** for providing us with necessary facilities and encouragement.

We solemnly extend our thanks to all the teachers and non-teaching staff ofour department, family and friends for their valuable support.

ABISHEK P

ARUN KUMAR V

KABILESH KUMAR S

LOGESH S

ABSTRACT

The most used and efficient method of communication in recent time is WhatsApp application. WhatsApp chat consists of various kinds of conversations held among group of people. This chat consists of various topics. This information can provide lots of data for latest technologies such as machine learning. The most important thing for NLP model is to provide the right learning experience which is indirectly affected by the data that we provide to the model. This tool aims to provide in depth analysis of this data which is provided by WhatsApp. Irrespective of whichever topic the conversation is based our developed code can be applied to obtain a better understanding of the data. The advantage of this tool is that is implemented using simple python libraries such as pandas, matplotlib, seaborn and sentiment analysis which are used to create data frames and plot different graphs, which is efficient and less resources consuming algorithm; therefore, it can be easily applied to largest dataset. This tool makes a better understanding of the personal usage of WhatsApp to the users. It displays the chat count, chat Average, Member/Sender Rank, Website /URL/Link Domain Rank, Word Count and Rank, Most Used Word by Sender, Emoji Usage Rank, Most Used Emoji by Sender, Timestamp Heatmap, Attachment Classification. The tool identifies the behavior of the chat that is inappropriate text and warn them. This tool also has translator which can translate messages that the user can understand.

This scenario explores the use of a WhatsApp webhook to dynamically check for vulgar messages, trigger a mail to a specified person, and translate the message to a user-specified language. By integrating NLP, ESP, and translation APIs into a web application that can receive the WhatsApp webhook payload, we can maintain the quality of messages exchanged on the platform and provide a better user experience. This approach can help businesses and individuals ensure that the messages they receive on WhatsApp are appropriate and can be understood by all parties involved, regardless of language barrier.

Using a WhatsApp webhook to check for vulgar messages, trigger a mail, and translate messages can also enhance communication efficiency for businesses. For example, a business with customers from different countries can use this approach to ensure that they can understand and respond to their customers' messages in a language they prefer. This can help the business provide better customer service and increase customer satisfaction.

Furthermore, this approach can be customized to suit specific business needs. For instance, instead of triggering a mail to a specified person, the web application can notify a group of moderators who can take appropriate action on the message. Additionally, the translation API can be configured to automatically detect the source language of the message, reducing the need for users to specify the language manually.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER I
INTRODUCTION

1.1 Text Analysis

This tool is based on data analysis and processing. The first step in implementing a machine learning algorithm is to understand the right learning experience from which the model starts improving on. Data pre-processing plays a major role when it comes to machine learning. WhatsApp claims that nearly 55 billion messages are sent each day. The average user spends 195 minutes per week on WhatsApp, and is a member of plenty of groups.

With this treasure house of data right under our very noses, it is but imperative that we embark on a mission to gain insights on the messages which our phones are forced to bear witness to. WhatsApp-Analyzer is a statistical analysis tool for WhatsApp chats. Working on the chat files that can be exported from WhatsApp it generates various plots showing, for example, which another participant a user responds to the most.

We propose to employ dataset manipulation techniques to have a better understanding of WhatsApp chat present in our phones. In this decade the upcoming technologies are mainly dependent on data. Since a lot of machine learning enthusiasts develop models which helps solve multiple problems the requirements of appropriate data are very large scale this project aims to provide a better understanding towards various types of chats. This analysis proves to be better input to machine learning models which essentially explore the chat data.

These models require proper learning instances which provides better accuracy for these models. Our project ensures to provide an in-depth exploratory data analysis on various types of WhatsApp chats and this tool provide a Safe Chat environment in social messages. Data pre-processing, the initial part of the project is to understand implementation and usage of various python-built modules. The above process helps us to understand why different modules are helpful rather than implementing those functions from scratch by the developer. These various modules provide better code representation and user understandability.

The following libraries are used such as NumPy, SciPy pandas, csv, matplotlib, sys, re, emoji, seaborn etc. Exploratory data analysis, first step in this to apply a algorithm which provides positives negative and neutral part of the chat and is used to plot pie chart based on these parameters. To plot a line graph which shows author and message count of each date, to plot a line graph which shows author and message count of each author, ordered graph of date vs message count, media sent by senders and their analysis report including behavior analysis.

1.2 Translation to Secondary language

The concept of culture provides a consistent basis for translation irrespective of the different alternative approaches to translation (Ginter, 2002). The word 'translation' has been defined as the representation of culture-specific communication. A translator is regarded as the "first reader", who is responsible for the production of specific meanings that are going to be perceived by the cultural community. Therefore, the translator is also expected to represent the primary process of translation. During the process of translation, the association between two or more cultures results in the production of hybrid text. The text appears by adapting few features of the translated text in the source culture (Loffredo & Perteghella, 2006). The translation of any text into a foreign language provides an instance for inter-cultural communication. The existence of literary text between different cultures is an outcome of negotiations between different languages that may include features, contradicting the target cultural norms and languages.

The definition of translation has been revised over time as the particular culture plays a significant role in the process of translation. Wojtasiewicz (1992), the author of first Polish monograph and a Polish linguist, studied and investigated adequate cultural approaches to the process of translation. The phenomenon of translating text from language 'A' into language 'B' should provide the reader with similar association as the reader, who reads the text formulated in language 'A' (Wojtasiewicz, 1992). Therefore, translators are considered as those who can reduce the gap between two cultures. On the basis of proficiency, knowledge, and perception, culture has been known as a factor that may influence the social sense of an individual. These aspects can be associated with the advancement in intellectual development that can be reflected through art.

The cultural identity of different individuals is depicted through the common factual language that involves the educational status, religion, customs, political institutions, and current affairs (Ginter, 2002). The translator, who is learning to translate a text is considered responsible to produce meanings, which can be perceivable for the readers' cultural community (Wolf, 1997). The compliance of text with cultural issues is a major problem for the translator. The term 'hybrid text' has been known as a feature of contemporary inter-cultural communication that extensively contributes to various studies related to translation. However, translation is regarded as a creative and intellectual task that is responsible for discovering and creating meanings that are hidden between the lines of the text. Translation plays a significant part in filling the gap between different nations and cultures.

Literary translations particularly assist different states to reach a universal culture on a mutual ground. A good translation is concerned normally with transferring the propositional content of the language text, and also its other pragmatic features. Literary translation is recognized as a critical procedure through which the texts are imagined, created, and read. It is known as a highly artistic and creative practice, which requires proper attention. Therefore, this study has mainly focused on the process of translation as "the making of text". Moreover, it has also explored the creativity in translation of a text and the multi-vocal association between the translated text and the writer.

Translation plays a significant part in filling the gap between different nations and cultures. Literary translations particularly assist different states to reach a universal culture on a mutual ground. A good translation is concerned normally with transferring the propositional content of the language text, and also its other pragmatic features. Literary translation is recognized as a critical procedure through which the texts are imagined, created, and read. Therefore, this study has mainly focused on the process of translation as "the making of text". Moreover, it has also explored the creativity in translation of a text and the multi-vocal association between the translated text and the writer

The text produced after translation sometimes delivers no similar pattern and style between the original and translated texts, as the translation provides a new source of reading. The perception has been implied to be a major responsibility of the translator. For this purpose, the study has provided an inclusive view of the major strategies related to the quality assessment of literary translation and its current challenges.

1.3 Vulgar text detection:

Hate speech is currently of broad and current interest in the domain of social media. The anonymity and flexibility afforded by the Internet has made it easy for users to communicate in an aggressive manner. And as the amount of online hate speech is increasing, methods that automatically detect hate speech is very much required. Moreover, these problems have also been attracting the Natural Language Processing and Machine Learning communities a lot. Therefore, the goal of this paper is to look at how Natural Language Processing applies in detecting hate speech. Furthermore, this paper also applies a current technique in this field on a dataset.

As neural network approaches out performs existing methods for text classification problems, a deep learning model has been introduced, namely the Convolutional Neural Network. This classifier assigns each tweet to one of the categories of a Twitter dataset: hate, offensive language, and neither. The performance of this model has been tested using the accuracy, as well as looking at the precision, recall and F-score. The final model resulted in an accuracy of 91%, precision of 91%, recall of 90% and a F-measure of 90%. However, when looking at each class separately, it should be noted that a lot of hate tweets have been misclassified. Therefore, it is recommended to further analyze the predictions and errors, such that more insight is gained on the misclassification.

Over the last decade, the increased use of social media has led to an increase in hateful activities in social networks. Hate speech is one of the most dangerous of these activities, so users have to protect themselves from these activities from YouTube, Facebook, Twitter etc. This paper introduces a method for using a hybrid of natural language processing and with machine learning technique to predict hate speech from social media websites. After hate speech is collected, steaming, token splitting, character removal and inflection elimination is performed before performing hate speech recognition process. After that collected data is examined using a killer natural language processing optimization ensemble deep learning approach (KNLPEDNN). This method detects hate speech on social media websites using an effective learning process that classifies the text into neutral, offensive and hate language. The performance of the system is then evaluated using overall accuracy, f-score, precision and recall metrics. The system attained minimum deviations mean square error − 0.019, Cross Entropy Loss − 0.015 and Logarithmic loss L-0.0238 and 98.71% accuracy.

1.3.1 Related Work

Linguistic Rule-Based Approach

In 2014, C. J. Hutto et al. proposed an approach to classify sentiment using VADER, which is a rule-based approach. At first, they created a list of lexical features that are highly sensitive to the sentiment of social media posts. After then they combined that list of lexical features with five general rules that encapsulate syntactical and grammatical rules for presenting sentiment intensity. At last, they have found that VADER performed 96% accuracy using the rule-based model on Twitter sentiments. Dennis Gitariet al. in 2015 proposed a method to identify the Sentiment Analysis of the Social Media Text using the Rule-based method . In this work, They categorized the hate speech problem into three fields religion, nationality, and race. The main objective of this paper is to develop a classification model that employs sentiment analysis. The developed model not only detects subjective sentences but also classifies and ranks the polarity of sentiment phrases. After then they relate the semantic and subjective features with hate speech. Finally, they achieved 71.55 % precision using the lexicon-based approach.

Supervised Learning Approach

Fatahillah et al. (2017) used Naive Bayes Classifier Algorithm to detect hate speech on Instagram using the k-nearest neighbor classifier. They collected the data set using Twitter API from Twitter and annotated those data set manually. After preprocessing and feature engineering phase, they applied the Naive Bayes Classifier algorithm and found 93% of accuracy. M. Ali Fauzi et al. (2018) proposed an approach to identify hate speech using a set of supervised learning algorithms. They ensembled five different classification algorithms, including K-Nearest Neighbours, Random Forest, Naive Bayes, Support Vector Machine, and Maximum Entropy. They collected the data set using Twitter API and annotated those data set manually. In preprocessing phase, They employed tokenization, filtering, stemming, and term weighting methods. They utilized the bag of words features with TFIDF techniques. The naive Bayes algorithm performed best with 78.3 % of accuracy among all the other five stand-alone classifiers. In 2019, P. Sari et al. proposed an approach to detect hate speech using logistic regression on Twitter.They collected the data from Twitter and employed Case Folding,

Tokenizing, Filtering, and Stemming methods in preprocessing phase. After Pre-processing, the TF-IDF technique is used for vectorization. After Feature engineering, the Logistic regression algorithm has been applied, and they have found 84% of accuracy. In 2020, Oluwafemi Oriola et al. proposed an approach to detect offensive speech on tweeter. The author collected the data set using Twitter API and annotated those data set into two sections, free speech „FS‟ and hate speech „HT.‟ In preprocessing phase, they removed special characters, emojis, punctuations, symbols, engineering phase, they employed the TF-IDF technique to transform the text into feature vectors. After applying an optimized support vector machine with n-gram, they have found 89.4% of accuracy. In 2020, Annisa Briliani et al. proposed an approach to identify hate speech on Instagram using the knearest neighbor classifier. They collected the data set using Instagram API from Instagram and annotated those data set manually. They divided the dataset into 2 labels, namely zero and one. In preprocessing phase, they cleaned the data and employed the TF-IDF technique in the feature engineering phase. After then, they applied the k-nearest neighbor algorithm and found 98.13% of accuracy.

Unsupervised Learning Approach

Rui Zhao et al. (2015) proposed an approach to detect cyberbullying using Semantic-Enhanced Marginalized Denoising Auto-Encoder. They used two sources of data set. The first source is Twitter, and the second source is Myspace. Twitter data was collected through Twitter stream API, and Myspace data was collected using the web crawling technique. They have achieved 84.9 % accuracy using smSDA for the Twitter dataset, and they have got 89.7% of accuracy with smSDA with the MySpace dataset. Axel Rodríguez et al. (2019) proposed an approach to detect hate speech content using sentiment analysis on Facebook. They used Graph API to extract the post and comments from Facebook. To remove the unrelated texts VADER and JAMMIN were used. In preprocessing phase, they filtered out all unnecessary stopwords or symbols. Preprocessed documents converted into the vector using TFIDF. The resulting matrix is passed to the k-means clustering algorithm as an input matrix. The most negative articles and responses were collected using sentiment and emotion analysis. Sylvia Jaki et al. (2019) demonstrated an approach to detect hate speech content using unsupervised learning on Twitter. They computed three clusters of the top 250 most biassed terms using spherical k-means clustering and skip-grams. As a result, they have got an 84.21% F1 score. Michele Di Capua et al. (2019) proposed an approach to detect cyberbullying using unsupervised learning.

Deep Learning Approaches

Hugo Rosa et al. (2018) proposed an approach to detect cyberbullying using deep learning . In this paper, the training and testing data set was collected from Kaggle. At first, they initiated CNN, which holds a certain similarity to the issue of cyberbullying. It starts with a single-layer CNN and continues with a completely linked layer with a dropout of 0.5 and softmax performance. Then they combined CNN-DNN-LSTM to achieve maximum accuracy. Tin Van Huynh et al. (2019) proposed an approach to detect hate speech using Bi-GRU-CNN-LSTM Model. In this paper, they collected data from Twitter and categorized their data into three labels (OFFENSIVE, HATE, and CLEAN). After cleaning the data, they implemented three neural network models such as BiGRU-LSTM-CNN, Bi-GRU-CNN, and TextCNN to identify hate speech. They achieved a 70.57% of F1 score as a result. Gambäck et al. (2019) utilized a deep learning algorithm to detect hate speech on Twitter. In this paper, they collected data from Twitter and divided the data set into four categories(sexism, racism, combined(sexism and racism), and non-hate-speech). They employed four CNN models that were trained with character n-gram, word2vec, random vectors combined(word2vec and character n-gram). The author utilized a 10-fold technique to improve the accuracy of the model. Among all four models, word2vec based CNN model performed well with a 78.3% of F-score.

Hybrid based Approach

Viviana Patti et al. (2019) proposed a Hybrid based approach to detect hate speech. In this paper, they employed two models. In their first model, they implemented a linear support vector classifier (LSVC), and in the second model, they employed a long short-term memory (LSTM) neural model with word embedding. They concatenated 17 categories, such as HurtLex, with two types, namely LSVC and LSTM. Joint learning with a multilingual word embedding model, including HurtLex, performed best with 68.7% of F1-score

Safa Alsafari et al. (2020) proposed a Hate speech detection model for Arabic social media. In this paper, they collected the data set using Twitter search API, and the data set is categorized into four classes (Religious, Nationality, Gender, and Ethnicity).

WhatsApp Webhooks are a powerful way to integrate WhatsApp into your application or service. Webhooks enable you to receive real-time notifications of events on the WhatsApp Business API, such as incoming messages or delivery reports, and take action based on those events. In this article, we will explore what WhatsApp Webhooks are, how they work, and how you can use them to enhance your application or service.

What is a WhatsApp Webhook?

A WhatsApp webhook is a way for you to receive real-time notifications from the WhatsApp Business API. When an event occurs on the WhatsApp Business API, such as a new message or a delivery report, the WhatsApp server sends a webhook to your application or service. You can then take action based on the data contained in the webhook. For example, you could use a webhook to automatically respond to a customer's message or update your database when a message is delivered.

How do WhatsApp Webhooks work?

When you create a webhook, you specify a URL where the webhook should send its notifications. When an event occurs on the WhatsApp Business API, the server sends a POST request to the specified URL with a JSON payload containing information about the event. Your application or service can then process the JSON data and take action based on the event.

To use WhatsApp Webhooks, you must have a WhatsApp Business API account and a web server that can receive HTTP requests. You will also need to create a webhook endpoint on your web server that can receive POST requests and process the JSON data contained in the request.

Types of WhatsApp Webhooks:

There are several types of WhatsApp webhooks that you can use to receive notifications from the WhatsApp Business API. These include:

- Message Received webhook: This webhook is triggered when a new message is received on the WhatsApp Business API.
- Message Delivered webhook: This webhook is triggered when a message is delivered to the recipient.
- Message Read webhook: This webhook is triggered when the recipient reads the message.
- Account Status webhook: This webhook is triggered when the status of your WhatsApp Business API account changes, such as when your account is approved or rejected.
- Payment webhook: This webhook is triggered when a payment is initiated or completed on the WhatsApp Business API.

Using WhatsApp Webhooks:

To use WhatsApp webhooks, you will need to create a webhook endpoint on your web server that can receive and process the JSON data contained in the webhook payload. You will also need to configure your WhatsApp Business API account to send notifications to your webhook endpoint.

Once you have set up your webhook endpoint, you can start using WhatsApp webhooks to receive real-time notifications from the WhatsApp Business API. You can use these notifications to trigger actions in your application or service, such as automatically responding to customer messages or updating your database with delivery information.

CHAPTER -2
LITERATURE SURVEY

2.1 Textual Analysis | Guide

Author: Published on November 8, 2019 by Jack Caulfield. Revised on November 25, 2022.

As a demo Survey analysis on the usage and Impact of WhatsApp Messenger, Various Studies and analysis has been done on the usage and impact of WhatsApp. Some of these studies are for finding the impact of WhatsApp on the students and some are based on for the general public in a local region. In a study of southern part of India was conducted on the age group of between 18 to 23 years to investigate the importance of WhatsApp among youth. Though this study it was found that students spent 8 hours per day on using WhatsApp and remain online almost 16 hours a day.

All the respondents agreed that they are using WhatsApp for communicating with their friends. They also exchange images, audio and video files with their friends using WhatsApp. It was also proved that the only application that the youth uses when they are spending time on their smart phone is WhatsApp. Methods used in this survey is to analyze the intensity of WhatsApp usage and its popular services and to identify the degree of positive or negative impacts of using WhatsApp. Content Analysis of WhatsApp conversation.

The dataset of WhatsApp group chat used for analysis is of 1 year (may, 2015-may,2016) which consists of 5,5563 records in total and comprises of certain characteristics that define how much a particular person is using WhatsApp chat group, such as the years of usage, duration of usage in a day, the response levels, type of messages posted by each individual in the group (Smiley, Text, Multitude), which age group people are more active and so on. The main attributes set for this analysis are type of message been send, duration of use per year/month/week/day/hour, timestamp (AM/PM), age group of senders, gender (Male/Female).

RStudio the most favored IDE for R is been used to perform exploratory data analysis and visualization for the collected data largely because of its open-source nature. Forensic analysis of WhatsApp Messenger. WhatsApp provides its users with various forms of communications, namely user-to-user communications, broadcast messages, and group chats. When communicating, users may exchange plain text messages, as well as multimedia files (containing images, audio, and video), contact cards, and geolocation information.

Each user is associated with its profile, set of information that includes his/her WhatsApp name, status line, and avatar (a graphic file, typically a picture). The profile of each user is stored on a central system, from which it is downloaded by other WhatsApp users that include that user in their contacts.

2.2 International Journal Of English Language And Translation Studies

Authors: Syed Sarwar Hussain Department of Linguistics and Translation Studies College of Languages and Translation, King Saud University Riyadh, Kingdom of Saudi Arabia.

The word 'translation' was coined around the year 1340, and it has been known to be derived from either Old French or Latin. The Latin meaning of the word 'translation' is 'transporting' and this has been regarded as the basic meaning of the word for the purpose of rendering texts into another language. Translation is a process, where an originally written text, known as the source language, is translated into different languages. It is translated in the form of a properly written text known as the target text (Munday, 2016). Translating a literary text involves the reshaping of words from one language to another. The process of translation has been recognized as a subsidiary form of art. The derivative nature of the approach holds a significant character in determining the quality of the translated content. Along the years, the important nature of the work has been underestimated to an extent, where it resulted in the devaluing of the text and lowering of the perceived standards. Over the time, the incompetent translations have generated negative effects on the literature in practical terms. The text translation has always witnessed a substantial significance in the history of translation. According to the readers, the translation of Holy books like Bible, Quran and others, into local languages have been identified as big milestones in the course of time. Thus, it is recognized as an impactful conduct in terms of all the cultural, social, religious, and political approach (Bassnett, 2013).

The translator is known as an efficient personnel, who has the ability to produce text with the same impression as the original text. It has been, therefore, considered necessary for the literary translators to contemplate the aesthetic aspects of the translated text in order to maintain the style and laxity of the original content.

Therefore, the translators are expected to have acquired extensive artistic skills along with efficient language abilities (Fowler & Hodges, 2011). The competence of a translator in the performance of translating a text from source to target language must be highly proficient. It

is considered thus because of the obvious fact that the target text, which is finally shared with the target audience, is basically the perception and final resolutions of the translator towards the material. Thus, the translation is realized and regarded more as a product rather than the process (Zanettin, Bernardini & Stewart, 2014).

Pooja Sharma et. al. has presented a survey work that has been performed in the field of document scanning. Some other features include quality assessment methods and metrices for document images

Also, there are some work done related to the web application translation of text in any language with help of python language libraries and Google Translator is good example of such topics. There are various steps in this which include pre-processing of document, character recognition, segmentation, text extraction, detection and translation of language. These factors when evaluated properly results in image enhancement and better text recognition.

There are some problems in text recognition. OCR being an evolving technology, is not 100% accurate while recognizing text and human inspection is necessary. Zeev Zelavsky et. al. suggested an algorithm for recognizing text based on fuzzy logic depending on data of the font. This procedure proposes a way for recognition of distorted letters using statistics and fuzzy logic. Their focus was recognition of text of Bible written in calligraphy. Another such work is done by Badawy et. al. on Automatic license plate recognition (ALPR) is related to text recognition which extracts the number from the number plate and the information about the vehicle. The information extracted can be used in many applications, such as toll n payment, parking fee payment, and freeway and arterial monitoring systems for traffic surveillance. The ALPR uses infrared camera to take images.

Recognition of text from document images as a process of an Optical Character Recognition (OCR) system is important concept.

There are different technologies for automatic identification and establishing position of OCR among these techniques. Shyam G.Dafe have presented different steps that are involved in the OCR system.

2.2.1 The Literary License

Literature has the ability to form its own language that may consist of a unique vocabulary. The main concerns, while translating any text are accuracy maintenance of relevance, and context of the idea (Gutt, 2014). For instance, the works of Shakespeare are thoroughly enjoyed while reading in other languages due to the parallel translations provided by the efficient translators.

The translations of his works into other languages have become comparatively comfortable, which has helped the translators a great deal in the understanding of the culture specific expressions and the poetic aspect of the language. Shakespeare's work is an example of intralingual translation that refers to the paraphrasing of source text. The approach has provided us with the time contained stories of plays, like Rome and Juliet, into narrations that are easily perceivable for the modern audience (Hoenselaars, 2014).

The researchers and translators of literature are amazed by the myriad and unique names that are specified to various characters in the literary works. However, the contemporary literature now fancies a literary license that consents the authors to use the fictional names and destinations in their works and present it to almost every culture across the world (Macaluso, 2015).

The approach has led us to the origin of various fictional characters like Harry Potter, Sir Lancelot, and fictional places including Stephen King's Castle Rock or Smallville (native town) of Superman. The careful crafting of a particular character in particular time and setting is the biggest challenge for a literary translator (Munday, 2016).

2.2.2 The Woes of Wordplay

Various verbal acrobatics are used to produce beautiful write ups, which are another great challenge for the translators to overcome to preserve the original beauty of the text. The writings of Shakespeare are thoroughly enjoyed by the literary minds as well as the common readers down the ages. The reason among many others, is the employment of the verbal acrobatics in his plays. He is regarded as the Master of Wordplay in the world of literature (Keller, 2017). However, those who read just for entertainment might not enjoy reading his writings as much as those, who read for the craving of art, knowledge and wisdom. The wordplay has been known as a literary device,

which can be shaped into many forms. For instance, wordplay can be used as a double entendre, simple pun.

2.2.3 Poetry and the Roving Rhymes

In the domain of translation of a literary text, poetry is regarded as a major problem, as it is difficult to maintain the essence of a poem after translating it into another language. It is indeed a difficult task to achieve the similar rhythmic pulse as of the original poetry (Macaluso, 2015). Rhymes have been associated with alliteration, which are known to defy and deceive the dexterous doyens of decoding. It would be of great benefit if a translator is able to work with alliterations. Therefore, this procedure has been known to bring the literature content close to the most devious devices among all the metaphors.

2.2.4 Analogous and Metaphorical Monstrosities

A metaphor is defined as a phrase that completely depends on the cultural familiarity of the audience (Herzfeld, 2014). For instance, the metaphor 'as stubborn as a donkey' could only be understood by the listeners if they are familiar with the nature of donkeys and the background of the statement. Therefore, metaphor has been regarded as a unique design that completely relies on the cultural familiarity with a particular culture to take full effect. For instance, a metaphor is merely an illogical comparison between oranges and bananas without any cultural reference.

2.3 Textual Hate Speech Detection

Authors: Fatimah Alkomah and Xiaogang Ma Department of Computer Science, University of Idaho, Moscow

The literature study in was devoted to building a generic metadata architecture for hate speech classification based on predefined score groups using semantic analysis and fuzzy logic analysis. The study in targeted hate speech concerning gender, religion, and race related to cyberterrorism and international legal frameworks; however, the study did not focus on Twitter or datasets for machine learning. Most of the papers cited in are related to the legal literature that defines hate speech for criminal sanctions. The study in was devoted to hate speech geographical aspects, social media platform diversity, and the generic qualitative or quantitative methods used by researchers. To the best of our knowledge, no review has been dedicated to English hate speech dataset analysis. This paper

aimed to deeply review hate speech concepts, methods, and datasets to provide researchers with an insight into the latest state-of-the-art studies in hate speech detection.

Information 2022,13, 2733 of 22This study carries a systematic literature review based on the methodology of Transfield et al. The method synthesizes and extracts results on evidence-based systematic literature extracted based on four aspects: the research question, review criteria, final literature review, and data extraction and synthesis.

CHAPTER 3
METHODOLOGY

3.1 EXISTING SYSTEM

There is a lot of development in the current system. In the older version there was no feature to display status, there was no feature to share documents and there was no feature to share location. In the current version, all of these features are available. In older version we couldn't share images through doc's format. In this system user is able to access WhatsApp in windows through WhatsApp web application, which can be connected through QR code. There is another feature called export chat where user can send or share or get the chat detail for data analysis through email, Facebook or some messenger application and there is a WhatsApp text translator which enables the user to translate the text into their desired languages to make conversation more comfortable and easier. Here added new improvement called WhatsApp Behavior Manager for safety conversation which enable the user to make a copy and report that the text is inappropriate.

Textual analysis is a broad term for various research methods used to describe, interpret and understand texts. All kinds of information can be gleaned from a text – from its literal meaning to the subtext, symbolism, assumptions, and values it reveals.

The methods used to conduct textual analysis depend on the field and the aims of the research. It often aims to connect the text to a broader social, political, cultural, or artistic context. Relatedly, it's good to be careful of confirmation bias when conducting these sorts of analyses, grounding your observations in clear and plausible ways.

Textual analysis in cultural and media studies

In the fields of cultural studies and media studies, textual analysis is a key component of research. Researchers in these fields take media and cultural objects – for example, music videos, social media content, billboard advertising – and treat them as texts to be analysed.

Usually working within a particular theoretical framework (for example, using postcolonial theory, media theory, or semiotics), researchers seek to connect elements of their texts with issues in contemporary politics and culture. They might analyse many different aspects of the text:

- Word choice
- Design elements
- Location of the text
- Target audience
- Relationship with other texts

Textual analysis in this context is usually creative and qualitative in its approach. Researchers seek to illuminate something about the underlying politics or social context of the cultural object they're investigating.

Textual analysis in the social sciences

In the social sciences, textual analysis is often applied to texts such as interview transcripts and surveys, as well as to various types of media. Social scientists use textual data to draw empirical conclusions about social relations.

Textual analysis in the social sciences sometimes takes a more quantitative approach, where the features of texts are measured numerically. For example, a researcher might investigate how often certain words are repeated in social media posts, or which colors appear most prominently in advertisements for products targeted at different demographics.

Some common methods of analyzing texts in the social sciences include content analysis, thematic analysis, and discourse analysis.

3.1.1 DISADVANTAGES OF EXISTING SYSTEM:

- The existing model does not translate the text message into other languages.
- This existing system will not be able to translate the information on the images.
- The existing model does not have the facility to send a report mail on the abusive content that is sent by the sender.

3.2 PROPOSED SYSTEM

Data pre-processing, the initial part of the project is to understand implementation and usage of various python-built modules. The above process helps us to understand why different modules are helpful rather than implementing those functions from scratch by the developer. These various modules provide better code representation and user understandability. The following libraries are used such as NumPy, SciPy pandas, csv, matplotlib, emoji, seaborn etc. Exploratory data analysis, first step in this to apply a sentiment analysis algorithm which provides positives negative and neutral part of the chat and is used to plot pie chart based on these parameters. To plot a line graph which shows author and message count of each date, to plot a line graph which shows author and message count of each author, ordered graph of date vs message count, media sent by authors and their count, Display the message which is do not have authors, plot graph.).

Businesses need to keep numerous sensitive documents on hand, including tax documents, entity documents, leases, human resources documents, financial documents, and more. These documents need to be stored confidentially yet in a way that they are easily accessible when needed.

The crucial document management challenges that businesses commonly face include:

- Business stakeholders need to search for various business documents routinely as required. Employees save documents at varied locations and in segregated department-wise repositories. Thus, it takes significant time to find these documents, making it impossible to leverage them during urgent needs.
- Employees typically do not follow a systematic and consistent protocol for labeling documents in organizations, making it tricky and time-consuming to search quickly.
- Not all business documents are scanned and stored in digital formats. Paper documents are difficult to search and at greater risk of unauthorized access, damage, and theft. These also require dedicated storage space. The storage space and infrastructural needs thus keep increasing as the business grows, escalating costs.
- In the absence of a centralized repository, documents cannot be accessed by all stakeholders conveniently. It consumes a lot of time to find and access the documents, update, and store the modified documents in the right place and format.
- Without real-time collaboration, multiple people working together face hassles like delayed

updates, document sharing, synchronous editing, version-clash, and overwriting.

- When documents are not available on a common platform, employees working remotely cannot access all the required business documents; this interrupts overall workflow and affects productivity.

Centralized Document Management

It is impossible to organize documents well and ensure error-free updates when multiple employees work on the same documents. A document management system enables centralized document management, digitization, and automation.

Software-based document management allows a unified view of documents and work activities on a single DMS platform. It brings all the organizational stakeholders in one place to enable seamless document control and workflow management.

Security

Data security is one of the crucial requirements in document management for businesses that need to store numerous confidential documents. It enables business managers to allow document access to specifically authorized people and easily restrict others from viewing/editing the docs. The managers also get a holistic record of document access and modification, including user details and time of access/modification. It thus ensures catching security breaches quickly and increases accountability.

Ease of Remote Working

Document management systems eliminate various challenges associated with remote working, including communication, coordination, document sharing, workflow monitoring, data security, and more. The software thus supports remote working, enhancing work-life balance for all the business stakeholders.

The proposed system is a web application that uses a WhatsApp webhook to dynamically check for vulgar messages, trigger a mail to a specified person, and translate the message to a user-specified language. This system can be used by businesses or individuals to ensure that the messages they receive on WhatsApp are appropriate and can be understood by all parties involved, regardless of language barriers.

The proposed system can be divided into four main components:

- WhatsApp Webhook Receiver: This component is responsible for receiving the webhook payload from WhatsApp and parsing the message data. It then sends the message data to the next component.
- Natural Language Processing (NLP) Engine: The NLP engine analyzes the message data received from the webhook receiver and checks for any vulgar language. If vulgar language is detected, the NLP engine triggers the next component to send a mail to a specified person with the details of the message.
- Email Service Provider (ESP): This component is responsible for sending an email to a specified person with the details of the message if vulgar language is detected. The ESP can also be customized to notify a group of moderators who can take appropriate action on the message.
- Translation API: This component is responsible for translating the message to a user-specified language. The API can be configured to automatically detect the source language of the message, reducing the need for users to specify the language manually.

The proposed system provides several benefits, such as ensuring that messages received on WhatsApp are appropriate and can be understood by all parties involved, regardless of language barriers. It can also be customized to suit specific business needs, such as notifying a group of moderators instead of triggering an email. Additionally, the proposed system can be integrated with other systems, such as CRMs or marketing automation tools, to provide a seamless workflow.

Overall, the proposed system provides a powerful tool for businesses and individuals to automate tasks, streamline communication, and improve their overall customer experience.

3.2.1 ADVANTAGES OF PROPOSED SYSTEM

- The proposed system is able to notify the abusive contents (i.e) report mail to the specified mail.
- The proposed system uses specified language library to translate the contents in the image provided by the user.
- Real-time updates: Webhooks provide real-time updates and notifications whenever an event occurs, eliminating the need for polling or manual updates.
- Efficient: Webhooks are efficient because they only transmit data when an event occurs, as opposed to constantly querying for new data.
- Automation: Webhooks allow for automation of tasks, freeing up time for manual tasks and increasing productivity.
- Customization: Webhooks can be customized to meet specific business needs, providing flexibility and control over data transmission.
- Scalability: Webhooks can easily scale to handle large volumes of data, making them suitable for businesses of all sizes.
- Cost-effective: Webhooks are often free or inexpensive to use, making them a cost-effective solution for businesses.
- Integration: Webhooks can be easily integrated with other systems, such as CRMs or marketing automation tools, providing businesses with a seamless workflow.

CHAPTER 4
SYSTEM ARCHITECTURE

4.1 SYSTEM REQUIREMENTS

4.1.1 Software Requirements

- Python
- IDE-Visual Studio Code
- Frontend-HTML, CSS
- Dataset-WhatsApp Chat History

4.1.2 Hardware Requirements

- Processor – Dual core
- Hard disk Capacity-250GB
- RAM -2GB

4.2 SOFTWARE DESCRIPTION

4.2.1 PYTHON

Python is a high-level, general-purpose programming language. Its design philosophy emphasizes code readability with the use of significant indentation. Python is dynamically-typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly procedural), object-oriented and functional programming. It is often described as a "batteries included" language due to its comprehensive standard library.

Guido van Rossum began working on Python in the late 1980s as a successor to the ABC programming language and first released it in 1991 as Python 0.9.0Python 2.0 was released in 2000 and introduced new features such as list comprehensions, cycle-detecting garbage collection, reference counting, and Unicode support. Python 3.0, released in 2008, was a major revision that is not completely backward-compatible with earlier versions. Python 2 was discontinued with version 2.7.18 in 2020Python consistently ranks as one of the most popular programming languages.

What is CGI?

CGI is actually an external application that is written by using any of the programming languages like **C** or **C++** and this is responsible for processing client requests and generating dynamic content.

In CGI application, when a client makes a request to access dynamic Web pages, the Web server performs the following operations:

- It first locates the requested web page *i.e* the required CGI application using URL.
- It then creates a new process to service the client's request.
- Invokes the CGI application within the process and passes the request information to the application.
- Collects the response from the CGI application.
- Destroys the process, prepares the HTTP response, and sends it to the client.

So, in **CGI** server has to create and destroy the process for every request. It's easy to understand that this approach is applicable for handling few clients but as the number of clients increases, the workload on the server increases and so the time is taken to process requests increases.

4.2.2 INSTALL PYTHON MODULES

As a popular open source development project, Python has an active supporting community of contributors and users that also make their software available for other Python developers to use underopen source license terms.

This allows Python users to share and collaborate effectively, benefiting from the solutions others have already created to common (and sometimes even rare!) problems, as well as potentially contributing their own solutions to the common pool.

This guide covers the installation part of the process. For a guide to creating and sharing your own Python projects, refer to the distribution guide.

Note For corporate and other institutional users, be aware that many organisations have their own policies around using and contributing to open source software. Please take such policies into account when making use of the distribution and installation tools provided with Python.

Key terms pip is the preferred installer program. Starting with Python 3.4, it is included by default with the Python binary installers.

A virtual environment is a semi-isolated Python environment that allows packages to be installed for use by a particular application, rather than being installed system wide. Venv is the standard tool for creating virtual environments, and has been part of Python since Python 3.3. Starting with Python 3.4, it defaults to installing pip into all created virtual environments. Virtualenv is a third party alternative (and predecessor) to venv. It allows virtual environments to be used on versions of Python prior to 3.4, which either don't provide venv at all, or aren't able to automatically install pip into created environments.

The Python Package Index is a public repository of open source licensed packages made available for use by other Python users.

The Python Packaging Authority is the group of developers and documentation authors responsible for the maintenance and evolution of the standard packaging tools and the associated metadata and file format standards. They maintain a variety of tools, documentation, and issue trackers on both GitHub and Bitbucket.

Distutils is the original build and distribution system first added to the Python standard library in 1998. While direct use of distutils is being phased out, it still laid the foundation for the current packaging and distribution infrastructure, and it not only remains part of the standard library, but its name lives on in other ways (such as the name of the mailing list used to coordinate Python packaging standards development).

4.2.3 Installing Python Distribution Packages

These are quick answers or links for some common tasks.
… install pip in versions of Python prior to Python 3.4?
Python only started bundling pip with Python 3.4. For earlier versions, pip needs to be "bootstrapped" as described in the Python Packaging User Guide.

Python Packaging User Guide: Requirements for Installing Packages
… install packages just for the current user?

Passing the --user option to python -m pip install will install a package just for the current user, rather than for all users of the system.

… install scientific Python packages?

A number of scientific Python packages have complex binary dependencies, and aren't currently easy to install using pip directly. At this point in time, it will often be easier for users to install these packages by other means rather than attempting to install them with pip.

4.2.4 Python Packaging User Guide: Installing Scientific Packages

Work with multiple versions of Python installed in parallel?

On Linux, macOS, and other POSIX systems, use the versioned Python commands in combination with the -m switch to run the appropriate copy of pip:

python2 -m pip install Some Package # default Python 2

python2.7 -m pip install Some Package # specifically Python 2.7

python3 -m pip install Some Package # default Python 3

python3.4 -m pip install Som ePackage # specifically Python 3.4

Appropriately versioned pip commands may also be available.

On Windows, use the py Python launcher in combination with the -m switch:

py -2 -m pip install SomePackage # default Python 2

py -2.7 -m pip install SomePackage # specifically Python 2.7

py -3 -m pip install SomePackage # default Python 3

py -3.4 -m pip install SomePackage # specifically Python 3.4

Common installation issues Installing into the system Python on Linux

On Linux systems, a Python installation will typically be included as part of the distribution. Installing into this Python installation requires root access to the system, and may interfere with the operation of the system package manager and other components of the system if a component is unexpectedly upgraded using pip.

On such systems, it is often better to use a virtual environment or a per-user installation when installing packages with pip.

Pip not installed

It is possible that pip does not get installed by default. One potential fix is:

python -m ensurepip --default-pip

There are also additional resources for installing pip.

Installing binary extensions

Python has typically relied heavily on source based distribution, with end users being expected to compile extension modules from source as part of the installation process.

With the introduction of support for the binary wheel format, and the ability to publish wheels for at least Windows and macOS through the Python Package Index, this problem is expected to diminish over time, as users are more regularly able to install pre-built extensions rather than needing to build them themselves.

Some of the solutions for installing scientific software that are not yet available as pre-built wheel files may also help with obtaining other binary extensions without needing to build them locally.

4.2.5 SMTP SERVER

SMTP servers are complicated, and if you're just dipping your toe into the how-email-gets-sent world, it's easy to feel overwhelmed. To help you navigate your email sending—specifically, an SMTP server to send email—we've put together a list of the most common SMTP server questions we receive, so you'll be an SMTP expert in no time. SMTP stands for Simple Mail Transfer Protocol, and it's an application used by mail servers to send, receive, and/or relay outgoing mail between email senders and receivers.

An SMTP email server will have an address (or addresses) that the mail client or application you use can set and generally looks like this: smtp.serveraddress.com. For example, the SMTP server Gmail uses is smtp.gmail.com, and Twilio SendGrid's is smtp.sendgrid.com. You can generally find your SMTP email server address in the account or settings section of your mail client.When you send an email, with SMTP host Gmail or AOL, the SMTP server processes your email, decides which server to send the message to, and relays the message to that server. The recipient's inbox service provider, such as Gmail or AOL, then downloads the message and places it in the recipient's inbox. Like most servers, the SMTP server processes data to send to another server, but it has the very specific purpose of processing data related to the sending, receiving, and relaying of email. An SMTP server is also not necessarily on a machine. It's an application constantly running in anticipation of sending new mail.

Why are SMTP servers important?

Without an SMTP server, your email wouldn't make it to its destination. Once you hit Send, your email transforms into a string of code sent to the SMTP server. The SMTP server processes that code and passes on the message. If the SMTP server wasn't there to process the message, it would be lost in translation.

Additionally, the SMTP server verifies that the outgoing email is from an active account, acting as the first safeguard in protecting your inbox from illegitimate email. It also will send the email back to the SMTP sender if it's undeliverable. This informs the sender that they have the wrong email address or that the receiving server has blocked the email.

If you're looking for more information on SMTP, check out our post, SMTP Service Crash Course.

Common SMTP server providers & settings

SMTP Provider	URL	SMTP Settings
AOL	aol.com	smtp.aol.com
AT&T	att.net	smtp.mail.att.net
Comcast	comcast.net	smtp.comcast.net
iCloud	icloud.com/mail	smtp.mail.me.com
Gmail	gmail.com	smtp.gmail.com
Outlook	outlook.com	smtp-mail.outlook.com
Yahoo!	mail.yahoo.com	smtp.mail.yahoo.com

4.2.6 Twilio Whatsapp Webhook

Twilio is a cloud-based communication platform that enables businesses to build SMS, voice, and messaging applications. With Twilio's WhatsApp API, businesses can send and receive WhatsApp messages using the same platform that powers their other communications channels. The API provides a range of features to help businesses engage with customers on WhatsApp, including the ability to send messages, receive messages, and automate messaging flows.

One key feature of the Twilio WhatsApp API is webhooks. Webhooks are a way for Twilio to notify your application when an event occurs, such as a new message being received or a message being sent successfully. By setting up webhooks, your application can receive real-time updates about the status of your WhatsApp messages, allowing you to take action based on those updates.

Setting up Twilio WhatsApp API Webhooks

To set up webhooks for the Twilio WhatsApp API, you'll need to create a webhook endpoint on your server that can receive incoming HTTP requests. When an event occurs, such as a new message being received, Twilio will send an HTTP POST request to your webhook endpoint with information about the event.

The webhook payload will include a variety of data, such as the message body, sender phone number, and recipient phone number. Your application can then use this information to take action, such as sending an automated response or updating a database.

There are a few steps involved in setting up webhooks for the Twilio WhatsApp API:

- Configure your WhatsApp sandbox: Before you can start using webhooks, you'll need to create a Twilio WhatsApp sandbox. This is a testing environment that allows you to experiment with the Twilio API without affecting your production account. Once you've created a sandbox, you'll need to configure the webhook URL in the Twilio console.
- Create a webhook endpoint: Next, you'll need to create a webhook endpoint on your server that can receive incoming HTTP requests from Twilio. You'll need to create a URL that Twilio can use to send webhook notifications.
- Set up webhook event handlers: Once you've created your webhook endpoint, you'll need to set up event handlers for the different types of webhook events that Twilio can send. For example, you might set up a handler to handle incoming messages, and another handler to handle message status updates.
- Test your webhook endpoint: Finally, you'll need to test your webhook endpoint to make sure it's receiving notifications correctly. You can use Twilio's API Explorer to send test messages to your sandbox and verify that your webhook endpoint is receiving notifications.

In addition to webhooks, the Twilio WhatsApp API offers a wide range of features to help businesses build powerful messaging applications. For example, businesses can use Twilio's message templates to create pre-approved message formats that can be used to send personalized messages to customers. Twilio also provides built-in support for two-factor authentication, enabling businesses to add an extra layer of security to their messaging workflows. Additionally, businesses can use Twilio's API to integrate with third-party services, such as CRM platforms and customer support tools, to create a seamless customer experience. With Twilio's WhatsApp API, businesses can unlock the power of WhatsApp messaging to engage with customers in new and innovative ways.

Another key advantage of the Twilio WhatsApp API is its scalability. Twilio's cloud-based platform is designed to handle high volumes of messaging traffic, making it ideal for businesses of all sizes. Whether you're a small startup or a large enterprise, Twilio can help you build a messaging application that can handle thousands or even millions of messages per day. Additionally, Twilio's platform provides built-in analytics and reporting tools, allowing businesses to monitor the performance of their messaging workflows and make data-driven decisions to optimize their messaging campaigns.

4.3 IMPLEMENTATION OF THE PROPOSED SYSTEM

Firstly, Python was downloaded. Code is implemented using Python. Users create an account in the google drive. Then the user can upload a file in the website. The user can view the analysis report generated by the user chat history. The modules of the project are as follows:

METHODOLY

Natural language processing strives to build machines that understand and respond to text or voice data—and respond with text or speech of their own—in much the same way humans do.
Natural language processing (NLP) refers to the branch of computer science—and more specifically, the branch of artificial intelligence or AI—concerned with giving computers the ability to understand text and spoken words in much the same way human beings can.

NLP combines computational linguistics—rule-based modelling of human language—with statistical, machine learning, and deep learning models. Together, these technologies enable computers to process human language in the form of text or voice data and to 'understand' its full meaning, complete with the speaker or writer's intent and sentiment.

NLP drives computer programs that translate text from one language to another, respond to spoken commands, and summarize large volumes of text rapidly—even in real time. There's a good chance you've interacted with NLP in the form of voice-operated GPS systems, digital assistants speech-to-text dictation software, customer service chatbots, and other consumer conveniences. But NLP also plays a growing role in enterprise solutions that help streamline business operations, increase employee productivity, and simplify mission-critical business processes.

NLP tasks

Human language is filled with ambiguities that make it incredibly difficult to write software that accurately determines the intended meaning of text or voice data. Homonyms, homophones, sarcasm, idioms, metaphors, grammar and usage exceptions, variations in sentence structure—these just a few of the irregularities of human language that take humans years to learn, but that programmers must teach natural language-driven applications to recognize and understand accurately from the start, if those applications are going to be useful.

NLP tools and approaches

Python and the Natural Language Toolkit (NLTK).The Python programming language provides a wide range of tools and libraries for attacking specific NLP tasks. Many of these are found in the Natural Language Toolkit, or NLTK, an open-source collection of libraries, programs, and education resources for building NLP programs.

The NLTK includes libraries for many of the NLP tasks listed above, plus libraries for subtasks, such as sentence parsing, word segmentation, stemming and lemmatization (methods of trimming words down to their roots), and tokenization (for breaking phrases, sentences, paragraphs and passages into tokens that help the computer better understand the text). It also includes libraries for implementing capabilities such as semantic reasoning, the ability to reach logical conclusions based on facts extracted from text.

Statistical NLP, machine learning, and deep learning

The earliest NLP applications were hand-coded, rules-based systems that could perform certain NLP tasks, but couldn't easily scale to accommodate a seemingly endless stream of exceptions or the increasing volumes of text and voice data.

Enter statistical NLP, which combines computer algorithms with machine learning and deep learning models to automatically extract, classify, and label elements of text and voice data and then assign a statistical likelihood to each possible meaning of those elements. Today, deep learning models and learning techniques based on convolutional neural networks (CNNs) and recurrent neural networks (RNNs) enable NLP systems that 'learn' as they work and extract ever more accurate meaning from huge volumes of raw, unstructured, and un labelled text and voice data sets.

NLP use cases

Natural language processing is the driving force behind machine intelligence in many modern real-world applications. Here are a few examples: Machine translation: Google Translate is an example of widely available NLP technology at work. Truly useful machine translation involves more than replacing words in one language with words of another. Effective translation has to capture accurately the meaning and tone of the input language and translate it to text with the same meaning and desired impact in the output language. Machine translation tools are making good progress in terms of accuracy. A great way to test any machine translation tool is to translate text to one language and then back to the original. An oft-cited classic example: Not long ago, translating “The spirit is

willing but the flesh is weak" from English to Russian and back yielded "The vodka is good but the meat is rotten."

Text summarization: Text summarization uses NLP techniques to digest huge volumes of digital text and create summaries and synopses for indexes, research databases, or busy readers who don't have time to read full text. The best text summarization applications use semantic reasoning and natural language generation (NLG) to add useful context and conclusions to summaries.

4.3.1 MODULE-1 TEXT ANALYSIS

The user is required to upload all the chats from a specific person in WhatsApp, which should be in a specific format named chat_example.txt. This format is necessary to ensure the proper functioning of the text analysis process. By adhering to this format, the user can ensure that the chat data is properly structured and can be effectively analyzed using the text analysis tool. Once the data is uploaded, the text analysis process can begin, providing insights into the content of the chat and enabling the user to draw meaningful conclusions based on the analysis. This process facilitates efficient and accurate analysis of chat data, allowing users to gain valuable insights from their WhatsApp conversations.

4.3.2 MODULE-2 VULGAR MESSAGE DETECTION

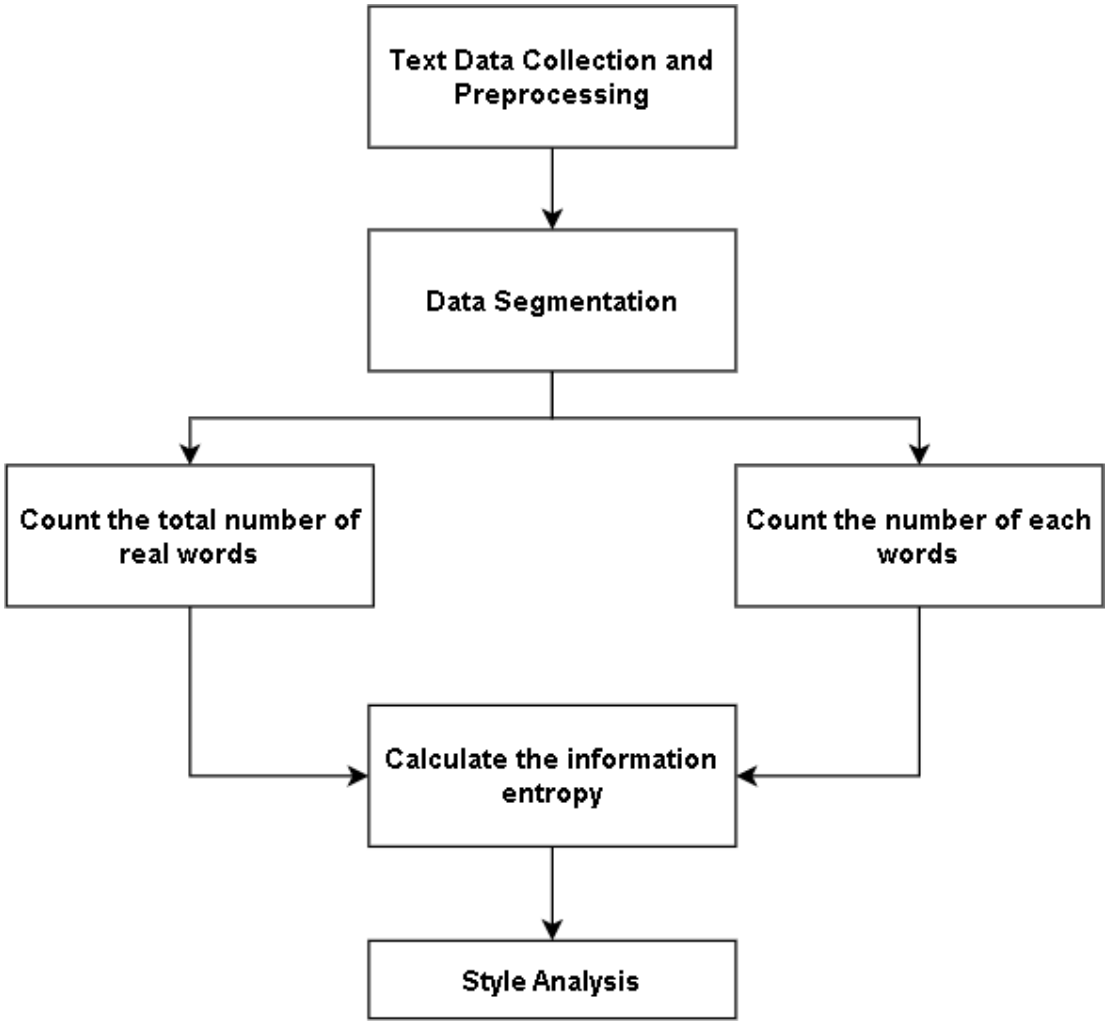
The scenario involves using the Twilio API to retrieve a message, which is then compared against a list of vulgar words contained in a CSV file. If a vulgar word is detected in the message, an email is sent to the specified user's email address using SMTP. However, if the user has not provided an email address, a predetermined email address is used as the recipient of the message. This process enables the filtering of inappropriate messages and alerts users to the presence of vulgar language in their incoming messages. The integration of the Twilio API and SMTP email services provides a seamless and automated approach to managing messaging workflows and maintaining a high standard of communication.

4.3.3 MODULE-3 TEXT TRANSLATION

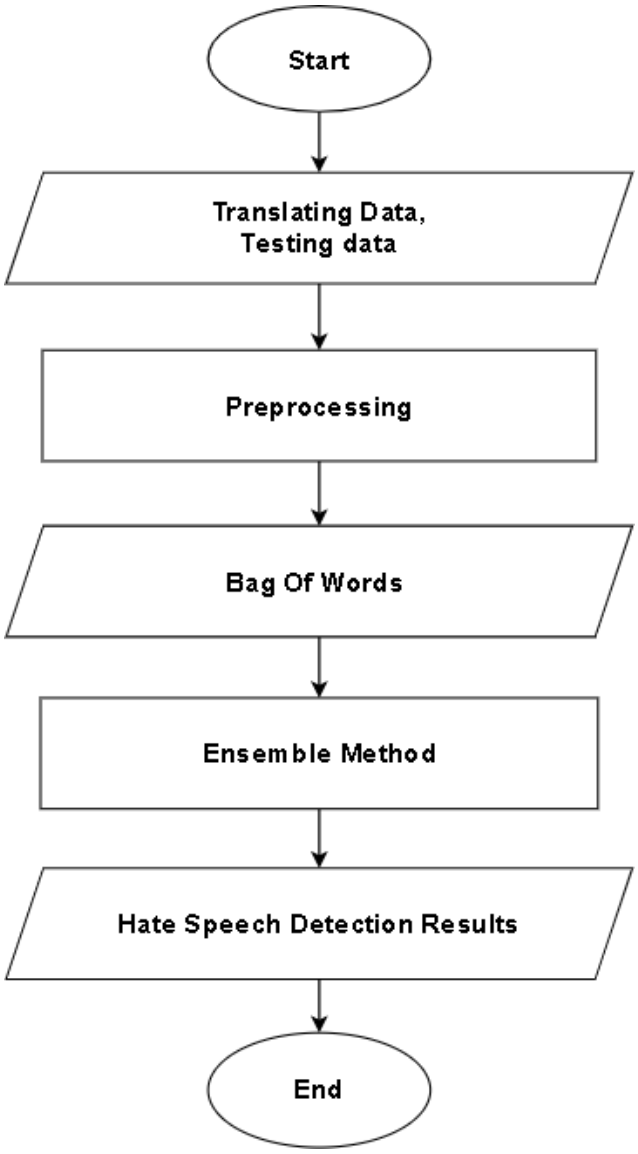
The user is required to upload an image to the platform. Once uploaded, the user selects the target language to translate the image to and submits the request. The platform then translates the image to the selected language and returns the translated version to the user.

4.4 FLOW DIAGRAMS

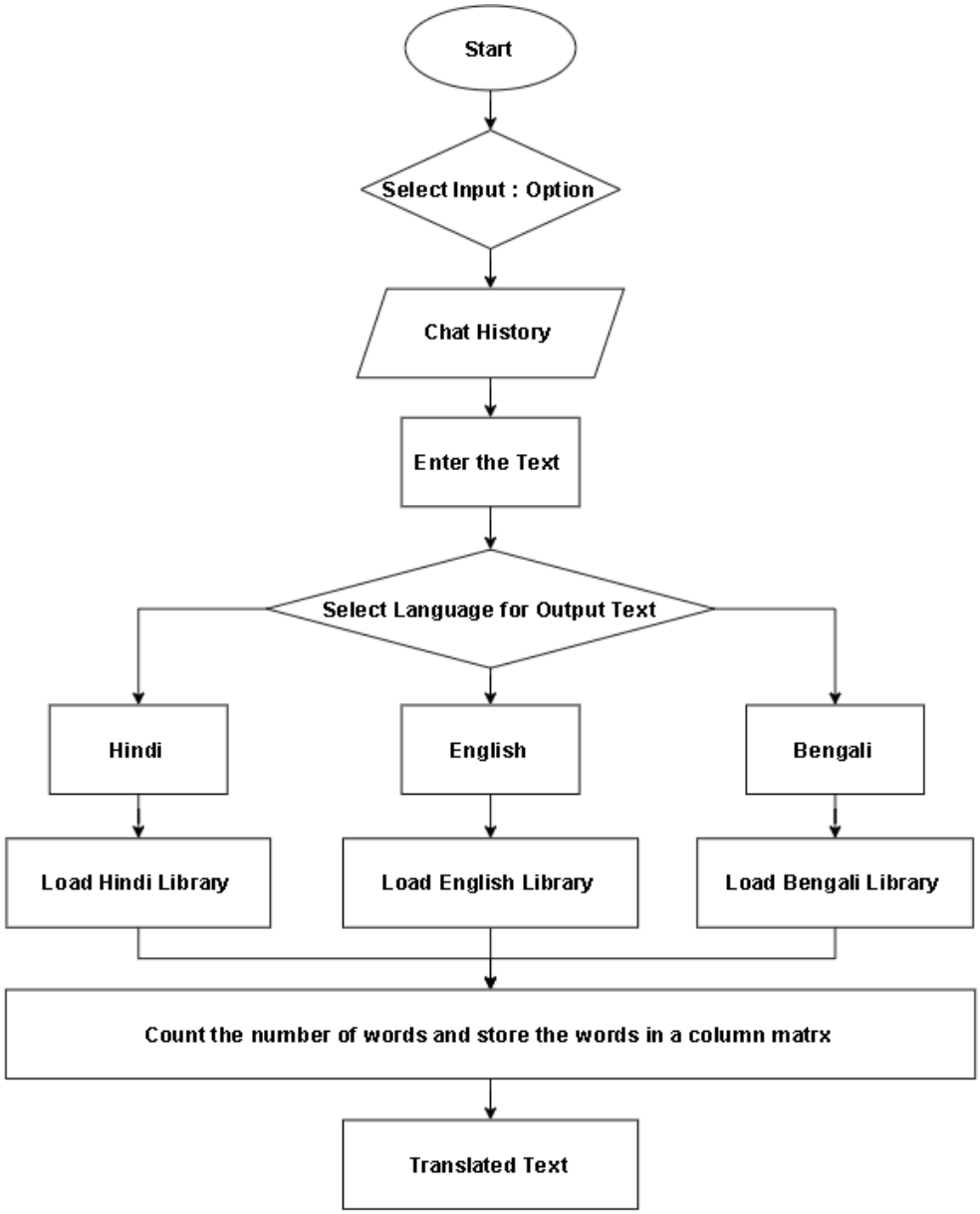
4.4.1 TEXT ANALYSIS



4.4.2 VULGAR MESSAGE DETECTION



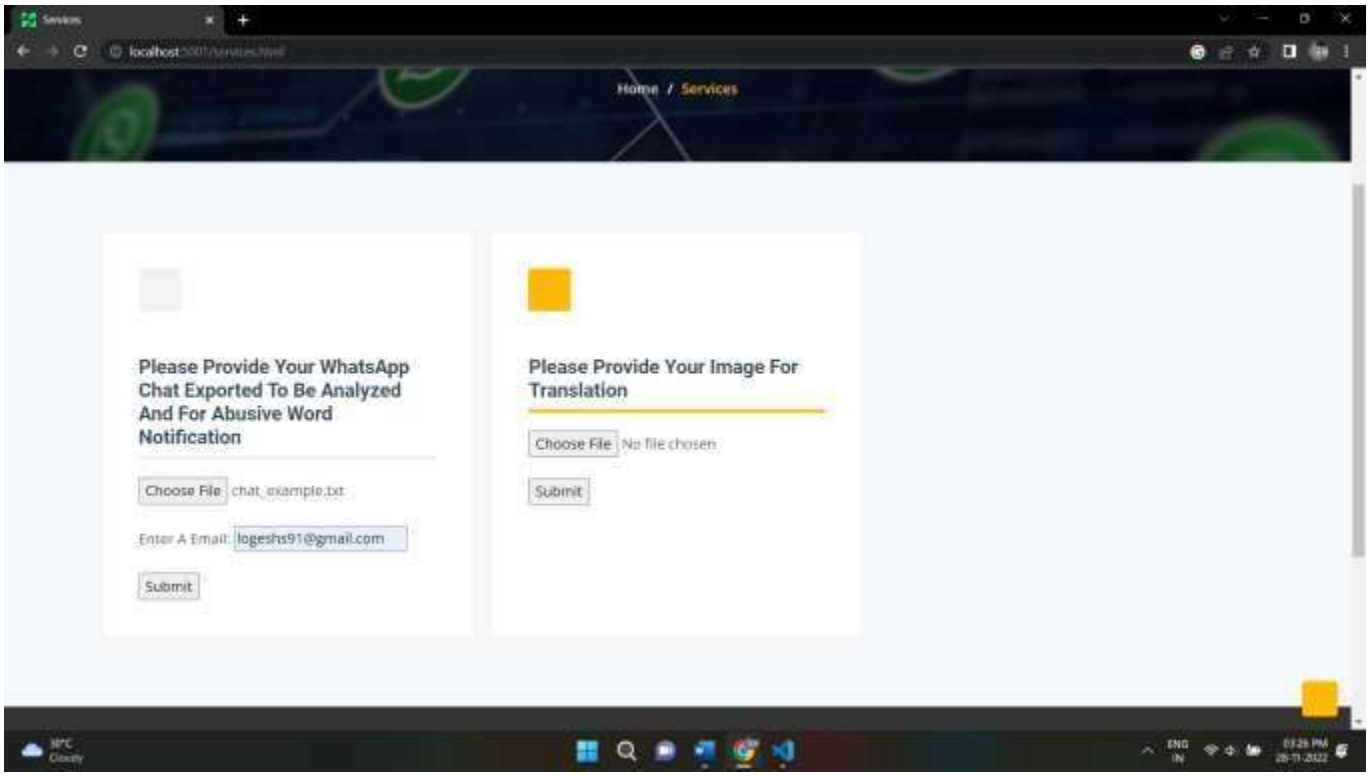
4.4.3 TEXT TRANSLATION



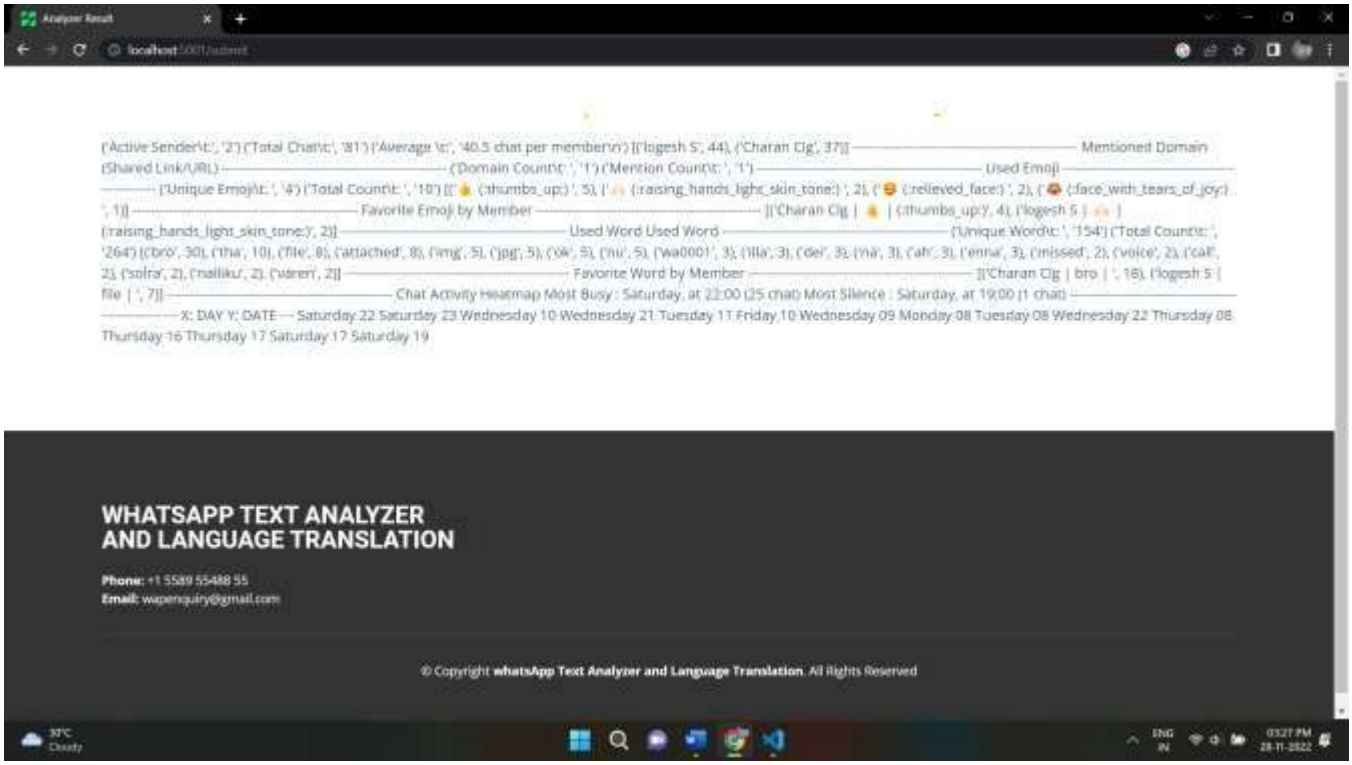
CHAPTER 5

RESULTS AND DISCUSSIONS

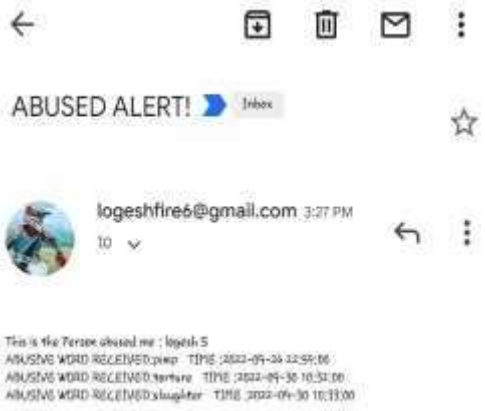
5.1 SERVICE PAGE:



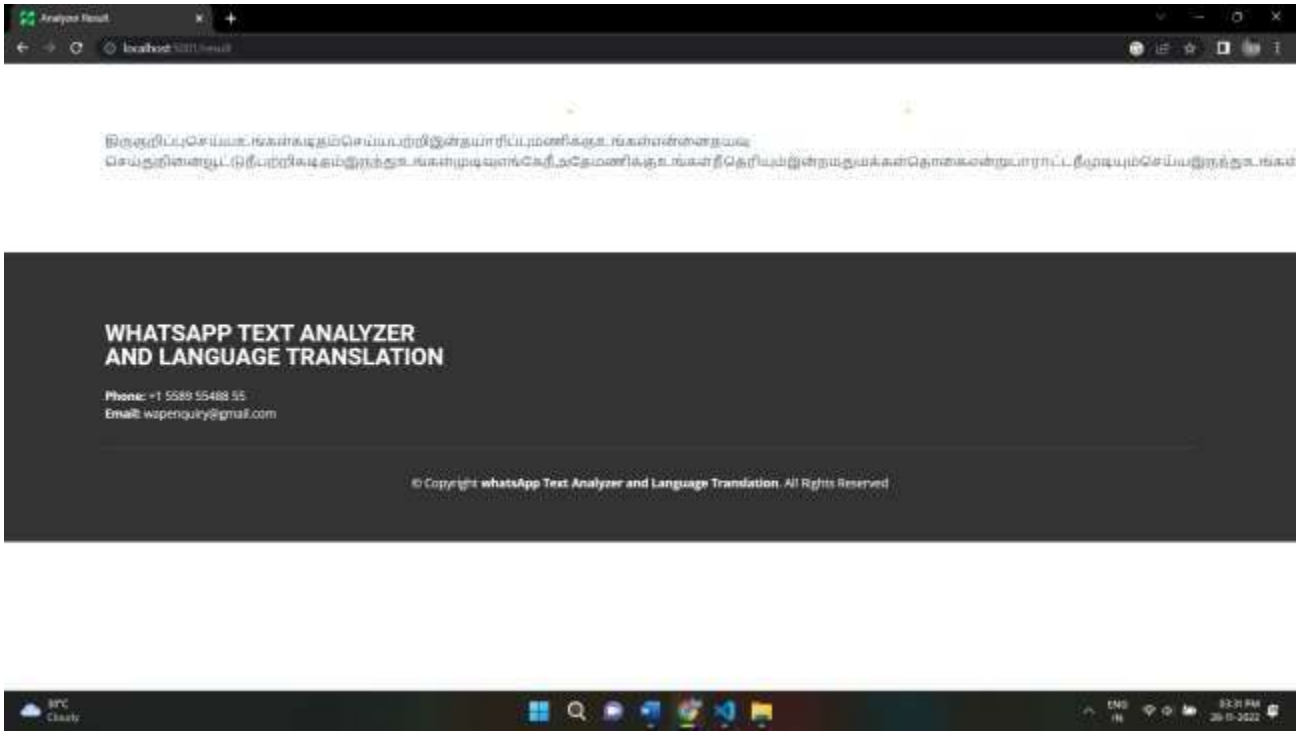
5.2 ANALYSIS REPORT:



5.3 GMAIL ALERT:



5.4 IMAGE TRANSLATION:



CHAPTER 6

CONCLUSION AND FUTURE WORKS

6.1 CONCLUSION

We addressed an interesting and long-lasting problem in WhatsApp. The proposed systems are resistant to make a higher security in WhatsApp. This System will result the total analysis of chat text and if there is a vulgar text it alerts user defined g-mail with a copy of that vulgar words. It allows user to make a translation from the image that user given.We can use a WhatsApp webhook to dynamically check if a message is vulgar or offensive. If the message is inappropriate, we can send a mail to a specified person and translate the message to a different language. This can be achieved by using NLP, ESP, and translation APIs, and integrating them into a web application that can receive the WhatsApp webhook. This approach can help maintain the quality of messages exchanged on the platform and provide a better user experience.

6.2 FUTURE WORKS

These Proposed uses Download chat history for analysis and translation, instead of making this system static we can make a dynamic system using WhatsApp webhook. Webhooks allows you to receive real-time HTTP notifications of changes to specific objects. For example, we could send you a notification when a user sends you a message or when a message template's status has changed. It’s is a website instead of that it can build on mobile app Which makes so reliable and easier for user.

CHAPTER 7

APPENDICES

7.1 SOURCE CODE

1.Main.py

```
#!/usr/bin/env  python
# -*- coding: utf-8 -*-
import argparse
from cgitb import text
import email
from gettext import translation
from importlib.resources import path
from string import octdigits
from tokenize import String
from turtle import Screen
import numpy
from email import header
from fileinput import close
from http import server
import io
import itertools
from operator import concat, index
import sys
import csv
from typing import Text, final
import emoji
from collections import Counter
from flask import Flask,render_template,request
app=Flask(_name_,template_folder='template')
@app.route("/")
def home():
return render_template("index.html")
@app.route("/submit",methods=['GET',  'POST'])
```

```
def submit():
if request.method=='POST':
analyze=[] #->RETURN OUTPUT
nameoffile=request.form['myFile']
list4=[]
lini=[]
# imported from current directory
from chatline import Chatline
from font_color import Color
"""
CLI Set
"""
parser = argparse.ArgumentParser(
description='Read and analyze whatsapp chat',
usage="python whatsapp_analyzer.py"
)

stop_words_options = [
"arabic","bulgarian","catalan","czech","danish","dutch","english","finnish","french","german","hebr
ew","hindi","hungarian","indonesian","italian","malaysian","norwegian","polish","portuguese","rom
anian","russian","slovak","spanish","swedish","turkish","ukrainian","vietnamese"]

parser.add_argument(
'-s',
'--stopword',
required=False,
choices=stop_words_options,
metavar='',
help="Stop Words: A stop word is a commonly used word (such as 'the', 'a', 'an', 'in').\
In order to get insightful most common word mentioned in the chat, we need to skip these type of
word.\
The Allowed values are: " + ", ".join(stop_words_options))
```

```
parser.add_argument(
'-c',
'--customstopword',
required=False,
metavar='',
help="Custom Stop Words. File path to stop word. File must a raw text. One word for every line"
)

args = parser.parse_args()

"""
READ FILE
"""
try:
with io.open(r"C:\Users\91701\OneDrive\Desktop\Project\WhatsApp-Analyzer-master\%s"%nameoffile, "r", encoding="utf-8") as file:
lines = file.readlines()

except IOError as e:
print("File \"" + args.file + "\" not found. Please recheck your file location")
sys.exit()

stop_words = []
if args.stopword:
try:
with io.open("stop-words/" + args.stopword + ".txt", "r", encoding="utf-8") as file:
stop_words = [x.strip() for x in file.readlines()]
except IOError as e:
print("Stop Words file not found in \"" + args.file + "\" not found.")
sys.exit()
```

```
if args.customstopword:
try:
with io.open(args.customstopword, "r", encoding="utf-8") as file:
stop_words = [x.strip() for x in file.readlines()]
except IOError as e:
print("Stop Words file not found in \"" + args.file + "\" not found.")
sys.exit()

"""
PARSING AND COUNTING
"""
chat_counter = {
'chat_count': 0,
'deleted_chat_count': 0,
'event_count': 0,
'senders': [],
'timestamps': [],
'words': [],
'domains': [],
'emojis': [],
'fav_emoji': [],
'fav_word': []
}


previous_line = None
for line in lines:
chatline = Chatline(line=line, previous_line=previous_line)
previous_line = chatline
lini.append(line)
# Counter
```

```
if chatline.line_type == 'Chat':
chat_counter['chat_count'] += 1

if chatline.line_type == 'Event':
chat_counter['event_count'] += 1

if chatline.is_deleted_chat:
chat_counter['deleted_chat_count'] += 1

if chatline.sender is not None:
chat_counter['senders'].append(chatline.sender)
for i in chatline.emojis:
chat_counter['fav_emoji'].append((chatline.sender, i))

for i in chatline.words:
chat_counter['fav_word'].append((chatline.sender, i))

if chatline.timestamp:
chat_counter['timestamps'].append(chatline.timestamp)

if len(chatline.words) > 0:
chat_counter['words'].extend(chatline.words)

if len(chatline.emojis) > 0:
chat_counter['emojis'].extend(chatline.emojis)

if len(chatline.domains) > 0:
chat_counter['domains'].extend(chatline.domains)

for i in chat_counter["timestamps"]:
list4.append(i)
```

```
"""
REDUCE AND ORDER DATA
"""

def reduce_and_sort(data):
return sorted(
dict(
zip(
Counter(data).keys(),
Counter(data).values()
)
).items(),
key=lambda x: x[1],
reverse=True
)

def reduce_and_filter_words(list_of_words):
val = [w.lower() for w in list_of_words if (len(w) > 1) and (w.isalnum()) and (not w.isnumeric()) and
(w.lower() not in stop_words)]
return val

def filter_single_word(w):
return (len(w) > 1) and (w.isalnum()) and (not w.isnumeric()) and (w.lower() not in stop_words)

def reduce_fav_item(data):
exist = []
arr = []
for i in data:
if i[1] > 0 and not i[0][0] in exist:
exist.append(i[0][0])
arr.append(i)
```

```
return arr

chat_counter['senders'] = reduce_and_sort(chat_counter['senders'])
chat_counter['words'] = reduce_and_sort(reduce_and_filter_words(chat_counter['words']))
chat_counter['domains'] = reduce_and_sort(chat_counter['domains'])
chat_counter['emojis'] = reduce_and_sort(chat_counter['emojis'])
chat_counter['timestamps'] = reduce_and_sort([(x.strftime('%A'), x.strftime('%H')) for x in chat_counter['timestamps']])
chat_counter['fav_emoji'] = reduce_fav_item(reduce_and_sort(chat_counter['fav_emoji']))
chat_counter['fav_word'] = reduce_fav_item(reduce_and_sort([x for x in chat_counter['fav_word'] if filter_single_word(x[1])]))
"""
VISUALIZE
"""
def printBar (value, total, label = '', prefix = '', decimals = 1, length = 100, fill = '', printEnd = "\r"):
filledLength = int(value / (total / length))
bar = fill * filledLength + ' * (length - filledLength)
print("\r{} |{} {}".format(label, bar, Color.bold(str(value))), end = printEnd)
print()

def printBarChart(data, fill=''):
if len(data) <= 0:
print("Empty data")
return

total = max([x[1] for x in data])
max_label_length = len(sorted(data, key=lambda tup: len(tup[0]), reverse=True)[0][0])
for i in data:
label = i[0] + ' ' * (max_label_length - len(i[0]))
printBar(i[1], total, length=50, fill=fill, label=label)

def printCalendar(data):
```

```
days = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
hours = ['0' + str(x) if len(str(x)) < 2 else str(x) for x in range(24)]
max_val = float(data[max(data, key=data.get)]) if len(data) else 0

ticks = [
0,
0.25 * max_val,
0.50 * max_val,
0.75 * max_val,
]

sys.stdout.write("     ")
for day in days:
sys.stdout.write('\t[' + day[:3] + "]")

sys.stdout.write('\n')

for hour in hours:
sys.stdout.write("[" + hour + ':00]')

for day in days:

dict_key = (day, hour)

if dict_key in data:
# tick = str(ct[dict_key])

if data[dict_key] > ticks[3]:
tick = Color.custom("████", bold=True, fg_red=True)
elif data[dict_key] > ticks[2]:
tick = Color.custom("▓▓▓▓", bold=True, fg_orange=True)
elif data[dict_key] > ticks[1]:
```

```
tick = Color.custom("▒▒▒▒", bold=True, fg_green=True)
else:
tick = Color.custom("▒▒▒▒", bold=True, fg_light_grey=True)
else:
tick = Color.custom('===', bold=False, fg_white=True)

sys.stdout.write('\t ' + tick)
sys.stdout.write('\n')

# Senders
data = chat_counter['senders']
analyze.append(Color.red("-" * 50))
analyze.append(Color.red("Chat Count by Sender"))
analyze.append(Color.red("-" * 50))
a4=("Active Sender\t:", Color.red("{}".format(len(data))))
a5=("Total Chat\t:", Color.red("{}".format(sum([x[1] for x in data]))))
a6=("Average \t:", Color.red("{:.1f} chat per member".format((sum([x[1] for x in data]) / len(data))
if len(data) else 0))+"\n")
print()
a7=(data[:20])
if len(data) > 20:
analyze.append("---")
analyze.append("Other from {} member | {}".format(Color.red(str(len(data[20:]))),
Color.red(str(sum([x[1] for x in data[20:]])))))
print()
print()

# Domains
data = chat_counter['domains']
a8=(Color.blue("-" * 50))
a9=(Color.blue("Mentioned Domain (Shared Link/URL)"))
a10=(Color.blue("-" * 50))
```

```
a11=("Domain Count\t: ", Color.blue(str(len(data))))
a12=("Mention Count\t: ", Color.blue(str(sum([x[1] for x in data]))))
print()
a13=(data[:20])
if len(data) > 20:
analyze.append("---")
analyze.append("Other {} domain | {}".format(Color.blue(str(len(data[20:]))),
Color.blue(str(sum([x[1] for x in data[20:]])))))
print()
print()


# Emojis
data = [(x[0] + " (" + emoji.demojize(x[0]) + ") ", x[1]) for x in chat_counter['emojis']]
a13=(Color.orange("-" * 50))
a14=(Color.orange("Used Emoji"))
a15=(Color.orange("-" * 50))
a16=("Unique Emoji\t: ", Color.orange(str(len(data))))
a17=("Total Count\t: ", Color.orange(str(sum([x[1] for x in data]))))
print()
a18=(data[:20])
if len(data) > 20:
analyze.append("---")
analyze.append("Other {} emoji | {}".format(Color.orange(str(len(data[20:]))),
Color.orange(str(sum([x[1] for x in data[20:]])))))
print()
print()

# Fav Emojis
data = [(x[0][0] + " | " + x[0][1] + " | (" + emoji.demojize(x[0][1]) + ")", x[1]) for x in
chat_counter['fav_emoji']]
a19=(Color.orange("-" * 50))
```

```
a20=(Color.orange("Favorite Emoji by Member"))
a21=(Color.orange("-" * 50))
print()
a22=(data[:20])
print()
print()

# Words
data = chat_counter['words']
a23=(Color.green("-" * 50))
a24=(Color.green("Used Word"))
a25=(Color.green("-" * 50))
a26=("Unique Word\t: ", Color.green(str(len(data))))
a27=("Total Count\t: ", Color.green(str(sum([x[1] for x in data]))))
print()
a28=(data[:20])
if len(data) > 20:
analyze.append("---")
analyze.append("Other {} word | {}".format(Color.green(str(len(data[20:]))),
Color.green(str(sum([x[1] for x in data[20:]])))))
print()
print()

# Fav Word
data = [(x[0][0] + " | " + x[0][1] + " | ", x[1]) for x in chat_counter['fav_word']]
a29=(Color.green("-" * 50))
a30=(Color.green("Favorite Word by Member"))
a31=(Color.green("-" * 50))
print()
a32=(data[:20])
print()
```

```
# Heatmap
data = chat_counter['timestamps']
a33=(Color.purple("-" * 50))
a34=(Color.purple("Chat Activity Heatmap"))
a35=(Color.purple("-" * 50))
a41=""
a42=""
if len(data) > 0:
a41=("Most Busy\t: {}, at {} ({} chat)".format(
Color.purple(str(data[0][0][0])),
Color.purple(str(data[0][0][1]) + ":00"),
Color.purple(str(data[0][1]))))
a42=("Most Silence\t: {}, at {} ({} chat)".format(
Color.purple(str(data[-1][0][0])),
Color.purple(str(data[-1][0][1]) + ":00"),
Color.purple(str(data[-1][1]))))
print()
a37=('---')
a38=('X: DAY')
a39=('Y: DATE')
a40=('---')
# a43=('Less [{}{}{}{}{}] More'.format(
#     Color.custom("===", bold=False),
#     Color.custom("▒▒▒▒", bold=True, fg_light_grey=True),
#     Color.custom("▒▒▒▒", bold=True, fg_green=True),
#     Color.custom("▒▒▒▒", bold=True, fg_orange=True),
#     Color.custom("▒▒▒▒", bold=True, fg_red=True)
# ))
print()
finalresult=""
analyze=[a4,a5,a6,a7,a8,a9,a10,a11,a12,a13,a14,a15,a16,a17,a18,a19,a20,a21,a22,a23,a24
,a24,a25,a26,a27,a28,a29,a30,a31,a32,a33,a34,a41,a42,a35,a37,a38,a39,a40]
```

```
for i in analyze:
finalresult=finalresult+str(i)+"\n"
for i in  dict(data):
finalresult=finalresult+(i[0]+" "+i[1])+"\n"
#BADWORD
Name=chat_counter['senders'][0]
list1=[]
list2=[]
msg=""
TEXT=""
c=0
for x in chat_counter['words']:
list1.append(x[0])
file=open('bad-words.csv')
type(file)
csvreader=csv.reader(file)
header=[]
header=next(csvreader)
for row in csvreader:
list2.append(row[0])
list5=[]
list6=[]
list3=[]
for i in chat_counter['words']:
for j in list2:
if i[0]==j:
list5.append(j)
for k in list5:
h=1
for o in lini:
char=o.split()
for r in char:
```

```
if k==r:
list6.append(h)
h=h+1
c=1
for i in list4:
for j in list6:
if c==j:
list3.append(i)
c=c+1

for (i,j) in itertools.zip_longest(list5,list3):
TEXT=TEXT+"ABUSIVE WORD RECEIVED:"+i+"  "+" TIME :"+str(j)+"\n"

TEXT="This is the Person abused me : "+Name[0]+"\n"+TEXT
#Mailer
qemail=request.form['email']
import smtplib
server=smtplib.SMTP_SSL("smtp.gmail.com",465)
server.login("logeshfire6@gmail.com","jatxquxctpxcqklg")
SUBJECT="ABUSED ALERT!"
mg='Subject:{}\n\n{}'.format(SUBJECT, TEXT)
server.sendmail("logeshfire6@gmail.com","{}".format(qemail),mg)
server.quit()

return render_template("analysis.html",name=finalresult)
else:
return "Exception"

@app.route("/result",methods=['GET', 'POST'])
def result():
if request.method=='POST':
imgname=request.form['image']
```

```
#Translation
import re
from PIL import Image
from pytesseract import pytesseract
path_to_tesseract = r'C:\Program Files\Tesseract-OCR\tesseract.exe'
path_to_image =(r'C:\Users\91701\OneDrive\Desktop\Project\WhatsApp-Analyzer-master\source\%s'%(imgname))
pytesseract.tesseract_cmd = path_to_tesseract
img = Image.open(path_to_image)
text = pytesseract.image_to_string(img)
text=text.replace("!","")
text=text.replace(".","")
wordz=re.split("\n",text)
dummy=""
for i in wordz:
dummy=dummy+i+" "

englishp=[]
single=[]
file=open('4000-most-common-english-words-csv.csv')
type(file)
csvreader=csv.reader(file)
header=[]
header=next(csvreader)
for row in csvreader:
englishp.append(row[0])
wordz1=dummy.split(" ")
for i in wordz1:
if i!=" ":
for j in englishp:
if i==j:
single.append(i)
```

```
output=[]
language=[]
#englishtoTamil
from googletrans import Translator,constants
#source
translator=Translator()
iot=""
# text5=""
# text5=single[0]
# detection = translator.detect(text5)
# dect=detection.lang
#languages
language.append(constants.LANGUAGES)
#translation
translations=translator.translate(single,dest="ta")
for translation in translations: # type: ignore
output.append(f"{translation.text}")
for i in output:
iot=iot+i
return render_template("analysis.html",name=iot)
else:
return "exception"

@app.route("/contact.html")
def contact():
return render_template("contact.html")
@app.route("/about.html")
def about():
return render_template("about.html")
@app.route("/services.html")
def service():
return render_template("services.html")
```

```
if __name__=='__main__':
app.run(debug=True,port=5001)
```

Chatline.py

```
# -*- coding: utf-8 -*-
import re
from dateutil import parser
import emoji
import patterns

# TODO: Classify attachment

class Chatline:

    def __init__(self, line="", previous_line=None, debug=False):
        self.previous_line = previous_line
        self.line = line
        self.line_type = None # Chat/Event/Attachment
        self.timestamp = None
        self.sender =  None
        self.body = ""
        self.is_startingline = False
        self.is_followingline = False
        self.is_deleted_chat = False
        self.words = []
        self.emojis = []
        self.domains = []

        self.parse_line(line)
        if debug:
            print()
```

```
        for i in self.__dict__:
            print(i, ':',  self.__dict__[i])
        print("----- ")

def replace_bad_character(self, line=""):
    line = line.strip()
    for x in patterns.BAD_CHARS:
        line = line.replace(x, "")

    return line

def is_starting_line(self, line=""):
    """
    Starting line mean a line that started with date time.
    Because there are multiline chat. I called it following line.
    A starting line must be classified before it's data being extracted.

    The Rule is:
    <datetime><separator><contact/phone number>
    """
    match = re.match(re.compile(patterns.IS_STARTING_LINE, re.VERBOSE), line)
    if match:
        return match

    return None

def is_chat(self, body=""):
    """
    "Is Chat" means the body of a line is not an event.
    May contains attachment

    The Rule is:
```

```
    <contact/phone number><separator><message body>
    """
    match = re.match(re.compile(patterns.IS_CHAT, re.VERBOSE), body)
    if match:
        return match

    return None

def is_deleted(self, body=""):
    """
    Deleted message
    """
    for p in patterns.IS_DELETED_CHAT:
        match = re.match(p, body)
        if match:
            return body
    return None

def contains_attachment(self, body=""):
    """
    Classify attachment
    Note: in Android, there is no difference pattern wether it's an image,
        video, audio, gif, document or sticker.
    """
    for p in patterns.IS_ATTACHMENT:
        if re.match(p, body):
            return body
    return None

def extract_timestamp(self, time_string=""):
    """
    EXTRACT TIMESTAMP
```

```
    """
    timestamp = parser.parse(time_string)
    return timestamp

def extract_url(self, body=""):
    """
    Check if chat contais a url
    """
    return re.findall(patterns.IS_URL, body)

def get_domain(self, url=""):
    domain = url[0].replace("http://", '')
    domain = domain.replace("https://", '')
    domain = domain.split("/")
    return domain[0]

def get_words(self, string=""):
    #remove non alpha content
    regex = re.sub(r"[^a-z\s]+", "", string.lower())
    regex = re.sub(r'[^\x00-\x7f]',r'', regex)
    words = re.sub(r"[^\w]", " ", string).split()

    return words

def extract_emojis(self, string=""):
    return [c["emoji"] for c in emoji.emoji_list(string)]

def is_event(self, body=""):
    """Detect wether the body of chat is event log.
    If the body if an event, it won't be count and the body of the message will not analized

    Event log means note of event.
```

```
    Below are known event log patterns in difference language
    - Group created
    - User joining group
    - User left group
    - Adding group member
    - Removing group member
    - Security code changed
    - Phone number changed
    -

    Feel free to add similar pattern for other known pattern or language

    Keyword arguments:
    body -- body of exported chat

    The Rule is:
    Match the known event message
    """
    for p in patterns.IS_EVENT:
        match = re.match(p, body)
        if match:
            return match
    return None

def parse_line(self, line=""):
    line = self.replace_bad_character(line)
    # Check wether the line is starting line or following line
    starting_line = self.is_starting_line(line)

    if starting_line:
        # Set startingline
        self.is_startingline = True
```

```
        # Extract timestamp
        dt = self.extract_timestamp(starting_line.group(2))
        # Set timestamp
        if dt:
            self.timestamp = dt

        # Body of the chat separated from timestamp
        body = starting_line.group(18)
        self.parse_body(body)

    else:
        # The line is following line
        # Set following
        self.is_followingline = True

        # Check if previous line has sender
        if self.previous_line and self.previous_line.sender:
            # Set current line sender, timestamp same to previous line
            self.sender = self.previous_line.sender
            self.timestamp = self.previous_line.timestamp
            self.line_type = "Chat"

        body = line
        self.body = line
        self.parse_body(body, following=True)

def parse_body(self, body="", following=False):
    # Check wether the starting line is a chat or an event
    chat = self.is_chat(body)

    if chat or following:
```

```
# Set line type, sender and body
self.line_type = "Chat"
message_body = body
if not following:
    self.sender = chat.group(1)
    message_body = chat.group(3)

self.body = message_body

has_attachment = self.contains_attachment(message_body)
if has_attachment:
    # Set chat type to attachment
    self.line_type = "Attachment"
else:
    if self.is_deleted(message_body):
        # Set deleted
        self.is_deleted_chat = True
    else:
        words = message_body

        #URL & Domain
        urls = self.extract_url(message_body)
        if urls:
            for i in urls:
                # Exclude url from words
                words = words.replace(i[0], "")

                # Set domains
                self.domains.append(self.get_domain(i))

        # Set Words
        self.words = self.get_words(words)
```

```
                #Emoji
                emjs = self.extract_emojis(message_body)
                if emjs:
                    self.emojis = emjs

        elif self.is_event(body):
            # Set line_type
            self.line_type = "Event"
```

Fontcolor.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from sys import platform

class Color(object):
    """
    reference from https://gist.github.com/Jossef/0ee20314577925b4027f and modified bit.

    """

    def __init__(self, text, **user_styles):

        styles = {
            # styles
            'reset': '\033[0m',
            'bold': '\033[01m',
            'disabled': '\033[02m',
            'underline': '\033[04m',
            'reverse': '\033[07m',
            'strike_through': '\033[09m',
            'invisible': '\033[08m',
```

```
    # text colors
    'fg_black': '\033[30m',
    'fg_red': '\033[31m',
    'fg_green': '\033[32m',
    'fg_orange': '\033[33m',
    'fg_blue': '\033[34m',
    'fg_purple': '\033[35m',
    'fg_cyan': '\033[36m',
    'fg_light_grey': '\033[37m',
    'fg_dark_grey': '\033[90m',
    'fg_light_red': '\033[91m',
    'fg_light_green': '\033[92m',
    'fg_yellow': '\033[93m',
    'fg_light_blue': '\033[94m',
    'fg_pink': '\033[95m',
    'fg_light_cyan': '\033[96m',
    'fg_white': '\033[97m',
    'fg_default': '\033[99m',
    # background colors
    'bg_black': '\033[40m',
    'bg_red': '\033[41m',
    'bg_green': '\033[42m',
    'bg_orange': '\033[43m',
    'bg_blue': '\033[44m',
    'bg_purple': '\033[45m',
    'bg_cyan': '\033[46m',
    'bg_light_grey': '\033[47m'
}

self.color_text = ''
for style in user_styles:
    try:
```

```
            self.color_text += styles[style]
        except KeyError:
            raise KeyError('def color: parameter `{}` does not exist'.format(style))

    self.color_text += text
    if platform == 'win32':
        self.color_text = text

def __format__(self):
    if platform == 'win32':
        return "{}".format(self.color_text)
    return '\033[0m{}\033[0m'.format(self.color_text)

@classmethod
def bold(clazz, text):
    cls = clazz(text, bold=True)
    return cls._format_()

@classmethod
def red(clazz, text):
    cls = clazz(text, bold=True, fg_red=True)
    return cls._format_()

@classmethod
def orange(clazz, text):
    cls = clazz(text, bold=True, fg_orange=True)
    return cls._format_()

@classmethod
def blue(clazz, text):
    cls = clazz(text, bold=True, fg_light_blue=True)
    return cls._format_()
```

```
@classmethod
def green(clazz, text):
    cls = clazz(text, bold=True, fg_green=True)
    return cls._format_()

@classmethod
def purple(clazz, text):
    cls = clazz(text, bold=True, fg_purple=True)
    return cls._format_()

@classmethod
def custom(clazz, text, **custom_styles):
    cls = clazz(text, **custom_styles)
    return cls._format_()
```

REFERENCES

[1] P. Sharma and S. Sharma, "An analysis of vision based techniques for quality assessment and enhancement of camera captured document images," 2016 6th International Conference - Cloud System and Big Data Engineering (Confluence), Noida, 2016, pp. 425-428, doi: 10.1109/CONFLUENCE.2016.7508157.

[2] S. Thakare, A. Kamble, V. Thengne and U. R. Kamble, "Document Segmentation and Language Translation Using Tesseract-OCR," 2018 IEEE 13th International Conference on Industrial and Information Systems (ICIIS), Rupnagar, India, 2018, pp. 148-151, doi: 10.1109/ICIINFS.2018.8721372.

[3] E. Gur and Z. Zelavsky, "Retrieval of Rashi Semi-cursive Handwriting via Fuzzy Logic," 2012 International Conference on Frontiers in Handwriting Recognition, Bari, 2012, pp. 354-359, doi: 10.1109/ICFHR.2012.262.

[4] S. Du, M. Ibrahim, M. Shehata and W. Badawy, "Automatic License Plate Recognition (ALPR): A State-of-the-Art Review," in IEEE Transactions on Circuits and Systems for Video Technology, vol. 23, no. 2, pp. 311-325, Feb. 2013, doi: 10.1109/TCSVT.2012.2203741.

[5] S. Malakar, S. Halder, R. Sarkar, N. Das, S. Basu and M. Nasipuri, "Text line extraction from handwritten document pages using spiral run length smearing algorithm," 2012 International Conference on Communications, Devices and Intelligent Systems (CODIS), Kolkata, 2012, pp. 616- 619, doi: 10.1109/CODIS.2012.6422278.

[6] Shyam G.Dafe, Shubham S. Chavhan , “Optical Character Recognition Using Image Processing”, International Research Journal of Engineering and Technology (IRJET)e-ISSN: 2395-0056 Volume: 05 Issue: 03 | Mar-2018.

[7] Available from: http://www. statista.com/statistics/260819/numberof-monthly-active-WhatsApp-users. Number of monthly active WhatsApp users worldwide from April 2013 to February 2016(in millions).

[8] Ahmed, I., Fiaz, T., “Mobile phone to youngsters: Necessity or addiction”, African Journal of Business Management Vol.5 (32), pp. 12512-12519, Aijaz, K. (2011).

[9] Aharony, N., T., G., The Importance of the WhatsApp Family Group: An Exploratory Analysis. “Aslib Journal of Information Management, Vol. 68, Issue 2, pp.1-37” (2016).

[10] Access Data Corporation. FTK Imager, 2013. Available at http://www.accessdata.com/support/product-downloads.

[11] D.Radha, R. Jayaparvathy, D. Yamini, “Analysis on Social Media Addiction using Data Mining Technique”, International Journal of Computer Applications (0975 – 8887) Volume 139 – No.7, pp. 23- 26, April 2016.

[12] Jessica Ho, Ping Ji, Weifang Chen, Raymond Hsieh, “Identifying google talk”, IEEE International Conference on Intelligence and Security Informatics, ISI ‘09, pp. 285-290, 2009.

[13] Mike Dickson, “An examination into AOL instant messenger 5.5 contact identification.”, Digital Investigation, ScienceDirect, vol. 3, issue 4, pp. 227-237, 2006.

[14] Mike Dickson, “An examination into yahoo messenger 7.0 contact identification”, Digital Investigation, ScienceDirect, vol. 3, issue 3, pp. 159-165, 2006.