TEXT BASED CHAT ANALYSIS

Submitted by

K. SAI NITHIN - 100521729081

T.SRI DIKSHITHA - 100521729087

D.MADHU SRI - 100521729020

T.SOUMYA - 100521729084

Under the guidance of

Dr. S PADMAJA

Associate Professor, Department of CSE

KMIT, Narayanguda - 5000029

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING (AIML)

UNIVERSITY COLLEGE OF ENGINEERING
Osmania University, Hyderabad 3-5-1206, Narayanguda,

Hyderabad – 500029

2023-2024

PROJECT OUTCOMES

p1

p2

L – LOW

M – MEDIUM

H – HIGH

PROJECT OUTCOMES MAPPING PROGRAM OUTCOMES

PO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
P1	H	H	-	-	-	M	-	-	H	L	-	L
P2	-	-	L	L	-	H	-	-	H	-	-	L

PROJECT OUTCOMES MAPPING PROGRAM SPECIFIC OUTCOMES

PSO	PSO1	PSO2
P1	-	L
P2	L	-

PROJECT OUTCOMES MAPPING PROGRAM EDUCATIONAL OBJECTIVES

PEO	PEO1	PEO2	PEO3	PEO4
P1	M	H	M	M
P2	-	M	-	L

ABSTRACT

WhatsApp is used by millions of users to express emotions and share feelings. The model presented in this project aims to perform sentimental and emotional analysis using textual messages and emojis used in WhatsApp chats. Code switching, which is quite prevalent over online conversations, is handled by the model by unifying and converting all the texts to a standard form. For each subject, multiple chats are taken; translated and using a neural network, each sentence and emoji is scored in a dimensional form. The composition of the emotions expressed by the subject (out of Happy, Sad, Bored, Fear, Anger and Excitement) are defined. The scores are added up for each subject. Throughout the analysis, the behavioral traits are extracted. It is determined that, if the subject likes to use emojis and if they use it as a replacement for words or as an add-on to express their emotions better. It is also observed that if the subject behaves differently on text according to the person in front of them with regard to these emotions and finally, if the subject is an introvert or extrovert.Text-based chat analysis has gained significant attention in recent years due to the proliferation of digital communication platforms and the wealth of data they generate. This project provides a comprehensive review of the methodologies, applications, and challenges associated with analyzing text-based chat interactions. By leveraging advanced analytical techniques and adopting ethical practices, researchers and practitioners can unlock the full potential of text-based chat data to inform decision-making and drive innovation in diverse fields.

LIST OF FIGURES

LIST OF TABLES

CHAPTER -1

INTRODUCTION

1. INTRODUCTION

1.1 Purpose of Project

The purpose of a text-based chat analysis project can vary depending on the specific goals and objectives of the project stakeholders. However, some common purposes include:

1. Understanding User Behavior: Analyzing text-based chat interactions can provide insights into user preferences, needs, and behaviors, which can inform product development, marketing strategies, and customer service improvements.

2. Improving Customer Service: By analyzing chat transcripts, organizations can identify common issues, trends, and pain points experienced by customers. This information can be used to optimize support processes, train customer service agents, and enhance overall customer satisfaction.

3. Sentiment Analysis: Text-based chat analysis can be used to gauge customer sentiment towards products, services, or brands. Sentiment analysis techniques can help organizations monitor customer satisfaction levels, identify areas for improvement, and respond to feedback in a timely manner.

4. Identifying Trends and Patterns: Analyzing large volumes of chat data can reveal emerging trends, patterns, and anomalies that may not be apparent through manual review. This information can

be valuable for strategic planning, market research, and competitive analysis.

5. Personalization and Recommendation: By analyzing chat interactions, organizations can gain insights into individual preferences and behaviors, allowing for personalized recommendations, content, and marketing messages tailored to the needs of specific users.

6. Risk Management and Compliance: Text-based chat analysis can help organizations identify potential risks, compliance issues, or regulatory violations within chat communications. This can be particularly important in industries such as finance, healthcare, and legal services.

7. Conversational AI Development: Analyzing chat data can provide valuable training data for developing and improving conversational AI systems, chatbots, and virtual assistants. By understanding how users interact and communicate in natural language, developers can create more effective and responsive conversational interfaces.

Overall, the purpose of a text-based chat analysis project is to extract actionable insights from chat data that can inform decision-making, drive innovation, and enhance user experiences across various domains and applications.

CHAPTER-2

SYSTEM REQUIREMENT SPECIFICATIONS

2. SYSTEM REQUIREMENT SPECIFICATIONS

System Requirement Specifications for Text-Based Chat Analysis

1. Functional Requirements:

- Data Collection: The system should be able to collect text-based chat data from various sources, including messaging platforms, social media, and customer service interactions.

- Preprocessing: Text preprocessing techniques such as tokenization, stemming, and stop-word removal should be implemented to clean and prepare the data for analysis.

- Analysis Algorithms: The system should support various analysis algorithms including sentiment analysis, topic modeling, entity recognition, and keyword extraction.

- Visualization: The system should provide visualization tools to present analysis results in an intuitive and interpretable manner, such as charts, graphs, and word clouds.

- Integration:The system should be able to integrate with external data sources and APIs for data enrichment and validation purposes.

- Customization: Users should have the ability to customize analysis parameters and algorithms based on specific requirements and use cases.

- Real-time Analysis: Optionally, the system may support real-time chat analysis to provide immediate insights and responses.

2. Non-Functional Requirements:

- Performance: The system should be capable of handling large volumes of chat data efficiently and provide timely analysis results.

- Scalability: The system architecture should be scalable to accommodate future growth and increasing data volumes.

- Security: Measures should be implemented to ensure the confidentiality, integrity, and privacy of chat data, including encryption, access controls, and user authentication.

- Reliability: The system should be reliable and robust, with minimal downtime and error handling mechanisms in place.

- Usability: The user interface should be intuitive and user-friendly, with features such as drag-and-drop functionality, customizable dashboards, and interactive visualizations.

- Compatibility:The system should be compatible with various operating systems, browsers, and devices to ensure accessibility and usability for a diverse user base.

- Documentation: Comprehensive documentation should be provided, including user manuals, installation guides, and API references.

- Compliance: The system should comply with relevant regulations and standards, such as data protection laws (e.g., GDPR) and industry-specific requirements (e.g., HIPAA for healthcare data).

3. Hardware and Software Requirements:

- Hardware: The system should run on standard hardware configurations, with specifications including CPU, memory, and storage capacity sufficient to support data processing and analysis tasks.

- Software:The system should be compatible with common programming languages (e.g., Python, Java), frameworks (e.g., TensorFlow, scikit-learn), and libraries (e.g., NLTK, spaCy) used for text analysis and machine learning.

By meeting these system requirement specifications, the text-based chat analysis system can effectively collect, process, and analyze chat data to derive valuable insights and support decision-making across various domains and applications.

CHAPTER -3

LITERATURE SURVEY

3. LITERATURE SURVEY

Literature Survey on Text-Based Chat Analysis

1. Introduction

Text-based chat analysis involves the use of computational techniques to analyze the content, structure, and patterns of text-based conversations. This field intersects natural language processing (NLP), machine learning, data mining, and social network analysis, providing insights into communication patterns, sentiment, and user behavior.

2. Key Areas of Research

1. **Sentiment Analysis**: Analyzing the emotions expressed in text to determine the sentiment (positive, negative, neutral) of the conversation. Techniques range from lexicon-based approaches to deep learning models.
    - Lexicon-based approaches use predefined lists of words associated with specific sentiments.
    - Machine learning approaches train models on labeled data to predict sentiment based on text features.
    - Deep learning approaches (e.g., LSTM, CNN, transformers) capture complex patterns in text data for more accurate sentiment analysis.
2. **Topic Modeling**: Identifying and extracting topics from chat data to understand the main themes discussed. Common methods include:
    - Latent Dirichlet Allocation (LDA): A generative probabilistic model that helps in discovering the hidden topics in a set of text documents.
    - Non-Negative Matrix Factorization (NMF): An algebraic method used for topic modeling that factors the document-term matrix into two lower-dimensional matrices.

3. **Conversation Analysis**: Studying the structure and dynamics of conversations, including turn-taking, dialogue management, and discourse analysis.
    - Turn-taking analysis examines how speakers alternate turns in a conversation.
    - Dialogue act recognition involves classifying segments of dialogue into categories such as questions, answers, statements, etc.
    - Discourse analysis investigates how coherence and meaning are constructed in dialogues.
4. **User Behavior Analysis**: Understanding user interactions, engagement patterns, and behavioral trends from chat data.
    - Behavioral clustering groups users based on similar interaction patterns.
    - Engagement metrics measure aspects like response times, message frequency, and session duration.
    - User profiling involves creating detailed profiles based on user interaction history.
5. **Spam and Abuse Detection**: Identifying and filtering out malicious or inappropriate content from chat conversations.
    - Rule-based methods use predefined rules and patterns to detect spam and abuse.
    - Machine learning methods train models to recognize patterns indicative of spam or abusive behavior.
6. **Language and Style Analysis:** Examining the linguistic features and stylistic elements of chat messages.
    - Stylometric analysis investigates authorship and stylistic fingerprints.
    - Language variation analysis studies differences in language use across different user groups or contexts.

3. Techniques and Tools

1. **Natural Language Processing (NLP)**: Core NLP techniques such as tokenization, part-of-speech tagging, named entity recognition, and syntactic parsing are foundational for chat analysis.
2. **Machine Learning**: Supervised and unsupervised learning algorithms are widely used, including SVM, decision trees, k-means clustering, and neural networks.
3. **Deep Learning**: Advanced deep learning models, especially those based on transformers (e.g., BERT, GPT), are highly effective in understanding and generating human-like text.
4. **Statistical Methods**: Statistical analysis and hypothesis testing are employed to derive meaningful insights and validate findings from chat data.

4. Applications

1. **Customer Support**: Enhancing customer service through automated chatbots, sentiment analysis, and real-time support analytics.
2. **Social Media Monitoring**: Tracking and analyzing conversations on social media platforms to gauge public opinion and detect trends.
3. **Healthcare**: Monitoring patient communications for early detection of mental health issues or disease outbreaks.
4. **Education**: Analyzing student interactions in online learning environments to improve engagement and learning outcomes.
5. **Market Research**: Understanding consumer preferences and feedback through the analysis of chat logs from various platforms.

5. Challenges and Future Directions

1. **Data Privacy and Ethics**: Ensuring user privacy and adhering to ethical standards while analyzing chat data.
2. **Scalability**: Developing methods that can efficiently process and analyze large-scale chat data in real-time.
3. **Multimodal Analysis:** Integrating text analysis with other data types (e.g., images, videos) for a more comprehensive understanding.

4. **Cross-Lingual Analysis**: Addressing challenges related to analyzing chats in multiple languages and dialects.

6. Conclusion

Text-based chat analysis is a rapidly evolving field with significant potential across various domains. Continued advancements in NLP, machine learning, and deep learning will drive more sophisticated and accurate analysis, providing deeper insights into human communication and behavior.

CHAPTER-4

SYSTEM DESIGN

4. SYSTEM DESIGN

System Design for Text-Based Chat Analysis

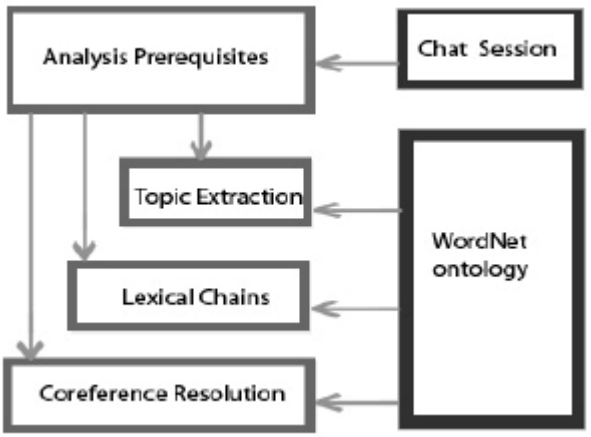
1. Introduction

Designing a system for text-based chat analysis involves integrating multiple components that handle data ingestion, preprocessing, analysis, and visualization. The system should be scalable, efficient, and capable of providing real-time insights. This design outlines the architecture, key components, technologies, and data flow for a comprehensive chat analysis system.

2. Architecture Overview

The system architecture for text-based chat analysis can be divided into several layers:

1. Data Ingestion Layer: Collects chat data from various sources.
2. Data Storage Layer: Stores raw and processed data.
3. Data Processing Layer: Handles data cleaning, transformation, and enrichment.
4. Analysis Layer: Performs text analysis tasks such as sentiment analysis, topic modeling, and user behavior analysis.
5. Visualization and Reporting Layer: Provides insights and visualizations for end-users.
6. Application Layer: Interfaces with users and external systems.



3. Key Components

1. Data Ingestion

- Sources: Chat applications, social media platforms, customer service logs, forums, etc**.**

2. Data Processing

- Preprocessing: Data cleaning, tokenization, normalization using Apache Spark or Apache Flink.

- Transformation: Extracting features, sentiment scores, and other relevant attributes.

- Enrichment: Adding metadata, user profiles, and contextual information.

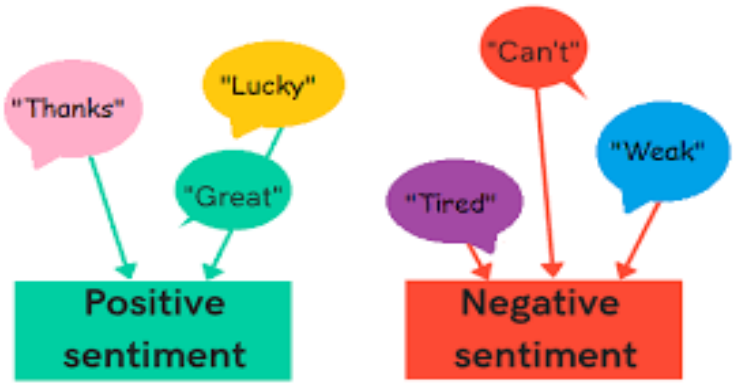
3. Analysis

- Sentiment Analysis: Using NLP libraries and frameworks like NLTK, SpaCy, and transformers

- Topic Modeling: Implementing LDA or Scikit-learn.

- Behavior Analysis: Applying clustering algorithms and behavioral analytics using Scikit-learn or TensorFlow.

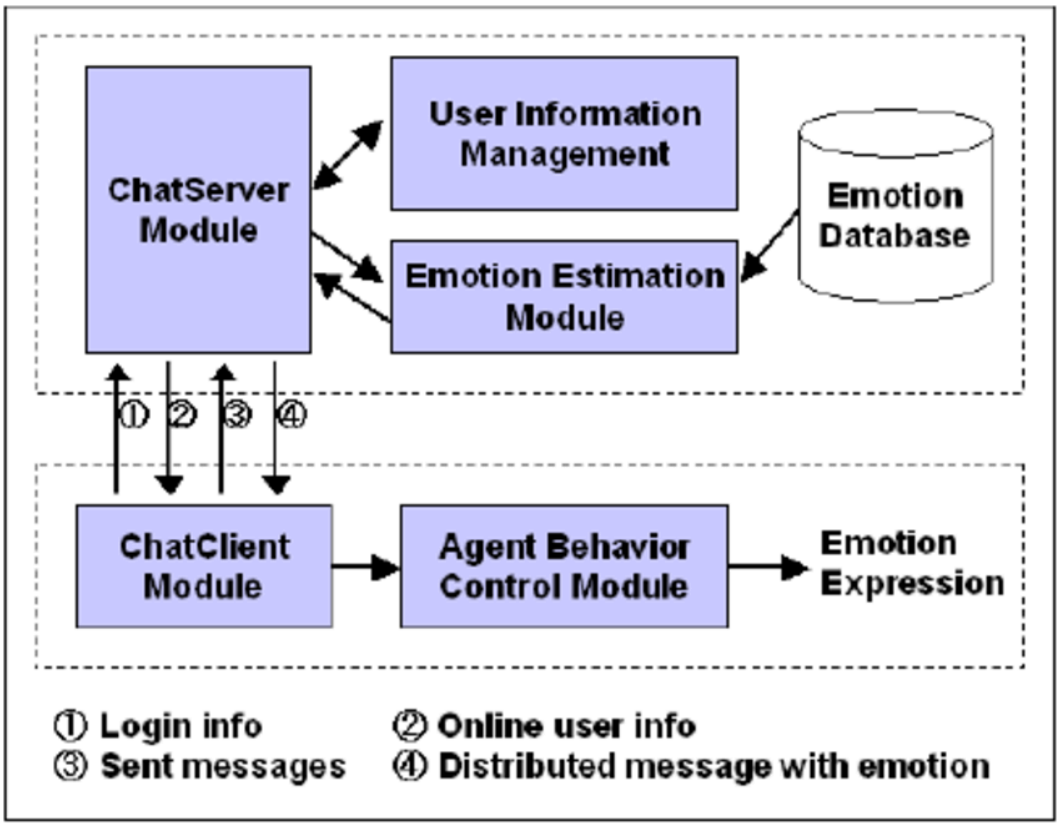
- Spam and Abuse Detection: Rule-based filtering and machine learning models to identify and filter unwanted content.



4. Visualization and Reporting

- Dashboards: Interactive dashboards using tools like Tableau, Power BI, or custom dashboards with Plotly**.**

- Reports: Automated report generation and distribution using Jupyter notebooks, LaTeX, or report generation frameworks.



5. Application Layer

- APIs: RESTful APIs or GraphQL for interfacing with other systems and services.

- Web Interface: Frontend applications using frameworks like React, Angular

- User Authentication: Secure access control using or custom authentication mechanisms.

4. Data Flow

1. Data Ingestion:

- Collect chat data from various sources.

2. Data Processing:

- Preprocess data (cleaning, tokenization)

- Transform data to extract features and enrich with additional information.

3. Analysis:

- Apply sentiment analysis models to determine the sentiment of messages.

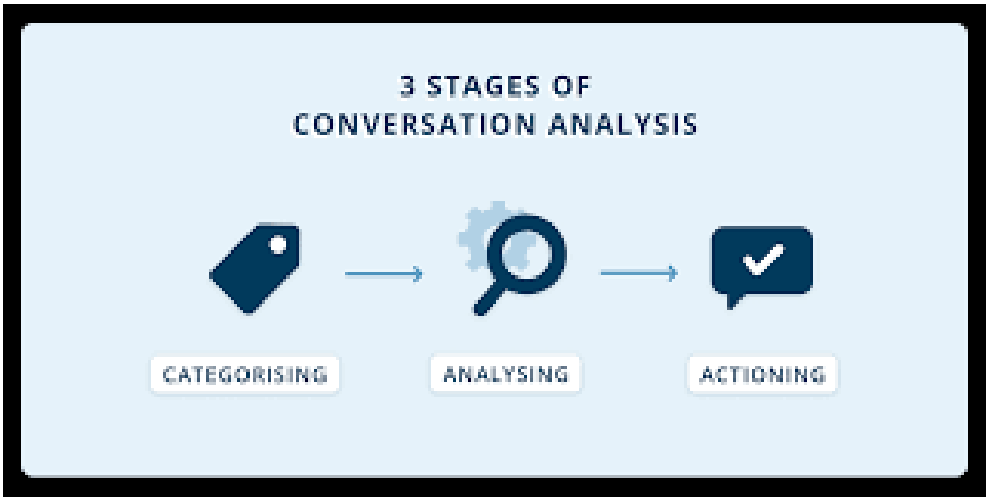
- Perform topic modeling to identify key themes and topics.

- Analyze user behavior to identify patterns and trends.

- Detect and filter spam or abusive messages.

- Provide APIs for external access to processed data and analysis results.

- Develop web interfaces for user interaction and data exploration.



5. Conclusion

The design of a text-based chat analysis system involves integrating multiple technologies and components to handle large-scale data efficiently.This comprehensive approach ensures that the system is robust, scalable, and capable of delivering real-time analytics.

CHAPTER -5

IMPLEMENTATION

5. IMPLEMENTATION

Importing libraries

```
import re
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

from collections import Counter, defaultdict
from wordcloud import WordCloud, STOPWORDS
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
import string
from itertools import combinations

from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.decomposition import LatentDirichletAllocation, TruncatedSVD, NMF
from sklearn.metrics import pairwise_distances_argmin_min, silhouette_score

import warnings
warnings.filterwarnings('ignore')
```

Extracting date time, contact-name, and message from the chat logs

In [5]:

```
def date_time(s):
    pattern = '^([0-9]+(\/)([0-9]+)(\/)[0-9]+, ([0-9]+):([0-9]+)\s(PM|AM|am|pm) - )'
    result = re.match(pattern, s)
    if result:
        return True
    return False

def contact(s):
    s=s.split(":")
    if len(s) == 2:
        return True
    return False

def getmsg(line):
    splitline = line.split(' - ')
    date, time = splitline[0].split(', ')
    msg = " ".join(splitline[1:])

    if contact(msg):
        split_msg = msg.split(': ')
        author = split_msg[0]
        msg = " ".join(split_msg[1:])
    else:
        author = None
    return date, time, author, msg
```

In [6]:

In [6]:
```
data = []
conversation = 'chats/WhatsApp Chat with Vishal Sir.txt'
with open(conversation, encoding="utf-8") as fp:
    fp.readline()
    msgBuffer = []
    date, time, author=None, None, None
    while True:
        line = fp.readline()
        if not line:
            break
        line = line.strip()

        if date_time(line):
            if len(msgBuffer) > 0 :
                data.append([date, time, author, " ".join(msgBuffer)])
            msgBuffer.clear()
            date, time, author, msg = getmsg(line)
            msgBuffer.append(msg)
        else:
            msgBuffer.append(line)
```

In [7]:
```
msgBuffer
```

Out[7]:
```
['It's ok sir', 'Any time']
```

Converting the extracted data into DataFrame

In [8]:
```
data = pd.DataFrame(data, columns=["Date", "Time", "Contact", "Message"])
data['Date'] = pd.to_datetime(data['Date'])
data = data.dropna()
data
```

Out[8]:

	Date	Time	Contact	Message
0	2021-03-31	7:41 PM	Sai Nithin Kadarla	Sir
1	2021-03-31	7:41 PM	Sai Nithin Kadarla	Mee voice msg by mistake lo mana clg group lo ...
2	2021-03-31	7:43 PM	Sai Nithin Kadarla	Sir ide
3	2021-03-31	7:44 PM	Sai Nithin Kadarla	Sir this too
4	2021-03-31	7:47 PM	Sai Nithin Kadarla	<Media omitted>

Cleaning the Data

In [9]:
```
stop_words = ["i", "me", "my", "myself", "we", "our", "ours", "ourselves", "you", "your", "yours", "yourself",
              "yourselves", "he", "him", "his", "himself", "she", "her", "hers", "herself", "it", "its", "itself",
              "they", "them", "their", "theirs", "themselves", "what", "which", "who", "whom", "this", "that", "these",
              "those", "am", "is", "are", "was", "were", "be", "been", "being", "have", "has", "had", "having", "do",
              "does", "did", "doing", "a", "an", "the", "and", "but", "if", "or", "because", "as", "until", "while",
              "of", "at", "by", "for", "with", "about", "against", "between", "into", "through", "during", "before",
              "after", "above", "below", "to", "from", "up", "down", "in", "out", "on", "off", "over", "under", "again",
              "further", "then", "once", "here", "there", "when", "where", "why", "how", "all", "any", "both", "each",
              "few", "more", "most", "other", "some", "such", "no", "nor", "not", "only", "own", "same", "so", "than",
              "too", "very", "s", "t", "can", "will", "just", "don", "should", "now"]
```

Cleaning the deleted messages and media ommitio logs

In [10]:
```
data["Message"] = data['Message'][data["Message"] != "<Media omitted>"]
data.dropna(axis=0, inplace=True)

string_to_match = " deleted this message"
data = data[~data["Message"].str.contains(string_to_match, case=False)]
data
```

Cleaning the messages from punctuations and stopwords and tokenized the messages

In [11]:
```
data['Cleaned_message'] = data["Message"].apply(lambda x: x.lower().translate(str.maketrans(' ', " ", string.punctuation)
data['Tokenized_words'] = data["Cleaned_message"].apply(lambda y: [word for word in word_tokenize(y) if word not in STOPW
```

Applying the lemmatization techniques

In [13]:
```
lemmatizer = WordNetLemmatizer()
data["Lemmatized"] = [[lemmatizer.lemmatize(token) for token in token_list] for token_list in data["Tokenized_words"]]
```

Sentiment analysis

Sentiments of each message

In [14]:
```
from nltk.sentiment.vader import SentimentIntensityAnalyzer
sentiments = SentimentIntensityAnalyzer()

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

Normalizing the values of the sentiment Analyzer

In [15]:
```
data["Positive"] = data["Positive"].apply(lambda x: np.ceil(x) if x - np.floor(x) >= 0.5 else np.floor(x))
data["Negative"] = data["Negative"].apply(lambda x: np.ceil(x) if x - np.floor(x) >= 0.5 else np.floor(x))
data["Neutral"] = data["Neutral"].apply(lambda x: np.ceil(x) if x - np.floor(x) >= 0.5 else np.floor(x))
```

In [16]:
```
data
```

Over all sentiment of the chat

In [17]:
```
pos = sum(data['Positive'])
neg = sum(data['Negative'])
neu = sum(data['Neutral'])

def score(a,b,c):
    if a>b and a>c:
        print('postive')
    elif b>a and b>c:
        print('negative')
    else:
        print("neutral")

score(pos, neg, neu)

pos, neg, neu
```

```
neutral
```

Out[17]: (23.0, 0.0, 62.0)

Added the sentiment in text form

In [18]:
```
def compare_values(row):
    if row['Positive'] > row['Negative'] and row['Positive'] > row['Neutral']:
        return 'positive'
    elif row['Negative'] > row['Positive'] and row['Negative'] > row['Neutral']:
        return 'negative'
    else:
        return 'neutral'
```

In [19]:
```
data["sentiment"] = data.apply(compare_values, axis = 1)
data
```

Extracting the sentiment trends into seperate dataframe

```
grouped = data.groupby('Date')

def most_common_sentiment(series):
    return Counter(series).most_common(1)[0][0]

# Create a new DataFrame with the required columns
sentiment_logs = grouped.agg(
    start_sentiment=('sentiment', 'first'),
    end_sentiment=('sentiment', 'last'),
    most_common_sentiment=('sentiment', most_common_sentiment)
).reset_index()

# Print the result DataFrame
print(sentiment_logs)
```

Extracted sentiment at start and end of the conversation

```
grouped = data.groupby('Date')

# Add columns for the first and last Sentiment values within each group
data['Start Conversation'] = grouped['sentiment'].transform('first')
data['Stop Conversation'] = grouped['sentiment'].transform('last')
data
```

Extracted sentiment at start and end of the conversation

```
grouped = data.groupby('Date')

# Add columns for the first and last Sentiment values within each group
data['Start Conversation'] = grouped['sentiment'].transform('first')
data['Stop Conversation'] = grouped['sentiment'].transform('last')
data
```

Analysing the word frequencies

Frequency of each word

```
combined_words = [word for sublist in data['Lemmatized'] for word in sublist]

fdist = nltk.FreqDist(combined_words)
```

Most common words

In [23]:
```
print("The most common 10 words are: \n")

for i,j in fdist.most_common(10):
    print(i, end=", ")
```

```
The most common 10 words are:

sir, ok, thank, send, please, certificate, one, name, call, lo,
```

Topic modeling

In [24]:
```
data["Tokenized_mgs"] = data["Tokenized_words"].apply(lambda x: " ".join(x))
data
```

Latent Dirichlet Allocation (LDA)

In [25]:
```
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(data['Tokenized_mgs'])


lda_model = LatentDirichletAllocation(n_components=5, random_state=0)
lda_model.fit(X)


def print_topics(model, vectorizer, top_n=10):
    for idx, topic in enumerate(model.components_):
        print(f"Topic {idx}:")
        print([vectorizer.get_feature_names_out()[i] for i in topic.argsort()[:-top_n - 1:-1]])

print_topics(lda_model, vectorizer)


topic_distribution = lda_model.transform(X)


def most_relevant(topic_distribution):
    most_relevant_topics = []
    for distribution in topic_distribution:
        most_relevant_topic = distribution.argmax()
        most_relevant_topics.append(most_relevant_topic)

    return most_relevant_topics

relevant_topics = most_relevant(topic_distribution)


data['LDA'] = relevant_topics
```

Latent Semantic Analysis (LSA)

In [26]:
```
tfvectorizer = TfidfVectorizer()
X = tfvectorizer.fit_transform(data["Tokenized_mgs"])


lsa_model = TruncatedSVD(n_components=5, random_state=0)
lsa_model.fit(X)


def print_topics(model, vectorizer, top_n=10):
    terms = vectorizer.get_feature_names_out()
    for idx, component in enumerate(model.components_):
        terms_in_topic = [terms[i] for i in component.argsort()[:-top_n - 1:-1]]
        print(f"Topic {idx}: {' '.join(terms_in_topic)}")

print_topics(lsa_model, tfvectorizer)


topic_distribution = lsa_model.transform(X)


def most_relevant(topic_distribution):
    most_relevant_topics = []
    for distribution in topic_distribution:
        # Get the topic with the highest value
        most_relevant_topic = distribution.argmax()
        most_relevant_topics.append(most_relevant_topic)
    return most_relevant_topics


relevant_topics = most_relevant(topic_distribution)


data['LSA'] = relevant_topics
```

Probabilistic Latent Semantic Analysis (pLSA)

In [28]:

```
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(data["Tokenized_mgs"])


plsa_model = LatentDirichletAllocation(n_components=5, doc_topic_prior=0.1, topic_word_prior=0.1, random_state=0)
plsa_model.fit(X)


def print_topics(model, vectorizer, top_n=10):
    for idx, topic in enumerate(model.components_):
        print(f"Topic {idx}:")
        print([vectorizer.get_feature_names_out()[i] for i in topic.argsort()[:-top_n - 1:-1]])

print_topics(plsa_model, vectorizer)


def most_relevant(topic_distribution):
    most_relevant_topics = []
    for distribution in topic_distribution:
        # Get the topic with the highest probability
        most_relevant_topic = distribution.argmax()
        most_relevant_topics.append(most_relevant_topic)
    return most_relevant_topics


topic_distribution = plsa_model.transform(X)


relevant_topics = most_relevant(topic_distribution)


data['pLSA'] = relevant_topics
```

Coherence scores

In [31]:

```
def print_topics(model, vectorizer, top_n=10):
    terms = vectorizer.get_feature_names_out()
    for idx, component in enumerate(model.components_):
        terms_in_topic = [terms[i] for i in component.argsort()[:-top_n - 1:-1]]
        print(f"Topic {idx}: {' '.join(terms_in_topic)}")


def coherence_score(model, vectorizer, documents, top_n=10):
    topics = model.components_
    words = vectorizer.get_feature_names_out()
    co_occurrences = defaultdict(int)
    for topic in topics:
        top_words = [words[i] for i in topic.argsort()[:-top_n - 1:-1]]
        for word_pair in combinations(top_words, 2):
            co_occurrences[word_pair] += sum(1 for doc in documents if word_pair[0] in doc and word_pair[1] in doc)
    coherence = sum(co_occurrences.values()) / len(co_occurrences)
    return coherence

# Compute coherence scores
lda_coherence = coherence_score(lda_model, tfvectorizer, data["Tokenized_mgs"])
nmf_coherence = coherence_score(nmf_model, vectorizer, data["Tokenized_mgs"])
plsa_coherence = coherence_score(plsa_model, tfvectorizer, data["Tokenized_mgs"])
lsa_coherence = coherence_score(lsa_model, vectorizer, data["Tokenized_mgs"])

print("\nCoherence Scores:")
print(f"LDA Coherence: {print_topics(lda_model, tfvectorizer)}\n")
print(f"NMF Coherence: {print_topics(nmf_model, vectorizer)}\n")
print(f"PLSA Coherence: {print_topics(plsa_model,tfvectorizer)}\n")
print(f"LSA Coherence: {print_topics(lsa_model, vectorizer)}\n")

print("\nCoherence Scores:")
print(f"LDA Coherence: {lda_coherence}")
print(f"NMF Coherence: {nmf_coherence}")
print(f"PLSA Coherence: {plsa_coherence}")
print(f"LSA Coherence: {lsa_coherence}")
```

Topic diversity

In [32]:

```
def topic_diversity(topics, top_n=10):
    unique_words = set()
    total_words = 0
    for topic in topics:
        top_words = [vectorizer.get_feature_names_out()[i] for i in topic.argsort()[:-top_n - 1:-1]]
        unique_words.update(top_words)
        total_words += len(top_words)
    return len(unique_words) / total_words

lda_diversity = topic_diversity(lda_model.components_)
nmf_diversity = topic_diversity(nmf_model.components_)
plsa_diversity = topic_diversity(plsa_model.components_)
lsa_diversity = topic_diversity(lsa_model.components_)

print(f"LDA Diversity: {lda_diversity}")
print(f"NMF Diversity: {nmf_diversity}")
print(f"pLSA Diversity: {plsa_diversity}")
print(f"LSA Diversity: {lsa_diversity}")
```

```
LDA Diversity: 0.84
NMF Diversity: 0.98
pLSA Diversity: 0.82
LSA Diversity: 0.8
```

Coherence scores

In [31]:

```
def print_topics(model, vectorizer, top_n=10):
    terms = vectorizer.get_feature_names_out()
    for idx, component in enumerate(model.components_):
        terms_in_topic = [terms[i] for i in component.argsort()[:-top_n - 1:-1]]
        print(f"Topic {idx}: {' '.join(terms_in_topic)}")


def coherence_score(model, vectorizer, documents, top_n=10):
    topics = model.components_
    words = vectorizer.get_feature_names_out()
    co_occurrences = defaultdict(int)
    for topic in topics:
        top_words = [words[i] for i in topic.argsort()[:-top_n - 1:-1]]
        for word_pair in combinations(top_words, 2):
            co_occurrences[word_pair] += sum(1 for doc in documents if word_pair[0] in doc and word_pair[1] in doc)
    coherence = sum(co_occurrences.values()) / len(co_occurrences)
    return coherence

# Compute coherence scores
lda_coherence = coherence_score(lda_model, tfvectorizer, data["Tokenized_mgs"])
nmf_coherence = coherence_score(nmf_model, vectorizer, data["Tokenized_mgs"])
plsa_coherence = coherence_score(plsa_model, tfvectorizer, data["Tokenized_mgs"])
lsa_coherence = coherence_score(lsa_model, vectorizer, data["Tokenized_mgs"])

print("\nCoherence Scores:")
print(f"LDA Coherence: {print_topics(lda_model, tfvectorizer)}\n")
print(f"NMF Coherence: {print_topics(nmf_model, vectorizer)}\n")
print(f"PLSA Coherence: {print_topics(plsa_model,tfvectorizer)}\n")
print(f"LSA Coherence: {print_topics(lsa_model, vectorizer)}\n")

print("\nCoherence Scores:")
print(f"LDA Coherence: {lda_coherence}")
print(f"NMF Coherence: {nmf_coherence}")
print(f"PLSA Coherence: {plsa_coherence}")
print(f"LSA Coherence: {lsa_coherence}")
```


Topic diversity

In [32]:

```
def topic_diversity(topics, top_n=10):
    unique_words = set()
    total_words = 0
    for topic in topics:
        top_words = [vectorizer.get_feature_names_out()[i] for i in topic.argsort()[:-top_n - 1:-1]]
        unique_words.update(top_words)
        total_words += len(top_words)
    return len(unique_words) / total_words

lda_diversity = topic_diversity(lda_model.components_)
nmf_diversity = topic_diversity(nmf_model.components_)
plsa_diversity = topic_diversity(plsa_model.components_)
lsa_diversity = topic_diversity(lsa_model.components_)

print(f"LDA Diversity: {lda_diversity}")
print(f"NMF Diversity: {nmf_diversity}")
print(f"pLSA Diversity: {plsa_diversity}")
print(f"LSA Diversity: {lsa_diversity}")
```

```
LDA Diversity: 0.84
NMF Diversity: 0.98
pLSA Diversity: 0.82
LSA Diversity: 0.8
```

Visualizations

Generating word clouds from the frequencies of each word

```
wordcloud = WordCloud().generate_from_frequencies(fdist)
```

```
plt.figure(figsize=(10, 6))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")  # Hide axis
plt.show()
```

Topic visualizations

In [35]:

```
num_topics = 5

vectorizer = TfidfVectorizer()
X_tfidf = vectorizer.fit_transform(data['Tokenized_mgs'])

# Create a Document-Term Matrix using CountVectorizer for LDA
count_vectorizer = CountVectorizer()
X_count = count_vectorizer.fit_transform(data["Tokenized_mgs"])

# Create Word Clouds
def create_word_cloud(model, vectorizer, num_topics):
    terms = vectorizer.get_feature_names_out()
    for idx, component in enumerate(model.components_):
        word_freq = {terms[i]: component[i] for i in component.argsort()[:-11:-1]}
        wordcloud = WordCloud(width=400, height=200, background_color='white').generate_from_frequencies(word_freq)
        plt.figure(figsize=(10, 5))
        plt.imshow(wordcloud, interpolation='bilinear')
        plt.axis("off")
        plt.title(f"Topic {idx}")
        plt.show()

print("\nWord Clouds for LDA:")
create_word_cloud(lda_model, count_vectorizer, num_topics)

print("\nWord Clouds for NMF:")
create_word_cloud(nmf_model, vectorizer, num_topics)

print("\nWord Clouds for pLSA:")
create_word_cloud(plsa_model, count_vectorizer, num_topics)

print("\nWord Clouds for LSA:")
create_word_cloud(lsa_model, vectorizer, num_topics)

# Create Bar Charts
def create_bar_chart(model, vectorizer, num_topics, top_n=10):
```

```
create_word_cloud(lda_model, count_vectorizer, num_topics)

print("\nWord Clouds for NMF:")
create_word_cloud(nmf_model, vectorizer, num_topics)

print("\nWord Clouds for pLSA:")
create_word_cloud(plsa_model, count_vectorizer, num_topics)

print("\nWord Clouds for LSA:")
create_word_cloud(lsa_model, vectorizer, num_topics)

# Create Bar Charts
def create_bar_chart(model, vectorizer, num_topics, top_n=10):
    terms = vectorizer.get_feature_names_out()
    for idx, component in enumerate(model.components_):
        top_terms = [(terms[i], component[i]) for i in component.argsort()[:-top_n - 1:-1]]
        df_top_terms = pd.DataFrame(top_terms, columns=['Term', 'Weight'])
        plt.figure(figsize=(10, 5))
        sns.barplot(x='Weight', y='Term', data=df_top_terms)
        plt.title(f"Top Terms for Topic {idx}")
        plt.show()

print("\nBar Charts for LDA:")
create_bar_chart(lda_model, count_vectorizer, num_topics)

print("\nBar Charts for NMF:")
create_bar_chart(nmf_model, vectorizer, num_topics)

print("\nBar Charts for pLSA:")
create_bar_chart(plsa_model, count_vectorizer, num_topics)

print("\nBar Charts for LSA:")
create_bar_chart(lsa_model, vectorizer, num_topics)
```

```
Word Clouds for LDA:
```

Visualizations

Generating word clouds from the frequencies of each word

```
wordcloud = WordCloud().generate_from_frequencies(fdist)
```

```
plt.figure(figsize=(10, 6))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")  # Hide axis
plt.show()
```

In [37]:

```
def get_sentiment(text):
    sentiment = sid.polarity_scores(text)
    return sentiment

sentiments = data['Tokenized_mgs'].apply(get_sentiment)
df = pd.concat([data, sentiments.apply(pd.Series)], axis=1)

# Aggregate sentiment scores by date
data["Date"] = pd.to_datetime(data['Date'])
data['Date'] = data['Date'].dt.date
daily_sentiment = df.groupby('Date')[['pos', 'neu', 'neg', 'compound']].mean().reset_index()

# Plot sentiment trends
plt.figure(figsize=(14, 7))
plt.plot(daily_sentiment['Date'], daily_sentiment['pos'], marker='o', label='Positive', color='green')
plt.plot(daily_sentiment['Date'], daily_sentiment['neu'], marker='o', label='Neutral', color='blue')
plt.plot(daily_sentiment['Date'], daily_sentiment['neg'], marker='o', label='Negative', color='red')
plt.plot(daily_sentiment['Date'], daily_sentiment['compound'], marker='o', label='Compound', color='purple')
plt.title('Sentiment Trend Over Time')
plt.xlabel('Date')
plt.ylabel('Sentiment Score')
plt.legend()
plt.grid(True)
plt.show()
```

In [38]:

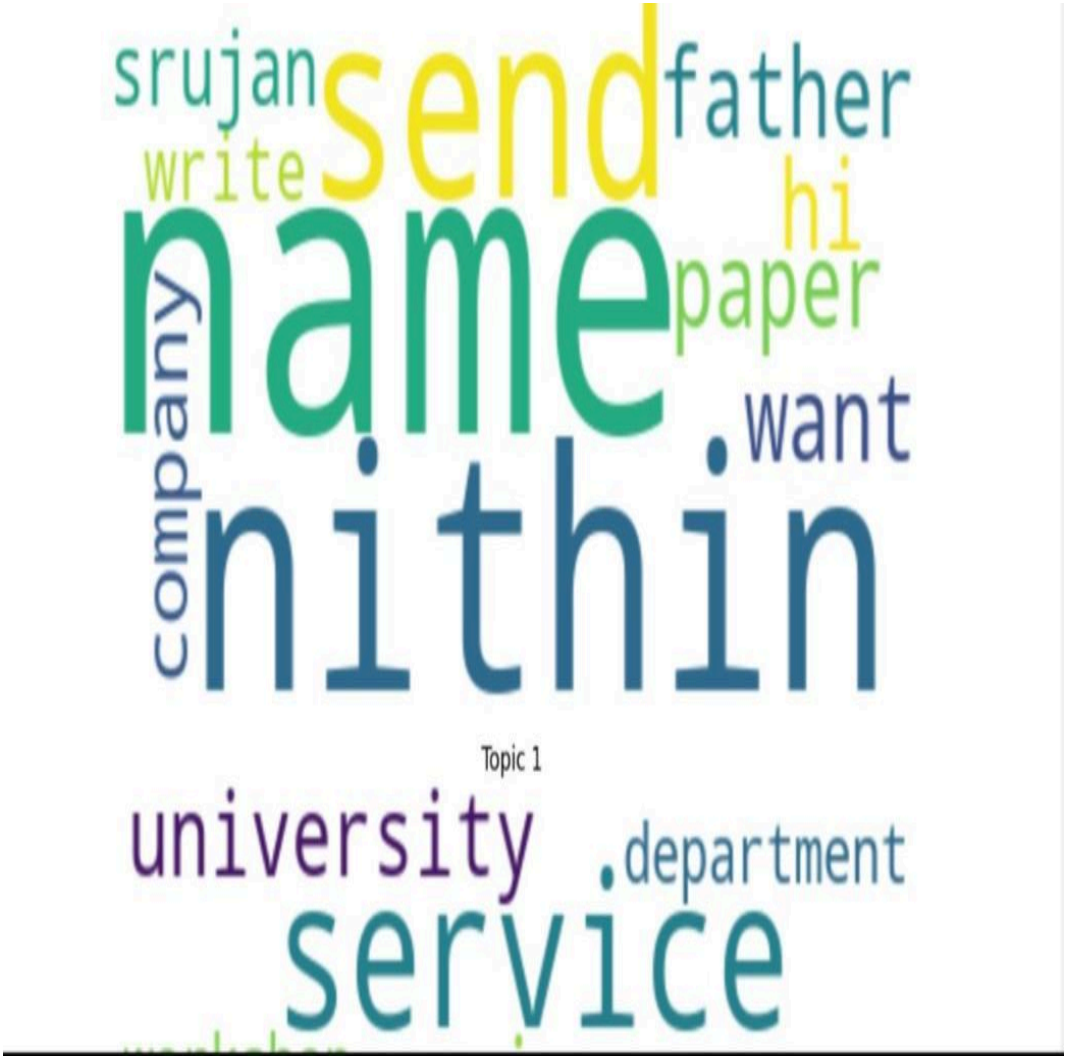
```
message_counts = data.groupby('Date').size().reset_index(name='Message Count')

# Plot message volumes over time
plt.figure(figsize=(14, 7))
sns.lineplot(data=message_counts, x='Date', y='Message Count', marker='o')
plt.title('Message Volumes Over Time')
plt.xlabel('Date')
plt.ylabel('Number of Messages')
plt.grid(True)
plt.show()
```

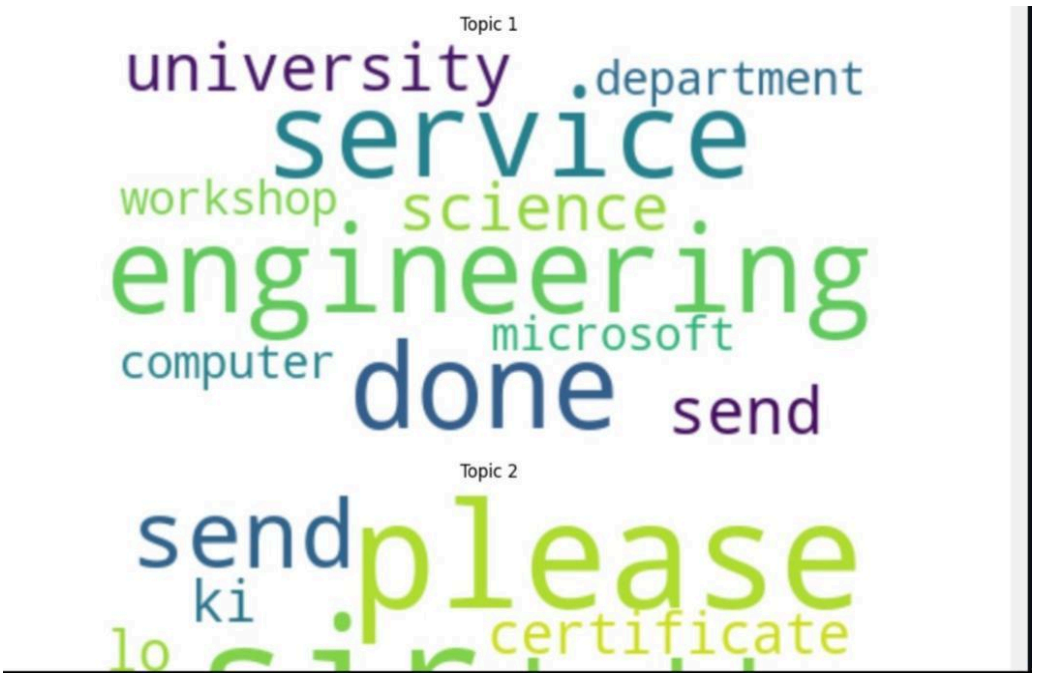
CHAPTER-6

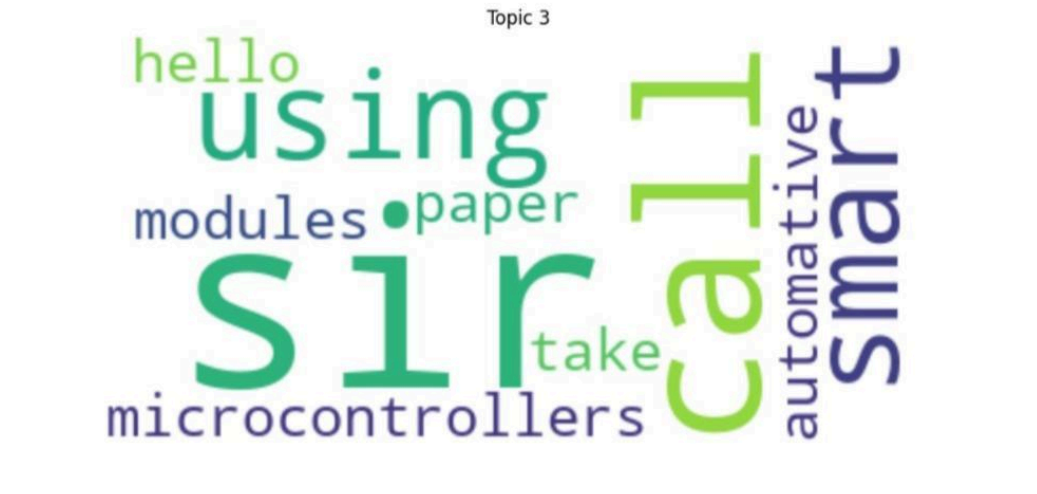
TESTING

6. TESTING

OUTPUTS OF VISUALIZATION



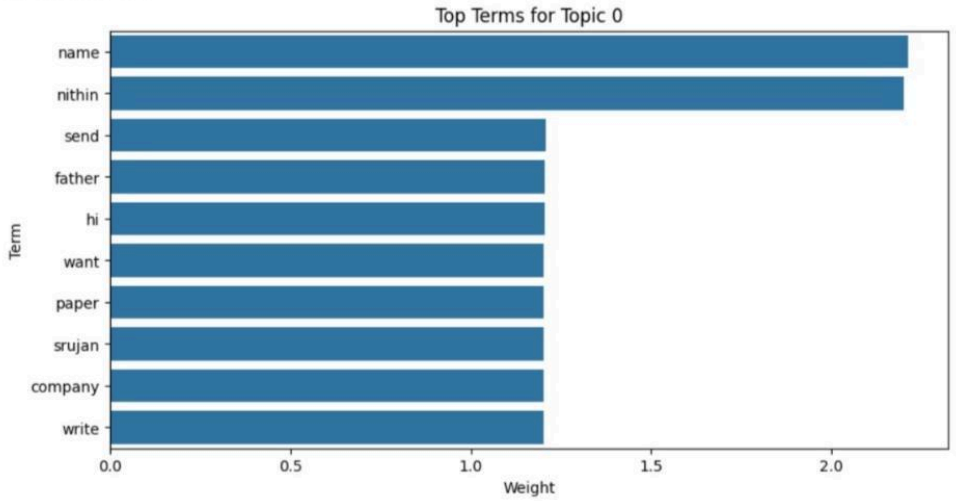

Topic 1
university
department
service
workshop
science
engineering
microsoft
computer
done
send
Topic 2
send
please
ki
certificate

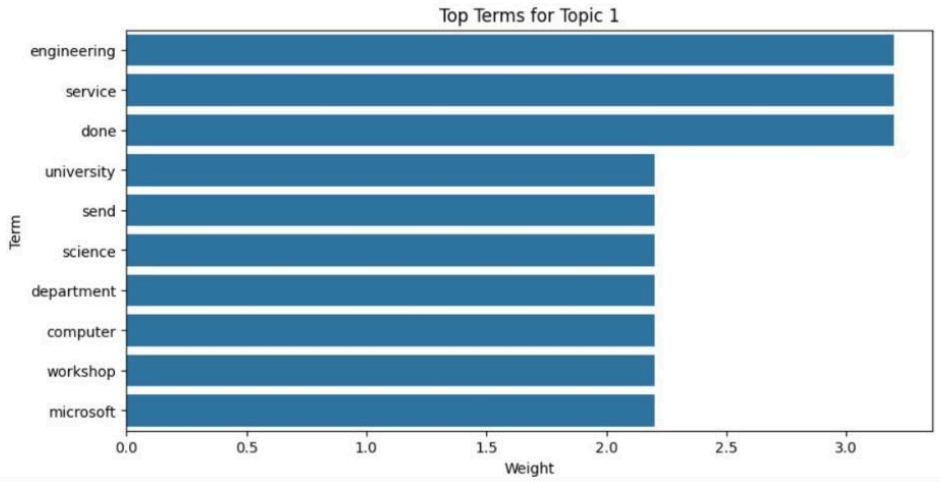

Topic 3
hello
using
call
smart
automative
modules
paper
sir
take
microcontrollers

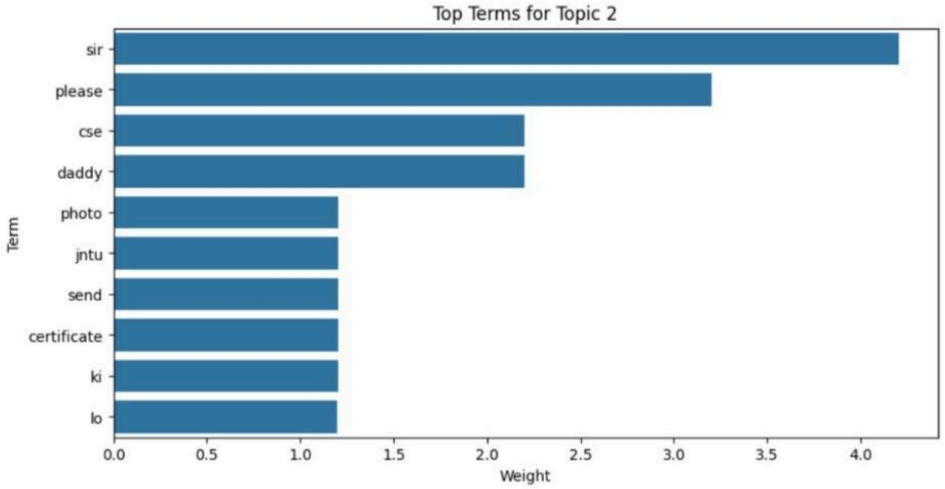

Topic 4
thank
ok
good
certificate
please
project
sir
lo
morning
one
Word Clouds for NMF:

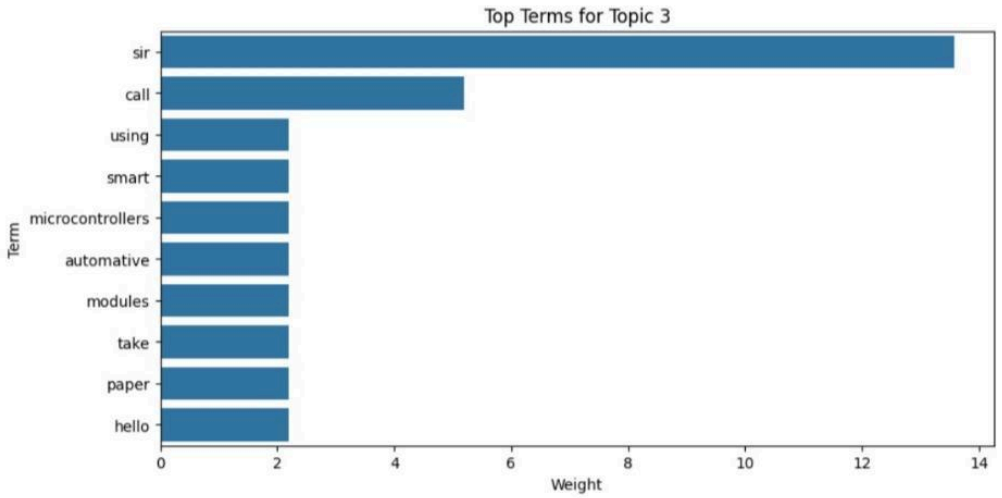
BAR CHARTS FOR LDA

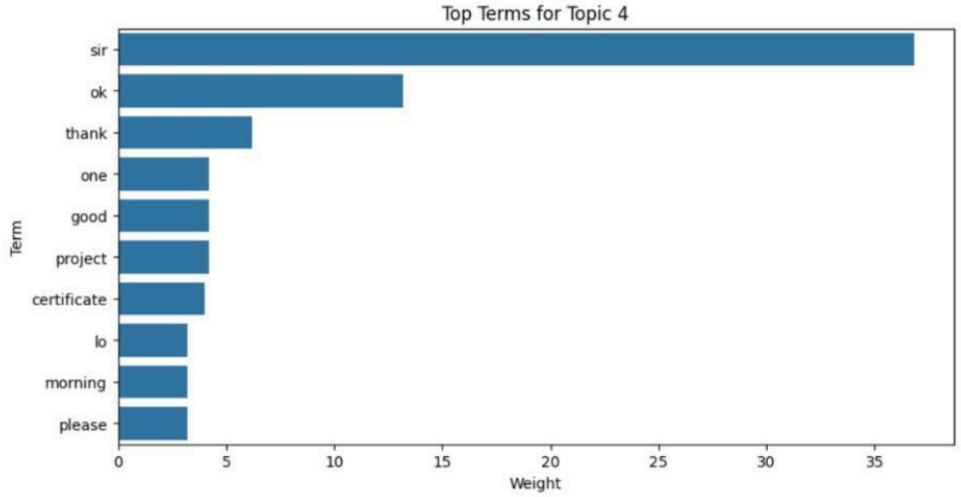
Bar Charts for LDA:








Top Terms for Topic 3
sir
call
using
smart
microcontrollers
automative
modules
take
paper
hello
Term
0
2
4
6
8
10
12
14
Weight

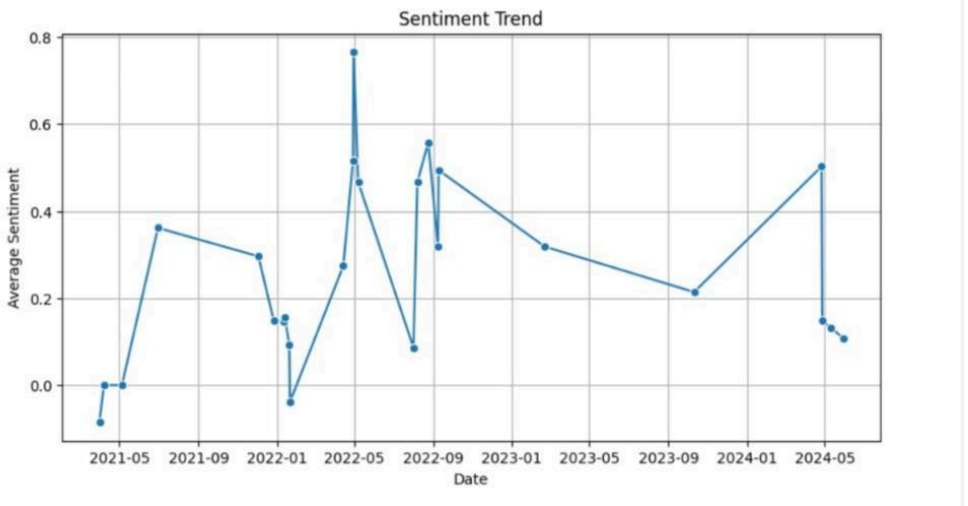

Top Terms for Topic 4
sir
ok
thank
one
good
project
certificate
lo
morning
please
Term
0
5
10
15
20
25
30
35
Weight

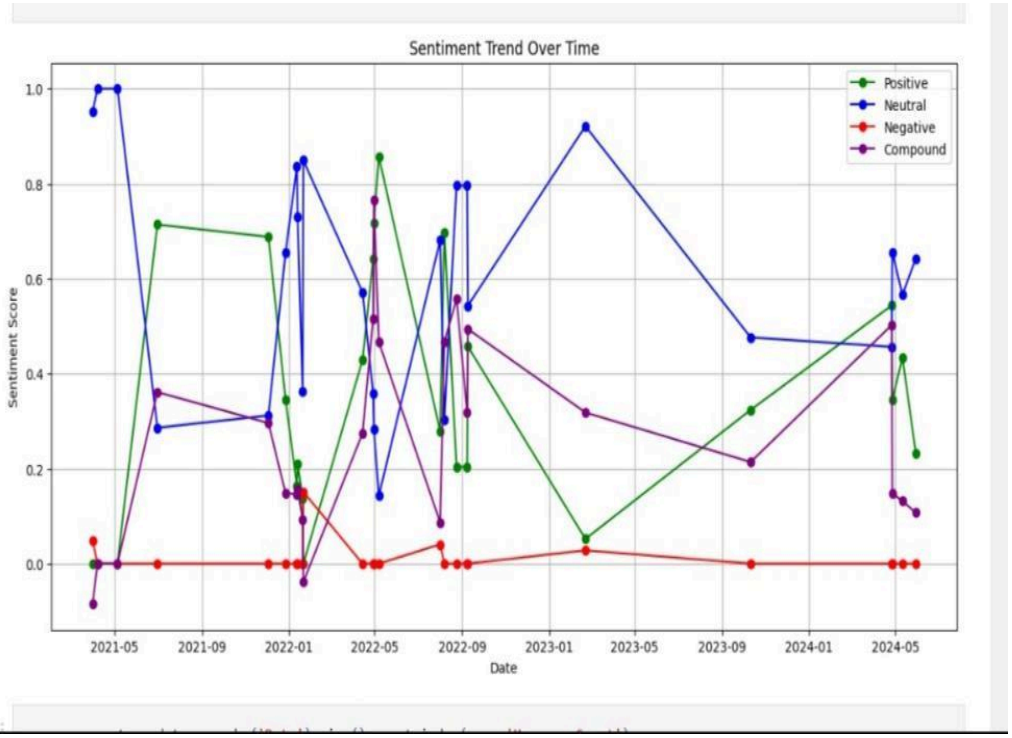
CHAPTER -7

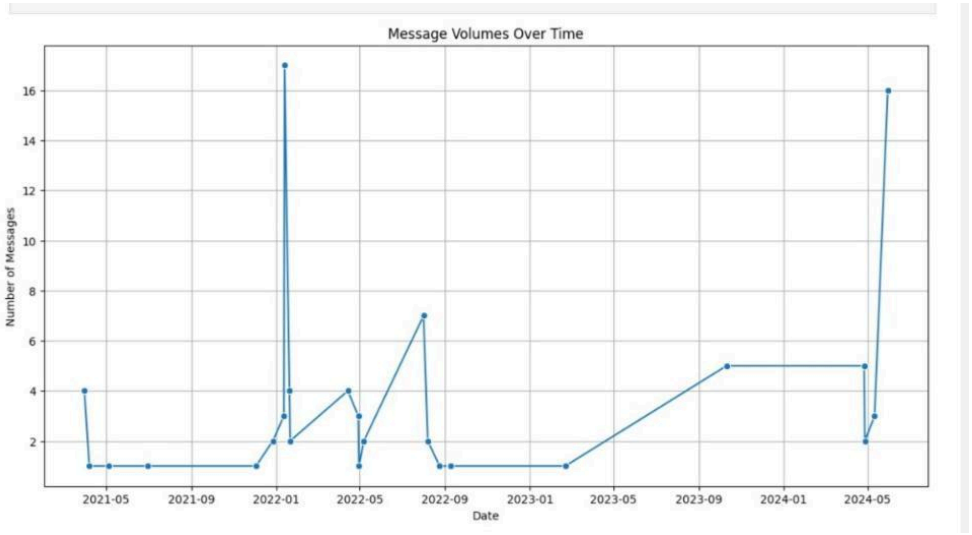
SCREENSHOTS

7. SCREENSHOTS

VISUALIZING SENTIMENTS TRENDS






Message Volumes Over Time
Number of Messages
Date
2021-05
2021-09
2022-01
2022-05
2022-09
2023-01
2023-05
2023-09
2024-01
2024-05

CHAPTER-8
FUTURE ENHANCEMENTS

8. FUTURE ENHANCEMENTS

1. Contextual Understanding:
    - Enhance models to better understand context, including long-term dependencies and nuanced conversational cues.
    - Incorporate memory mechanisms to maintain context across multiple turns.
2. Emotion Recognition:
    - Develop models that can recognize emotions expressed in text.
    - Consider sentiment analysis and affective computing to improve chatbot responses.
3. Techniques:
    - Address biases present in training data by fine-tuning models on more diverse and representative datasets.
    - Implement fairness-aware techniques to reduce bias in responses.
4. Multilingual Support:
    - Extend chat analysis to handle multiple languages seamlessly.
    - Explore zero-shot or few-shot learning for low-resource languages.
5. Personalization:
    - Customize responses based on user preferences, history, and personality.
    - Use reinforcement learning to adapt to individual users.
6. Explainability:
    - Develop methods to explain model predictions and decisions.
    - Enable users to understand why a chatbot responds in a certain way.

CHAPTER-9

CONCLUSION

9. CONCLUSION

Certainly! When analyzing text-based chat data, drawing meaningful conclusions is essential. Here are some conclusions to be noted:

Text-based chat analysis is a powerful tool that leverages natural language processing (NLP), machine learning, and data analytics to derive meaningful insights from chat data. This field has vast applications across various industries, from customer support to social media monitoring, healthcare, education, and market research. Here are the key conclusions drawn from the current state and potential of text-based chat analysis:

1. Enhanced Customer Experience

Text-based chat analysis significantly enhances customer experience by enabling businesses to understand customer sentiment and feedback in real-time. Automated sentiment analysis helps in identifying and addressing negative experiences promptly, leading to improved customer satisfaction and loyalty. Chatbots powered by NLP can provide instant support, reducing response times and enhancing service quality.

2. Improved Decision Making

Organizations can make more informed decisions by analyzing chat data to uncover trends, preferences, and emerging issues. Topic modeling helps in identifying common themes and concerns among users, allowing businesses to prioritize and address critical areas. Behavior analysis provides insights into user engagement and interaction patterns, guiding strategy and operational improvements.

3. Real-Time Monitoring and Alerts

Real-time chat analysis enables proactive monitoring and alerting of potential issues such as customer dissatisfaction, spam, or abusive content. This capability is crucial for maintaining a positive user environment, especially on social media platforms and customer service channels. Real-time insights help in mitigating risks and taking corrective actions swiftly.

4. Personalization and User Profiling

By analyzing chat interactions, businesses can create detailed user profiles and offer personalized experiences. Understanding individual user preferences and behaviors allows for targeted marketing, customized support, and tailored content delivery. This personalization enhances user engagement and drives better outcomes in customer retention and conversion rates.

5. Scalable and Efficient Data Processing

Advancements in big data technologies and cloud computing have made it possible to process and analyze large volumes of chat data efficiently. Tools like Apache Kafka, Spark, and cloud storage solutions enable scalable data ingestion, processing, and storage. This scalability is essential for handling the growing amount of chat data generated across various platforms.

6. Ethical Considerations and Data Privacy

While chat analysis offers significant benefits, it also raises ethical considerations regarding user privacy and data security. It is crucial for

organizations to implement robust data protection measures and adhere to ethical standards. Transparent data usage policies and compliance with regulations like GDPR are essential to maintain user trust and prevent misuse of personal information.

7. Integration with Multimodal Data

Future advancements in chat analysis will likely involve integrating text data with other data types, such as images, videos, and audio. This multimodal analysis will provide a more comprehensive understanding of user interactions and context. Combining text analysis with other modalities can lead to richer insights and more effective solutions.

8. Advances in NLP and Machine Learning

The field of text-based chat analysis continues to evolve with advancements in NLP and machine learning. State-of-the-art models like transformers (e.g., BERT, GPT) offer superior performance in understanding and generating human-like text. Continued research and development in these areas will lead to more accurate, context-aware, and robust analysis capabilities.

 # Final Thoughts

Text-based chat analysis is a transformative technology that enables organizations to unlock valuable insights from conversational data. By leveraging advanced analytics, businesses can enhance customer experiences, improve decision-making, and drive operational efficiencies. As the field progresses, ethical considerations and data privacy will remain paramount, ensuring that the benefits of chat analysis are realized responsibly and sustainably.

10. REFERENCE

Certainly! Here are some references related to text-based chat analysis:

1. **Noticing and Text-Based Chat:**
   - **A study by Chun Lai and Yong Zhao from Michigan State University examined the capacity of text-based online chat to promote learners' noticing of their problematic language productions and interactional feedback from interlocutors. It found that text-based chat promotes noticing more than face-to-face conversations, especially in terms of learners' noticing of their own linguistic mistakes[1].**
2. **Data Analytics on Chat Sessions:**
   - **To analyze chat data, start by identifying central themes through buzzword combinations and categories. Tag cloud formats provide simple visualizations for identifying prominently mentioned terms within content[2].**
3. **Automated Chat Transcript Analysis Using Topic Modeling:**
   - **Researchers used Latent Dirichlet Allocation (LDA) topic modeling to automatically extract topics from chat transcripts generated over five years from a large university library[3].**
4. **Machine Learning and Natural Language Processing for Library Chat Reference Transcripts:**
   - **This study analyzed one academic library's chat transcripts over eight years using machine learning and natural language processing methods[4].**
5. **WhatsApp Chat Analyser:**
   - **WhatsApp Chat Analyser is a web-based service that can analyze WhatsApp chat communication by receiving chat records. It retrieves a text-based version of conversation histories and stores chat histories for analysis[5].**