Abstract

The most used and efficient method of communication in recent times is an application called WhatsApp. WhatsApp chats consist of various kinds of conversations held among groups of people. This chat consists of various topics. This information can provide lots of data for the latest technologies such as machine learning. The most important thing for a machine learning 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 modules such as pandas, matplotlib, seaborn and sentiment analysis which are used to create data frames and plot different graphs, which then it is displayed in the flutter application which is efficient and less resources consuming algorithm, therefor it can be easily applied to largest dataset.

Introduction

- ## Domain Description:

The domain of the WhatsApp Chat Analyzer project encompasses the realm of digital communication analysis within the context of the popular instant messaging platform, WhatsApp. As one of the most widely used messaging applications globally, WhatsApp facilitates millions of conversations daily across diverse demographics and purposes.

Within this domain, the analyzer focuses on extracting meaningful insights and patterns from WhatsApp chat data. This includes understanding user behaviour, communication dynamics, sentiment trends, and content preferences within individual or group conversations.

The WhatsApp Chat Analyzer project delves into the nuances of modern communication, exploring how text-based interactions shape relationships, influence decision-making processes, and reflect societal trends. It caters to users seeking to gain deeper insights into their messaging habits, researchers investigating social dynamics, and businesses analyzing customer feedback and communication patterns.

Overall, the domain of the WhatsApp Chat Analyzer project intersects technology, linguistics, psychology, and sociology, offering a multifaceted approach to understanding and analyzing digital communication within the WhatsApp platform.

- ## Motivation

The motivation behind the creation of the WhatsApp Chat Analyzer project stems from the increasing significance of digital communication in modern society. With the proliferation of instant messaging platforms like WhatsApp, communication has become more accessible, immediate, and pervasive than ever before. However, amidst the vast volume of messages exchanged daily, there lies a treasure trove of valuable insights waiting to be uncovered.

This project aims to address the need for tools that enable individuals, researchers, and organizations to make sense of the wealth of data generated through WhatsApp conversations. By analyzing chat logs, users can gain deeper insights into their communication patterns, social dynamics, and emotional trends. Researchers can utilize the analyzer to study language usage, sentiment fluctuations, and cultural phenomena within digital communities. Additionally, businesses can leverage the tool to analyze customer feedback, identify market trends, and enhance their communication strategies.

In essence, the motivation behind the WhatsApp Chat Analyzer project lies in its potential to unlock the hidden insights within WhatsApp conversations, enabling users to gain a deeper understanding of their digital interactions and harness the power of data-driven communication analysis.

- ## Scope of Work

The scope of the WhatsApp Chat Analyzer project encompasses the development of a comprehensive tool for analyzing chat data generated on the WhatsApp messaging platform. The project aims to provide users with a range of functionalities to extract insights, trends, and meaningful information from their WhatsApp conversations.

The "WhatsApp Chat Analyzer" project can be highly beneficial for colleges in several ways. For Example, Colleges can utilize the analyzer to research student communication patterns, social dynamics within student groups, and sentiment analysis of student feedback. This data can provide valuable insights into student engagement, satisfaction levels, and areas for improvement.

- ## Problem Statement

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 participant a user responds to the most. We propose to employ dataset manipulation techniques to have a better understanding of the WhatsApp chat present on our phones.

Methodology

What is a WhatsApp chat Analyser?

- A WhatsApp Chat Analyzer is a software tool designed to analyze and extract insights from chat data generated on the WhatsApp messaging platform.
- The WhatsApp Chat Analyser is a Python-based application built using Streamlit, Seaborn, Matplotlib, and the Pandas library. It allows users to upload WhatsApp chat logs for analysis and provides various statistics and visualizations.
- This tool is useful for anyone looking to gain a deeper understanding of their WhatsApp conversations, whether for personal reflection, academic research, or social analysis.

Advantages of WhatsApp Chat Analyser

1. Runs on all devices.
2. Shows based on WhatsApp chat file.
3. Shows different visualisations.

- Total Messages
- Total words
- Media shared
- Link shared
- Monthly timeline
- Most busy day
- Most busy month
- Weekly Activity
- Most busy users
- Most used words
- Emoji analysis

Features of Whatsapp Chat Analyser

It tells you:

➢ User friendly.
➢ Time saving.
➢ It runs on any device.

- ➢ Analyse any WhatsApp imported chat file.
- ➢ Easy to use.

Working

Steps to Export Chat:

- ❖ Open WhatsApp chat for a group ->click on the menu ->click on more- ->select export chat->choose without media.

Working of WhatsApp chat analysis.

1. Intially open the WhatsApp chat analyser web page.

2. Upload the exported chat file.

3. Click on Show Analysis.

4. Preprocessing of data is done by trained model.

5. Select overall or single-person analysis.

6. Trained model shows analysis it includes top statistics, word cloud, activity map, monthly timeline, daily timeline, and emoji analysis.

Data Analysis

Data analysis is a process for acquiring raw data and transforming it into useful information for decision-making by users. This project provides a basic statistical analysis of WhatsApp chat. Following is the analysis made :

- To find total messages, total words, total media and links shared in the WhatsApp chat.

- To find the most active people in the group

- To find the most used emojis in the group.

- To find the busiest day and least busy in a month.

- To find the most frequently and commonly used words in the group.

- To find the frequency of chat every day and month.

System Modules

(a) **Install and import dependencies**: In this step Streamlit, matplotlib, pandas, collections, seaborn, emoji, Wordcloud, URL extract, and re are installed and imported.

(b) **Pre-processing:** In this step pre-processing of the data is done. Here the data is formatted and separated in the form of date, time, name of the user and message of the use.

(c) **Export chat document from WhatsApp and Upload:** Here the document is exported from WhatsApp. Steps to export chat → Open individual or Group chat → Tap Options → More → Export Chat → Choose export without media → Document is set. Upload the chat file and click on show analysis.

(d) **Analyze the data:** Here Analysis made:

1. Top Statistics: These involve total messages, total words, media shared, and links shared.

2. Monthly Timeline: The frequency of chat in every month.

3. Daily Timeline: The frequency of chat in a day.

4. Activity Map: Shows the busiest day and least busy day similarly to the month

5. Weekly Activity map.

6. Wordcloud: Most commonly and frequently used word.

7. Most Busy Users: Mostly active people. 8. Emoji analysis: Most commonly and frequently used emojis.

Testing

Sno	Test Case	Description	Expected Result	Test Result
1	Top Statistics	These involve total messages, total words, total media and links shared.	Statistics of total messages, words, links, and media shared in the group	PASS
2	Monthly Timeline	The frequency of chat in every month	Graph of monthly timeline	PASS
3	Activity Map	Shows the busiest day and least busy day in a month	Bar graph of most busy day and most busy month	PASS
4	Word Cloud	Most commonly and frequently used words	Word cloud of most used words	PASS

Feasibility Study of WhatsApp Chat Analyser:

1. Introduction:

The WhatsApp Chat Analyzer is designed to provide detailed insights into WhatsApp chat logs through statistical analysis and visualization. This tool is particularly useful for users who want to understand their communication patterns, identify the most active participants, and analyze the content of their messages.

2. Objectives:

● Analyze WhatsApp chat logs to provide comprehensive statistics.

● Visualize data through charts, graphs, and word clouds.

● Identify the most active users, common words, and emojis used in chats.

● Offer an easy-to-use interface for uploading and analyzing chat logs.

3. Technical Feasibility:

● Technology Stack: The project leverages Python for data processing and analysis, Streamlit for the web interface, and libraries such as Pandas, Seaborn, Matplotlib, WordCloud, URLEXtract, and emoji for specialized tasks.

● Development Tools: VS Code.

● Operating Systems: Compatible with Windows, Linux, and macOS.

● Skills Required: Proficiency in Python, data analysis, web development with Streamlit, and familiarity with the chosen libraries.

4. Operational Feasibility:

● User Interface: Streamlit provides a user-friendly interface that requires minimal technical knowledge to operate.

● Workflow: Users can easily upload their chat logs, and the tool processes the data to display various statistics and visualizations.

● **Maintenance:** Regular updates and bug fixes can be managed efficiently due to the modular structure of the code.

5. Economic Feasibility:

● Cost of Development: Primarily involves the time and effort of the developer, with minimal financial investment for tools and software, as most are open-source.

● Benefits: Provides significant value to users by offering insights into their communication patterns, which can be used for personal reflection or academic research.

6. Legal Feasibility:

● Data Privacy: Ensures user data is handled securely and privately, with no data stored on external servers. All analysis is performed locally on the user's machine.

● Compliance: Adheres to data protection regulations, ensuring user consent for data analysis.

7. Market Feasibility:

● Target Audience: Individuals, researchers, and organizations interested in communication patterns and social behaviour analysis.

● Competition: Few tools offer the same level of detailed analysis and

user-friendly interface, giving the WhatsApp Chat Analyzer a competitive edge.

8. Environmental Feasibility:

● Resource Usage: The application is lightweight and does not require extensive computational resources, making it environmentally sustainable.

● Energy Consumption: Efficient coding practices ensure minimal energy consumption during data processing.

9. Risk Analysis:

● Technical Risks: Potential bugs or compatibility issues with different operating

systems. Mitigation involves thorough testing and updates.

● Operational Risks: User data privacy concerns. Mitigation involves ensuring

robust data handling practices.

● Market Risks: Adoption rates may be affected by competition or lack of awareness. Mitigation involves effective marketing and continuous improvement of features.

Design Description

DFD is the abbreviation for Data Flow Diagram. DFD represents the flow of data in a system or a process. It also gives insight into the inputs and outputs of each entity and the process itself. DFD does not have a control flow and no loops or decision rules are present. A flowchart can explain specific operations depending on the type of data. Data Flow Diagrams can be represented in several ways. The DFD belongs to structured-analysis modelling tools. Data Flow diagrams are very popular because they help us to visualise the major steps and data involved in software-system processes.

Levels of DFD:

DFD uses hierarchy to maintain transparency thus multi-level DFDs can be created.

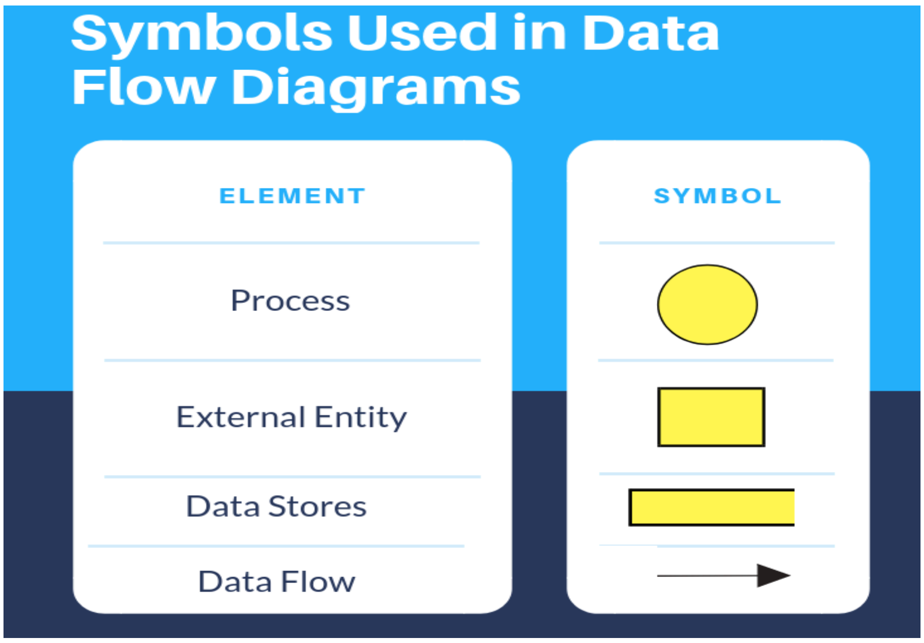
Levels of DFD are as follows:

➢ 0-level DFD

➢ 1-level DFD

➢ 2-level DFD

For drawing DFD there are 4 types of symbols used.

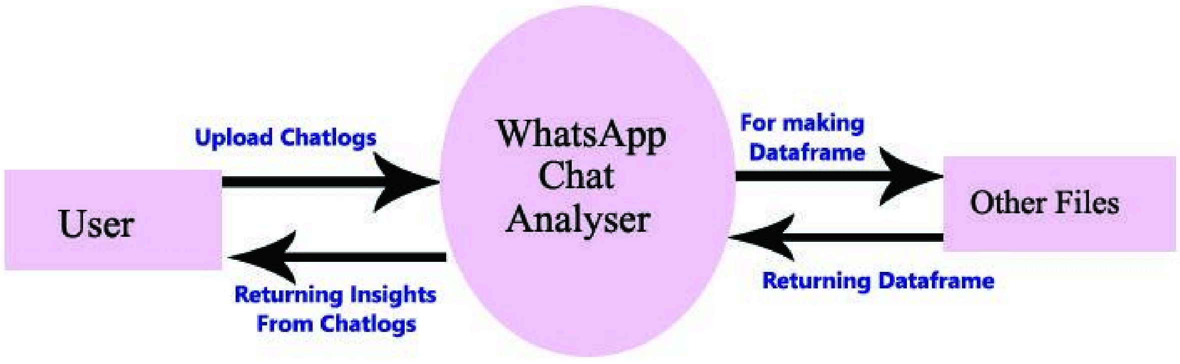


Symbols Used in Data Flow Diagrams

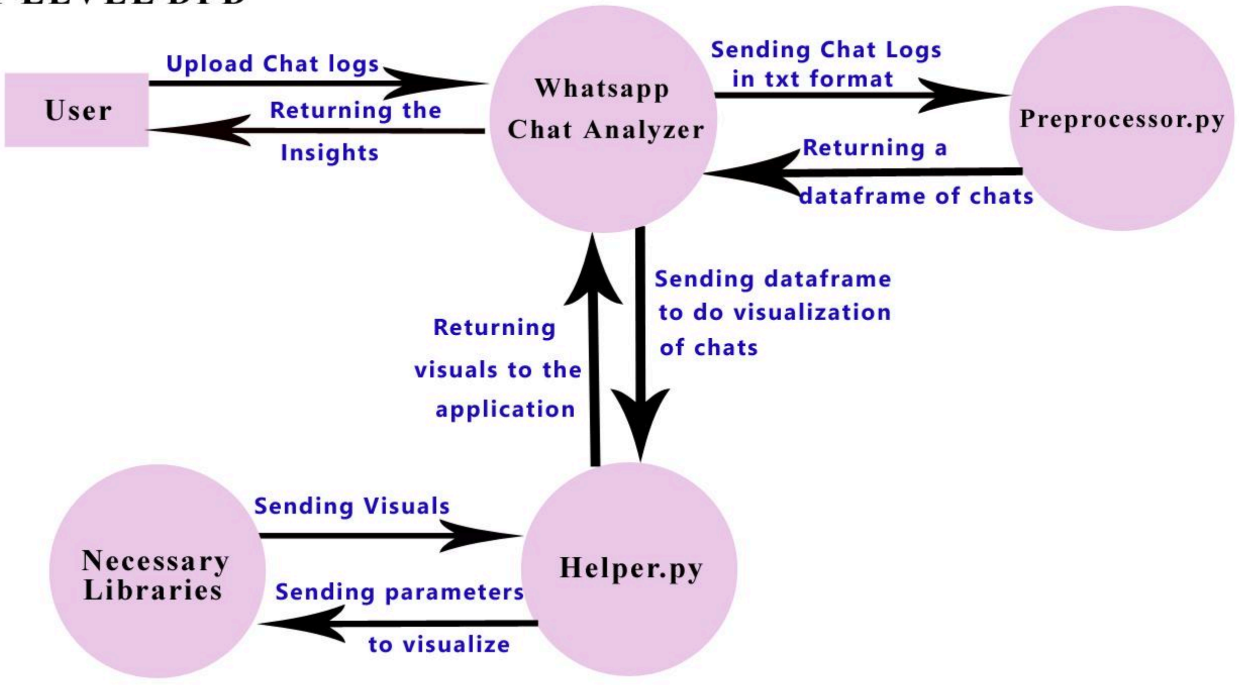
ELEMENT	SYMBOL
Process	●
External Entity	■
Data Stores	▭
Data Flow	→

For building this project, there is used a different type of DFD. Below I demonstrate all one by one.

0-Level DFD



1-LEVEL DFD


User
Upload Chat logs
Returning the Insights
Whatsapp Chat Analyzer
Sending Chat Logs in txt format
Returning a dataframe of chats
Preprocessor.py
Sending dataframe to do visualization of chats
Returning visuals to the application
Helper.py
Sending Visuals
Sending parameters to visualize
Necessary Libraries

Model to be used:

For the WhatsApp Chat Analyzer project, the most suitable Software Development Life Cycle (SDLC) model would likely be the Iterative Model. This model is ideal for projects where requirements are not fully understood from the beginning and where iterative refinement of the system is necessary. Here's why the Iterative Model is a good fit:

- **1. Incremental Development:** The project can be developed and delivered in small, incremental steps, allowing for parts of the system to be developed and refined before moving on to the next phase.
- **2. Flexibility and Adaptability:** As you gather user feedback or encounter new requirements, you can adapt and make changes in subsequent iterations without needing to overhaul the entire project.
- **3. Risk Management:** Early iterations can focus on the most critical and high-risk components, ensuring that any potential issues are addressed early in the development process.
- **4. Continuous Improvement:** Each iteration builds on the previous one, allowing for continuous refinement and improvement based on user feedback and testing results.

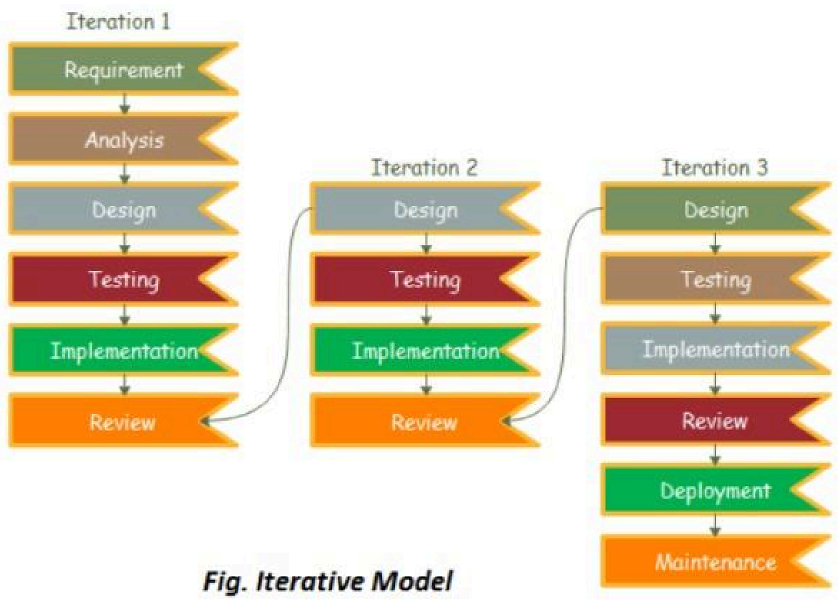


Fig. Iterative Model

- **Software Requirement:**

1. Visual Studio Code
2. Jupyter notebook
3. Google Chrome Browser

- ## Language to be used:

  1. **Python:**

     Python is a versatile programming language that offers an extensive range of libraries catering to various functionalities required for different projects.

  2. **Pandas:**

This open-source Python library is widely utilized in the fields of Data Science and machine learning. It serves as a valuable resource for data analysis, offering powerful tools for data manipulation.

  3. **Numpy:**

Derived from the term "Numeric Python," Numpy is a Python library renowned for its data analysis capabilities. Moreover, Numpy is equipped with powerful multi-dimensional array objects that enable efficient processing of arrays.

  4. **Matplotlib:**

Matplotlib, a user-friendly and powerful visualization library for Python, proves to be an excellent choice for creating stunning visual representations. Matplotlib offers a diverse range of plot types, including pie charts, line charts, bar charts, scatter plots, histograms, and more.

  5. **Seaborn:**

     Seaborn, a popular Python library, specializes in statistical plotting, enhancing the visual appeal of plots with its exquisite colorpalettes and predefined styles. In this project, Seaborn is utilized to create visually appealing heatmap visualizations. These heatmaps represent 24 hours across 7 days, utilizing varying colours to denote the intensity of messages from highest to lowest.

  6. **Streamlit:**

     In this project, the mentioned library is employed to generate visually appealing web elements and objects. These components are utilized to represent the analysis of WhatsApp chats using a variety of charts and visualizations on the Streamlit platform.

Algorithm Description

- **whatsapp_chat_analyser.py**

Step 1. Import necessary libraries:

- `helper` for custom functions and data analysis.

- `seaborn` for data visualisation.

- `streamlit` for creating the web app interface.

- `preprocessor` for preprocessing WhatsApp chat data.

- `matplotlib. pyplot` for basic plotting functionalities.

Step 2. Set up the Streamlit page configuration:

- Set the page title, icon, and layout.

Step 3. Create the sidebar:

- Add an image to the sidebar using Markdown for branding.

- Set the title for the WhatsApp Chat Analyser app.

Step 4. Enable file upload:

- Allow users to upload their WhatsApp chat data files.

Step 5. Data processing and analysis:

- If a file is uploaded, preprocess the data using the `preprocessor` library.

- Fetch unique users from the data and prepare options for analysis selection.

Step 6. Show analysis based on user selection:

- Display statistics such as total messages, words, media shared, and links shared.

- Visualize monthly and daily timelines of message distribution using line plots.

- Show activity maps for the most texted day and month.

- Display a weekly activity heatmap using Seaborn's heatmap.

- Identify the most busy user in the group and display relevant data.

- Create a word cloud and visualize the most common words.

- Analyze emoji usage and display emoji statistics using a pie chart.

Step 7. Display the results:

- Render the analysis results and visualizations in the Streamlit app interface.

- ## **helper.py**

Step 1. Import necessary libraries:

- `emoji` for handling emojis in messages.

- `pandas` for data manipulation and analysis.

- `Counter` from `collections` for counting occurrences of elements.

- `WordCloud` from `wordcloud` for creating word clouds.

- `URLExtract` from `urlextract` for extracting URLs from messages.

Step 2. Define functions for data analysis and visualization:

- `fetch_stats`: Fetch statistics such as number of messages, words, media messages, and links shared.

- `fetch_most_busy_user`: Identify and fetch the most active users in terms of message count.

- `create_wordcloud`: Generate a word cloud based on messages, excluding stop words and emojis.

- `most_common_word`: Determine the most common words used, excluding stop words and emojis.

- `emoji_helper`: Analyze and count emoji usage in messages.

- `monthly_timeline`: Generate a monthly timeline of message distribution.

- `daily_timeline`: Generate a daily timeline of message distribution.

-`week_activity_map`: Calculate the activity distribution by day of the week.

- `month_activity_map`: Calculate the activity distribution by month.

- `activity_heatmap`: Create a heatmap of activity based on day and period (e.g., morning, afternoon).

Step 3. Data preprocessing and filtering:

- Remove system notifications and media messages from the data.

- Exclude stop words and emojis from word-based analyses.

Step 4. Data aggregation and visualization:

- Aggregate data for analysis based on selected users or overall statistics.

- Generate visualizations such as word clouds, bar charts, and heatmaps using Matplotlib and Seaborn.

Step 5. Output analysis results:

- Display statistics, timelines, most common words, emoji usage, and activity maps in the Streamlit app interface.

- **preprocessor.py**

Step 1. Import necessary libraries:

- `re` for regular expressions.

- `pandas` for data manipulation.

Step 2. Define a function named `preprocess` that takes `data` as input:

- Split the input `data` into lines and extract messages using a regular Expression pattern.

- Extract date and time information from each message based on a different date format.

- Create a pandas DataFrame with columns for user messages and message dates.

- Convert the message dates to datetime format using the `pd.to_datetime` function with format 'mixed'.

- Extract users and messages from the user_message column and update the DataFrame accordingly.

- Extract additional date-related information such as year, month, day, day of the week, hour, minute, and date timeline.

- Create a period column based on the hour to categorize messages into time periods.

- Drop any rows with NaN values in the DataFrame.

Step 3. Return the preprocessed DataFrame `df_new`.

IMPLEMENTATION

whatsapp_chat_analyser.py

```
import streamlit as st
import preprocessor as ppr
import helper
import json
import matplotlib.pyplot as plt
import seaborn as sn
from streamlit_lottie import st_lottie

st.set_page_config(
   page_title="WhatsApp Chat Analyser",

page_icon="https://static.vecteezy.com/system/resources/previews/023/98
6/631/non_2x/whatsapp-logo-whatsapp-logo-transparent-whatsapp-icon-tr
ansparent-free-free-png.png",
   layout = "wide"
)

with open("Animation - 1711349086623.json",'r') as file:
   animation = json.load(file)
st.lottie(animation,width=550,height=550)
st.sidebar.title(":violet[Whatsapp Chat Analyser]")
uploaded_file = st.sidebar.file_uploader(":green[Upload your file here]
",type=['txt'])
if uploaded_file is not None:
   # Converting the uploaded file whic is a byte stream into string
   bytes_data = uploaded_file.getvalue()
   # Converting data into string
   data = bytes_data.decode("utf-8")
   df = ppr.preprocess(data)

   #fetch unique user
   user_list = df['user'].unique().tolist()
   if 'group_notification' in user_list:
      user_list.remove('group_notification')
      user_list.sort()
      user_list.insert(0,"Overall")
```

```
        selected_user = st.sidebar.selectbox(":blue[Show Analysis] ",user_list)

    if st.sidebar.button("Show Analysis"):
        st.snow()
        num_msgs,words,num_media_msg,links =
helper.fetch_stats(selected_user,df)
        st.markdown("<h1 style='text-align:center';>Whatsapp
Statistics</h1>",unsafe_allow_html=True)
        st.markdown("---")
        col1,col2,col3,col4,col5 = st.columns(5)
        if selected_user == "Overall":
            with col1:
                st.header("Total Persons")
                st.title(len(user_list))
        with col2:
            st.header("Total Message ")
            st.title(num_msgs)
        with col3:
            st.header("Total Words")
            st.title(words)
        with col4:
            st.header("Total Media")
            st.title(num_media_msg)
        with col5:
            st.header("Total Links")
            st.title(links)

        # finding the most busy users in the group
        if selected_user == "Overall":
            st.markdown("<h1 style='text-align:center';>Most Busy
Users</h1>",unsafe_allow_html=True)
            x,new_df = helper.most_busy_users(df)
            fig,ax = plt.subplots()
            col1,col2 = st.columns(2)
            with col1:
                ax.bar(x.index,x.values,color="red")
                plt.xticks(rotation="vertical")
                st.pyplot(fig)
            with col2:
                st.dataframe(new_df)
         # Creating WordCloud
        st.markdown("---")
        st.markdown("<h1 style='text-align:center';>Word
Cloud</h1>",unsafe_allow_html=True)
        st.markdown("---")
        df_wc = helper.create_wordCloud(selected_user,df)
        fig,ax = plt.subplots()
        ax.imshow(df_wc)
        st.pyplot(fig)


        # most common words
        st.markdown("---")
        st.markdown("<h1 style='text-align:center';>Most Common
Words</h1>",unsafe_allow_html=True)
        st.markdown("---")
        most_common_df = helper.most_commonWords(selected_user,df)
        fig,ax = plt.subplots()
        ax.barh(most_common_df[0],most_common_df[1])
        plt.xticks(rotation='vertical')
        st.pyplot(fig)


        # Emoji analysis
        st.markdown("---")
        st.markdown("<h1 style='text-align:center';>Emoji
Analysis</h1>",unsafe_allow_html=True)
        st.markdown("---")
        emoji_df = helper.emoji_helper(selected_user, df)
        if len(emoji_df) != 0:
            col1, col2 = st.columns(2)
            with col1:
                st.dataframe(emoji_df.head())

            with col2:
                # The 'Segoe UI Emoji' font family includes emojis and other
Unicode symbols, making it suitable for displaying emoji characters in
plots.
                plt.rcParams['font.family'] = 'Segoe UI Emoji'
                fig, ax = plt.subplots()
```

```
            ax.pie(emoji_df[1].head(), labels = emoji_df[0].head(),
autopct="%0.1f%%")
            st.pyplot(fig)

      else:
         st.write(f"{selected_user} have not send any emoji right now...")


      #Monthly timeline
      st.markdown("---")
      st.markdown("<h1 style='text-align:center';>Monthly Timeline
Analysis</h1>",unsafe_allow_html=True)
      st.markdown("---")
      timeline = helper.monthly_timeline(selected_user,df)
      fig,ax = plt.subplots()
      ax.plot(timeline['time'],timeline['message'],color="red")
      plt.xticks(rotation = 'vertical')
      st.pyplot(fig)


      #Daily Timeline
      st.markdown("---")
      st.markdown("<h1 style='text-align:center';>Daily
Timeline</h1>",unsafe_allow_html = True)
      st.markdown("---")
      daily_timeline = helper.daily_timeline(selected_user,df)
      fig,ax = plt.subplots()

ax.plot(daily_timeline['daily_timeline'],daily_timeline['message'],color="mag
enta")
      plt.xticks(rotation = 'vertical')
      st.pyplot(fig)


      # Activity map
      st.markdown("---")
      st.markdown("<h1 style='text-align:center';>Activity
Map</h1>",unsafe_allow_html = True)
      st.markdown("---")
      col1,col2 = st.columns(2)
```

```
    with col1:
        st.subheader("**Most Busy Day**")
        busy_day = helper.week_activity_map(selected_user,df)
        fig,ax = plt.subplots()
        ax.bar(busy_day.index,busy_day.values,color="green")
        plt.xticks(rotation='vertical')
        st.pyplot(fig)
    with col2:
        st.subheader("**Most Busy Month**")
        busy_month = helper.month_activity_map(selected_user,df)
        fig,ax = plt.subplots()
        ax.bar(busy_month.index,busy_month.values,color="black")
        plt.xticks(rotation='vertical')
        st.pyplot(fig)


    # User Activity Heatmap
    st.markdown("---")
    st.markdown("<h1 style='text-align:center';>Weekly Activity
Heatmap</h1>",unsafe_allow_html = True)
    st.markdown("---")
    user_heatmap = helper.activity_heatmap(selected_user,df)
    fig,ax = plt.subplots(figsize=(20,8))
    ax = sn.heatmap(user_heatmap,annot=True,
cmap=sn.cubehelix_palette(as_cmap=True))
    st.pyplot(fig)
```

preprocessor.py

```
import re
import pandas as pd

# It will Preprocess the data and return the dataframe
def preprocess(data):
    # Message Extracting
    pattern = r'\s-\s'
    message = [re.split(pattern, line)[1] if re.split(pattern, line)[1:] else line for
line in data.split('\n')]

    # Date and time Extracting

    if 'AM' in data or 'PM' in data or 'am' in data or 'pm' in data:
        date_pattern =
r'^(\d{1,2}/\d{1,2}/\d{2,4},\s\d{1,2}:\d{2}\s[apAP][mM]\s-)\s'
        dates = [re.findall(date_pattern, line)[0] if re.findall(date_pattern, line)
else '' for line in data.split('\n')]
        if dates[0] == '':
            date_pattern = r'^(\d{1,2}/\d{1,2}/\d{2,4},\s\d{1,2}:\d{2}\s-)\s'
            dates = [re.findall(date_pattern, line)[0] if re.findall(date_pattern,
line) else '' for line in data.split('\n')]

    for i in range(len(dates)):
        dates[i] = dates[i].replace(' -', '')
        dates[i] = dates[i].replace(' -', '')

    df = pd.DataFrame({'user_message': message, 'message_date': dates})

    # Mixed is used to convert the string date format into the date-time
format
    df['message_date'] = pd.to_datetime(df['message_date'], format='mixed')
    df.rename(columns={'message_date': 'date'}, inplace=True)
    #separate users and messages
    users = []
    messages = []
    for msg in df['user_message']:
        entry = msg.split(':')
        if entry[1:]: #username
```

```
            users.append(entry[0])
            messages.append(entry[1])
        else:
            users.append('group_notification')
            messages.append(entry[0])
    df['user'] = users
    df['message'] = messages
    df.drop(columns=['user_message'],inplace = True)
    df['daily_timeline'] = df['date'].dt.date
    df['year'] = df['date'].dt.year
    df['month_num'] = df['date'].dt.month
    df['month'] = df['date'].dt.month_name()
    df['day'] = df['date'].dt.day
    df['day_name'] = df['date'].dt.day_name()
    df['hour'] = df['date'].dt.hour
    df['minute'] = df['date'].dt.minute

    period = []
    for hour in df[['day_name','hour']]['hour']:
        if hour == 23:
            period.append(str(hour) + "-" + str('00'))
        elif hour == 0:
            period.append(str('00') + "-" + str(hour+1))
        else:
            period.append(str(hour) + "-" + str(hour-1))
    df['period']  = period
```
# It will show only the users in the dataframe chats will not be included
```
    df_new = df.dropna()
    return df_new
```

helper.py

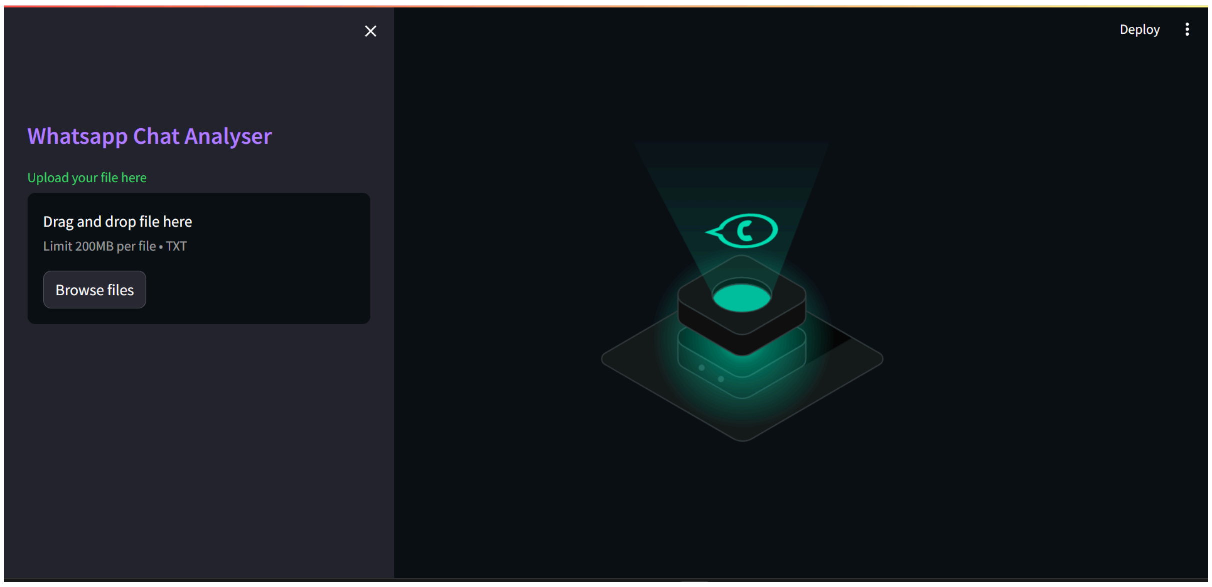
```
from urlextract import URLExtract
import re
import matplotlib.pyplot as plt
from wordcloud import WordCloud
import pandas as pd
from collections import Counter
import emoji

extract  = URLExtract()
def fetch_stats(selected_user,df):
    if selected_user != 'Overall':
        df = df[df['user'] == selected_user]

    #1. fetch number of messages
    num_msgs = df.shape[0]

    #2. fetch number of words
    words = []
    for message in df['message']:
        words.extend(message.split())

    # 3. Fetch number of media messages
        num_media_msg = df[df['message'] == ' <Media omitted>'].shape[0]

    # Count total links shared
        links = df[df['message'] == ' https'].shape[0]

    return num_msgs,len(words),num_media_msg,links

#For busy users
def most_busy_users(df):
    x = df['user'].value_counts().head()
    df =
round((df['user'].value_counts()/df.shape[0])*100,2).reset_index().rename(col
umns={'users':'name','count':'percent'})
    return x,df
```

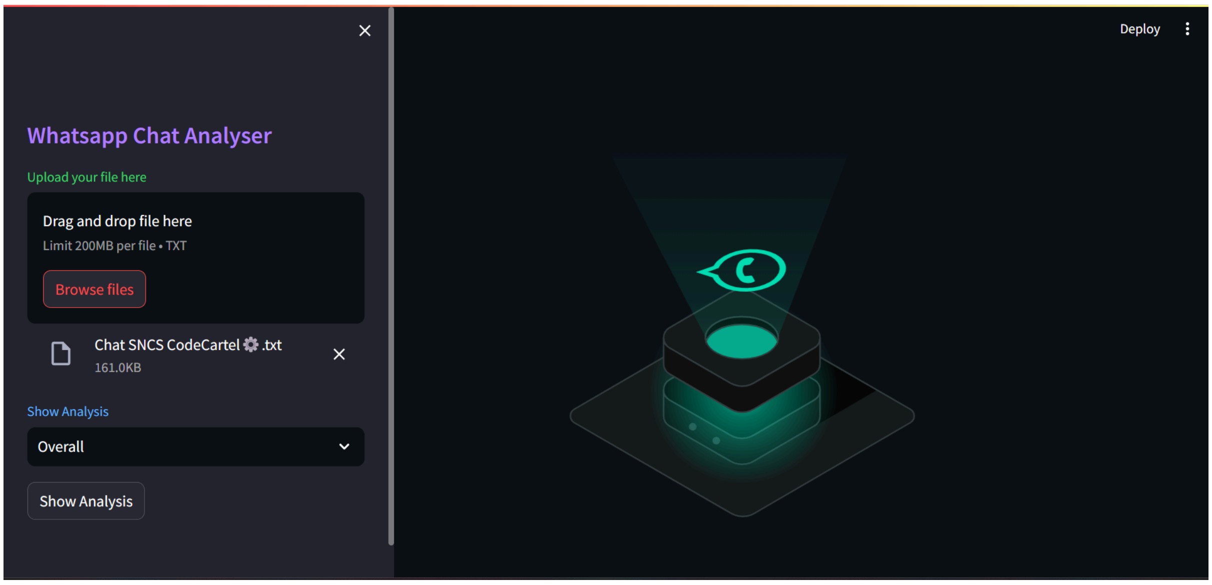
```
# For Word Cloud
def create_wordCloud(selected_user,df):
    f = open("stop_hinglish.txt",'r')
    stop_words = f.read()

    if selected_user != 'Overall':
        df = df[df['user'] == selected_user]

    temp = df[df['user'] != 'group_notification']
    temp = temp[temp['message'] != r' <Media omitted>']


# To remove all the hinglish stop word
    def remove_stopWords(message):
        y = []
        for word in message.lower().split():
            if word not in stop_words:
                y.append(word)
        return " ".join(y)

    wc =
WordCloud(width=500,height=500,min_font_size=10,background_color
='white')
    temp['message'] = temp['message'].apply(remove_stopWords)

# For generating word cloud
    df_wc = wc.generate(temp['message'].str.cat(sep=" "))
    return df_wc


# For most common words
def most_commonWords(selected_user,df):
    f = open("stop_hinglish.txt",'r')
    stop_words = f.read()

    if selected_user != 'Overall':
        df = df[df['user'] == selected_user]
    temp = df[df['user'] != 'group_notification']
    temp = temp[temp['message'] != r' <Media omitted>']
    words = []
    for message in temp['message']:
        for word in message.lower().split():
            if word.lower() not in stop_words and not emoji.is_emoji(word.lower()):
                words.append(word.lower())

    most_common_df =  pd.DataFrame(Counter(words).most_common(20))
    return most_common_df

# For Emoji
def emoji_helper(selected_user, df):
    if selected_user != 'Overall':
        df = df[df['user'] == selected_user]

    emojis=[]
    for message in df['message']:
        emojis.extend([c for c in message if emoji.is_emoji(c)])

    emoji_df =
pd.DataFrame(Counter(emojis).most_common(len(Counter(emojis))))
    return emoji_df


# For monthly  timeline
def monthly_timeline(selected_user,df):
    if selected_user != 'Overall':
        df = df[df['user'] == selected_user]
    timeline =
df.groupby(['year','month_num','month']).count()['message'].reset_index()
    time = []
    for i in range(timeline.shape[0]):
        time.append(timeline['month'][i]+ "-" + str(timeline['year'][i]))
    timeline['time'] = time

    return timeline


# For daily timeline
def daily_timeline(selected_user,df):
    if selected_user != 'Overall':
        df = df[df['user'] == selected_user]
    daily_timeline =
df.groupby('daily_timeline').count()['message'].reset_index()

    return daily_timeline
```

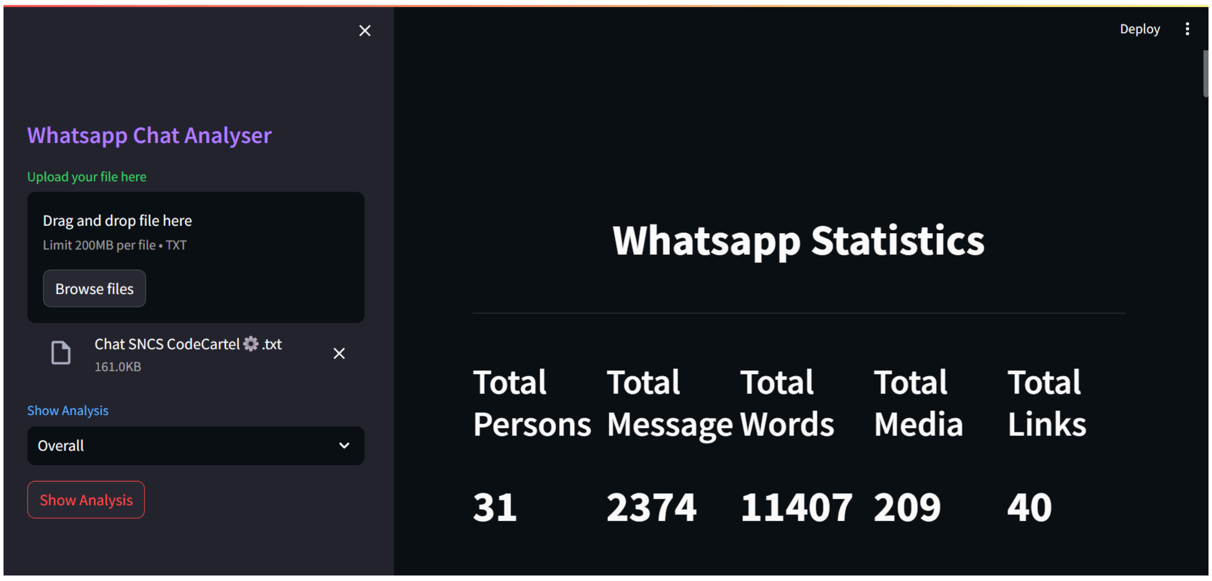
# Activity map

```
def week_activity_map(selected_user,df):
    if selected_user != 'Overall':
        df = df[df['user'] == selected_user]
    return df['day_name'].value_counts()

def month_activity_map(selected_user,df):
    if selected_user != 'Overall':
        df = df[df['user'] == selected_user]
    return df['month'].value_counts()


def activity_heatmap(selected_user,df):
    if selected_user != 'Overall':
        df = df[df['user'] == selected_user]
    user_heatmap =
df.pivot_table(index='day_name',columns='period',values='message',aggfunc
='count').fillna(0)
    return user_heatmap
```

OUTPUT

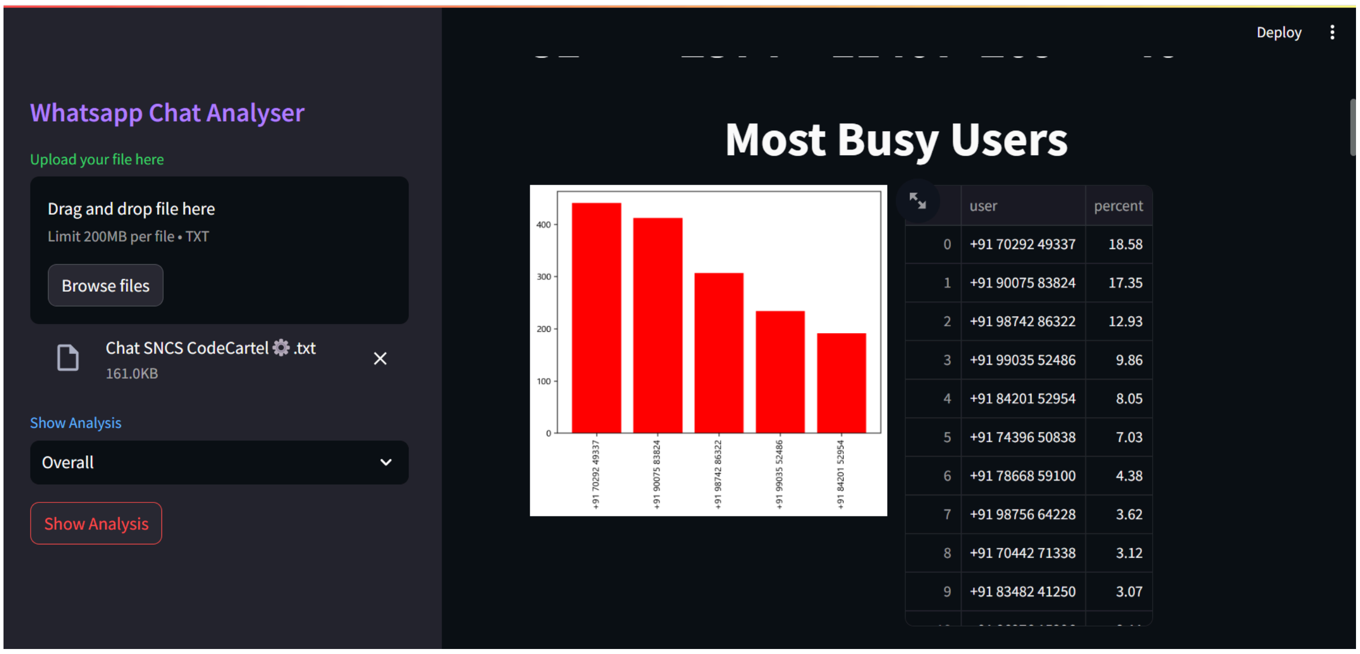






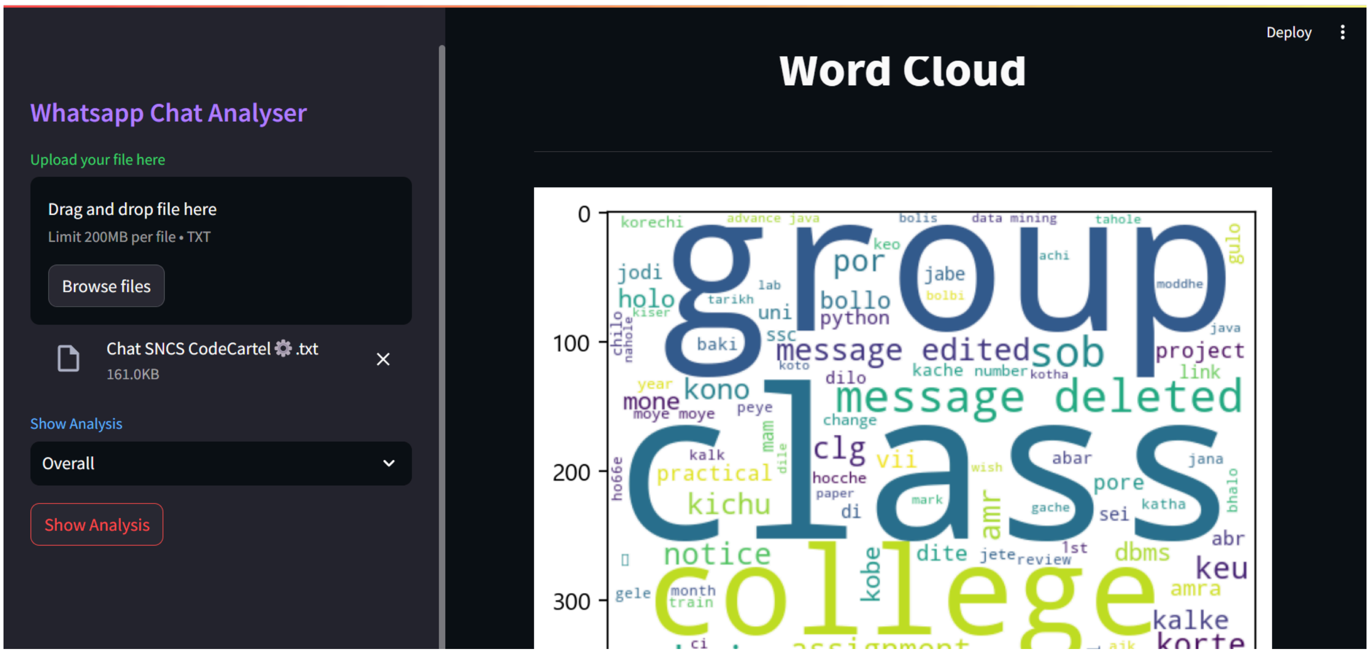
Whatsapp Statistics

Total Persons	Total Message	Total Words	Total Media	Total Links
31	2374	11407	209	40

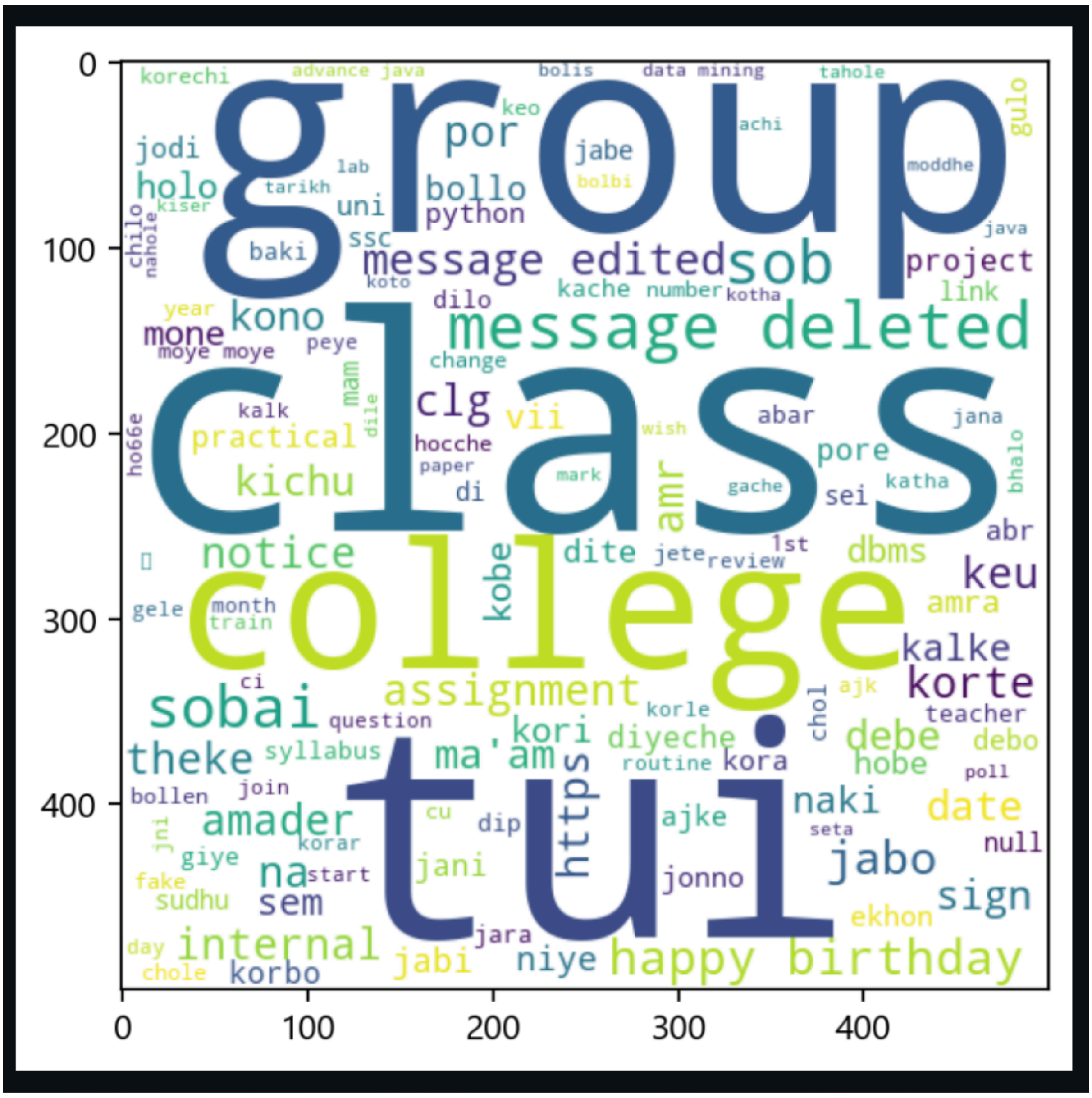


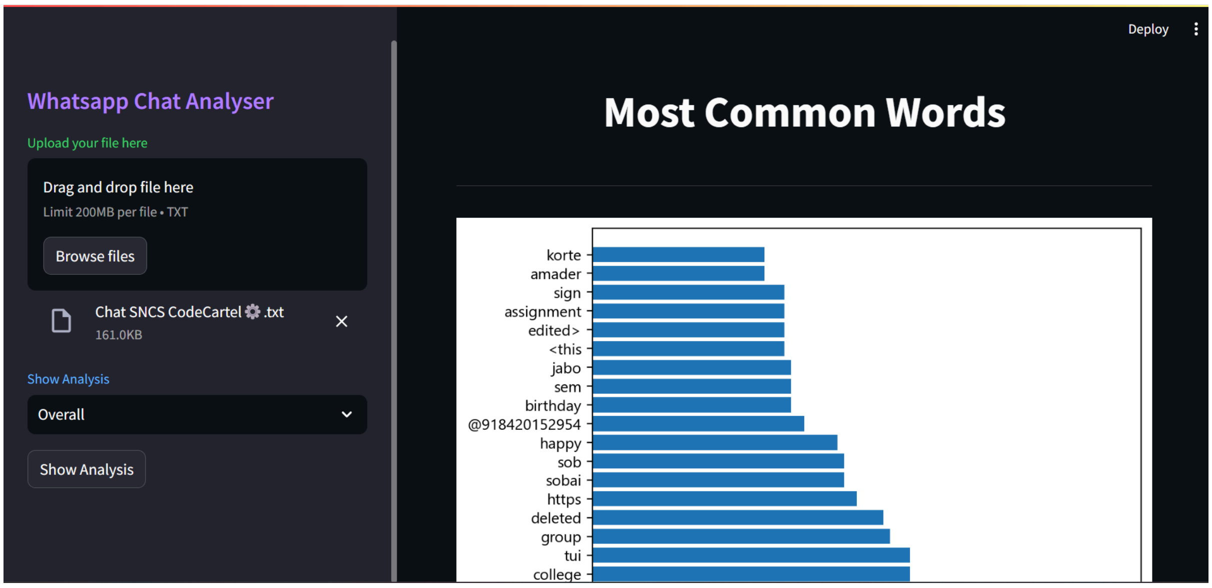
Most Busy Users

	user	percent
0	+91 70292 49337	18.58
1	+91 90075 83824	17.35
2	+91 98742 86322	12.93
3	+91 99035 52486	9.86
4	+91 84201 52954	8.05
5	+91 74396 50838	7.03
6	+91 78668 59100	4.38
7	+91 98756 64228	3.62
8	+91 70442 71338	3.12
9	+91 83482 41250	3.07



Word Cloud





Deploy

Whatsapp Chat Analyser

Upload your file here

Drag and drop file here
Limit 200MB per file • TXT

Browse files

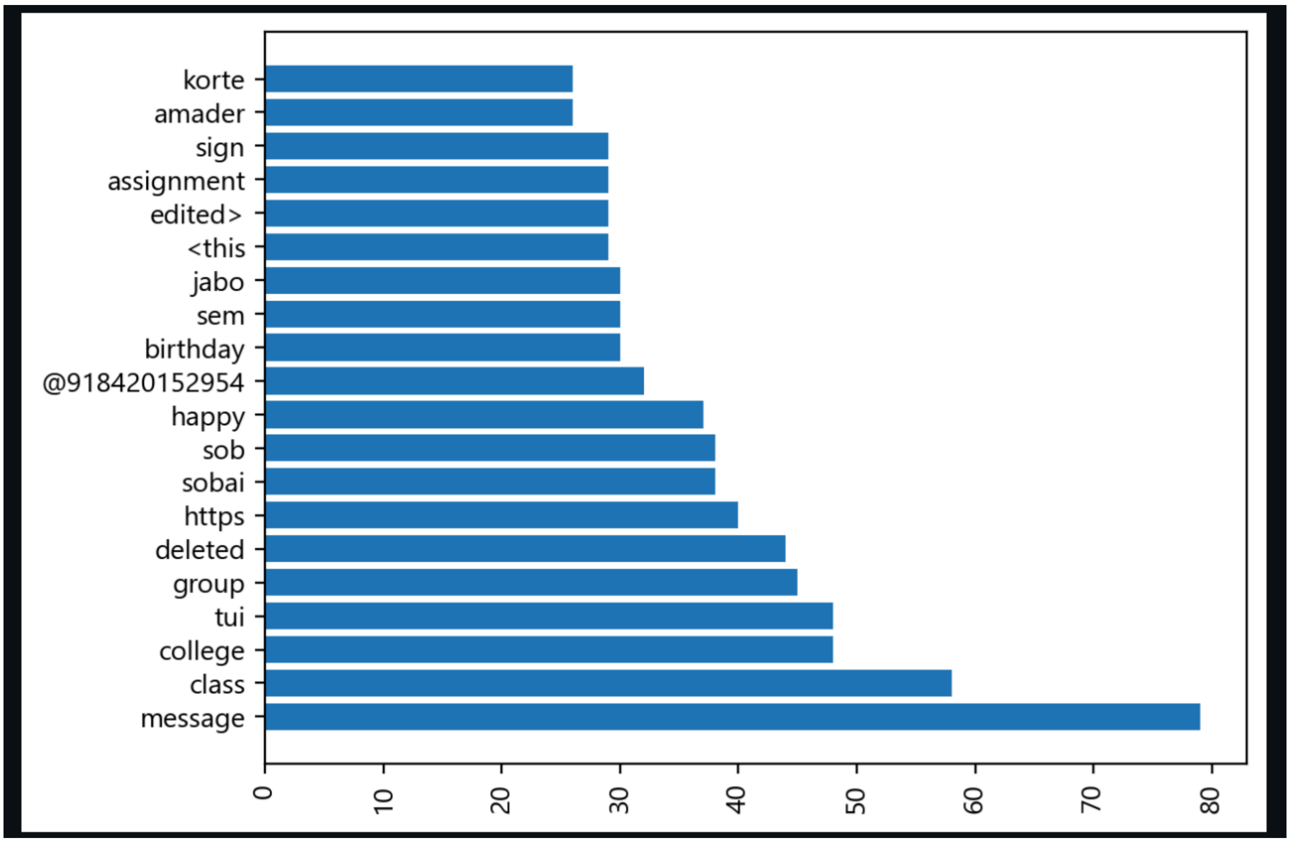
Chat SNCS CodeCartel ⚙️.txt
161.0KB

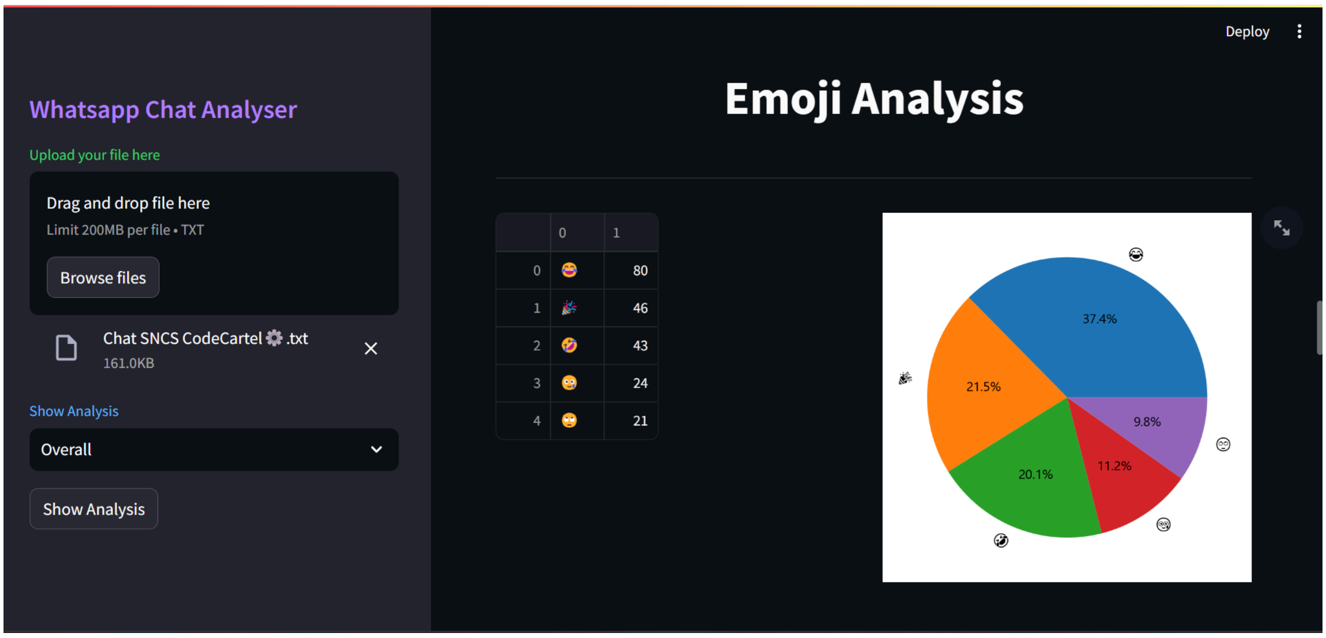
Show Analysis

Overall

Show Analysis

Most Common Words





Whatsapp Chat Analyser

Upload your file here

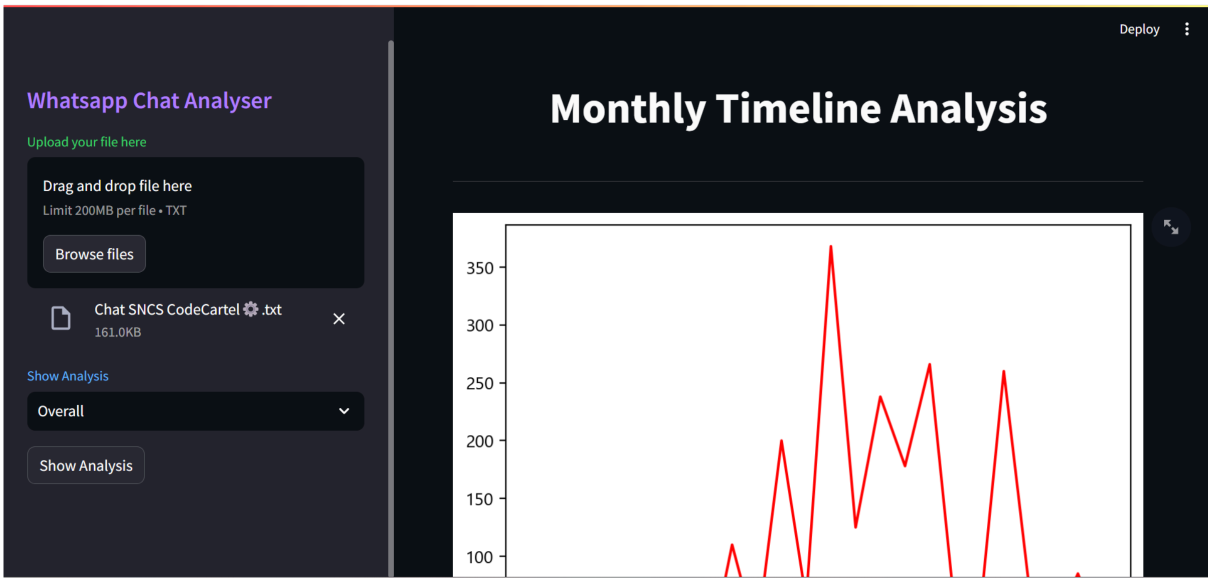
Drag and drop file here
Limit 200MB per file • TXT

Chat SNCS CodeCartel ⚙️.txt
161.0KB

Show Analysis
Overall

Emoji Analysis

	0	1
0	😂	80
1	🎉	46
2	🤣	43
3	🥹	24
4	🙄	21


Deploy

Whatsapp Chat Analyser

Upload your file here

Drag and drop file here

Limit 200MB per file • TXT

Browse files

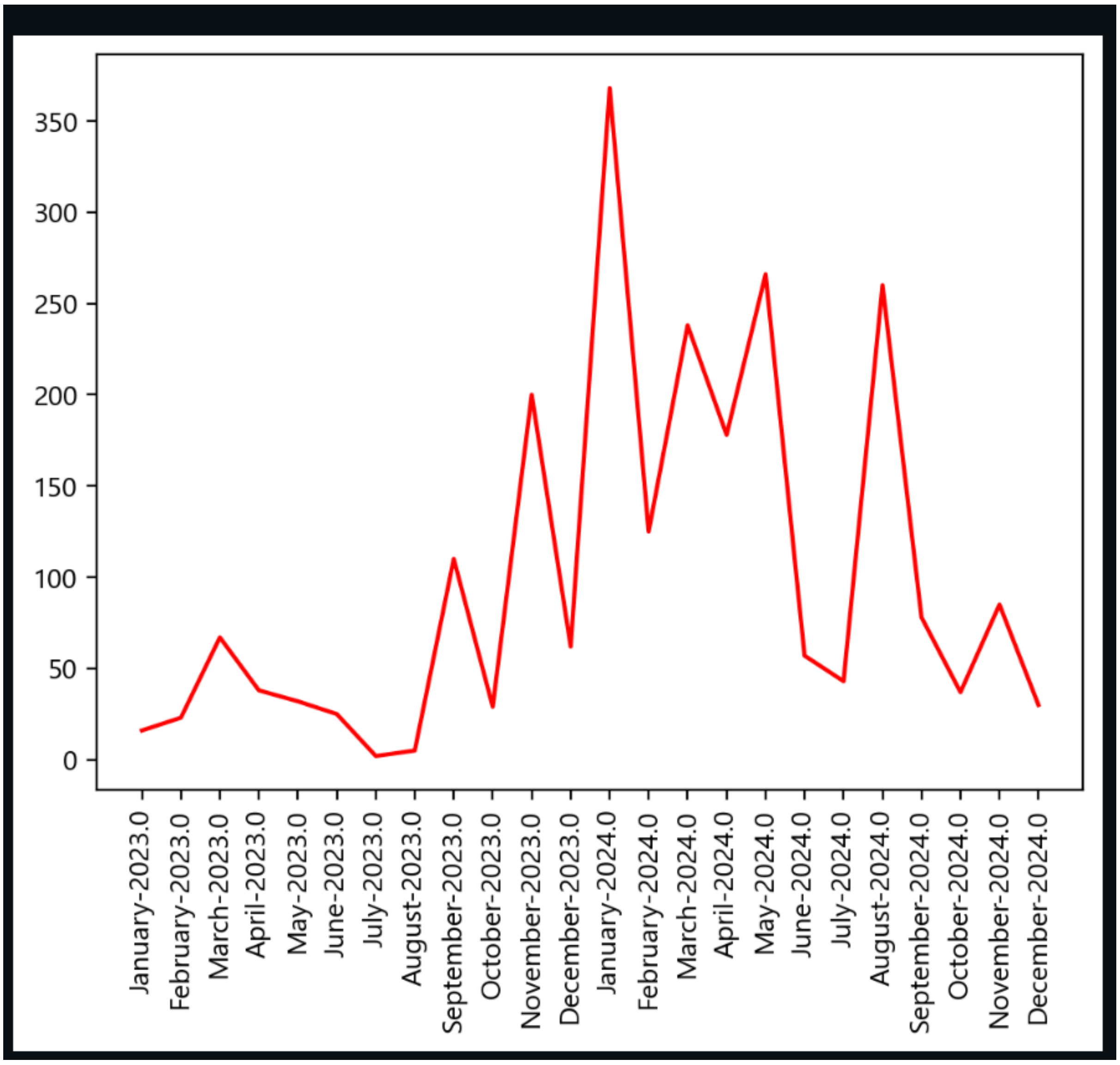
Chat SNCS CodeCartel ⚙ .txt
161.0KB

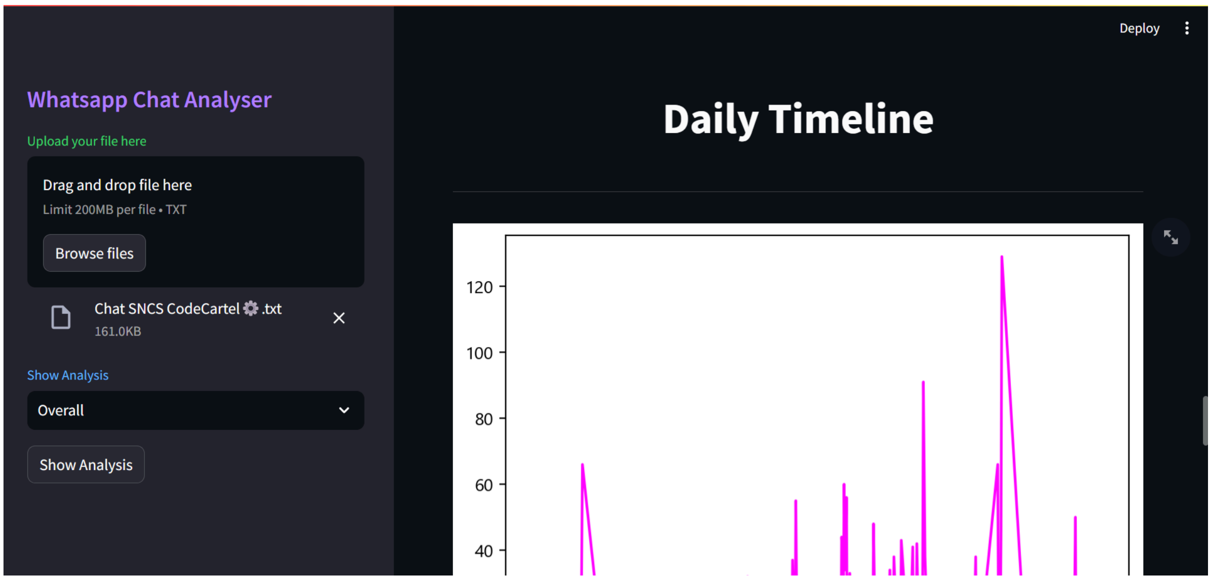
Show Analysis

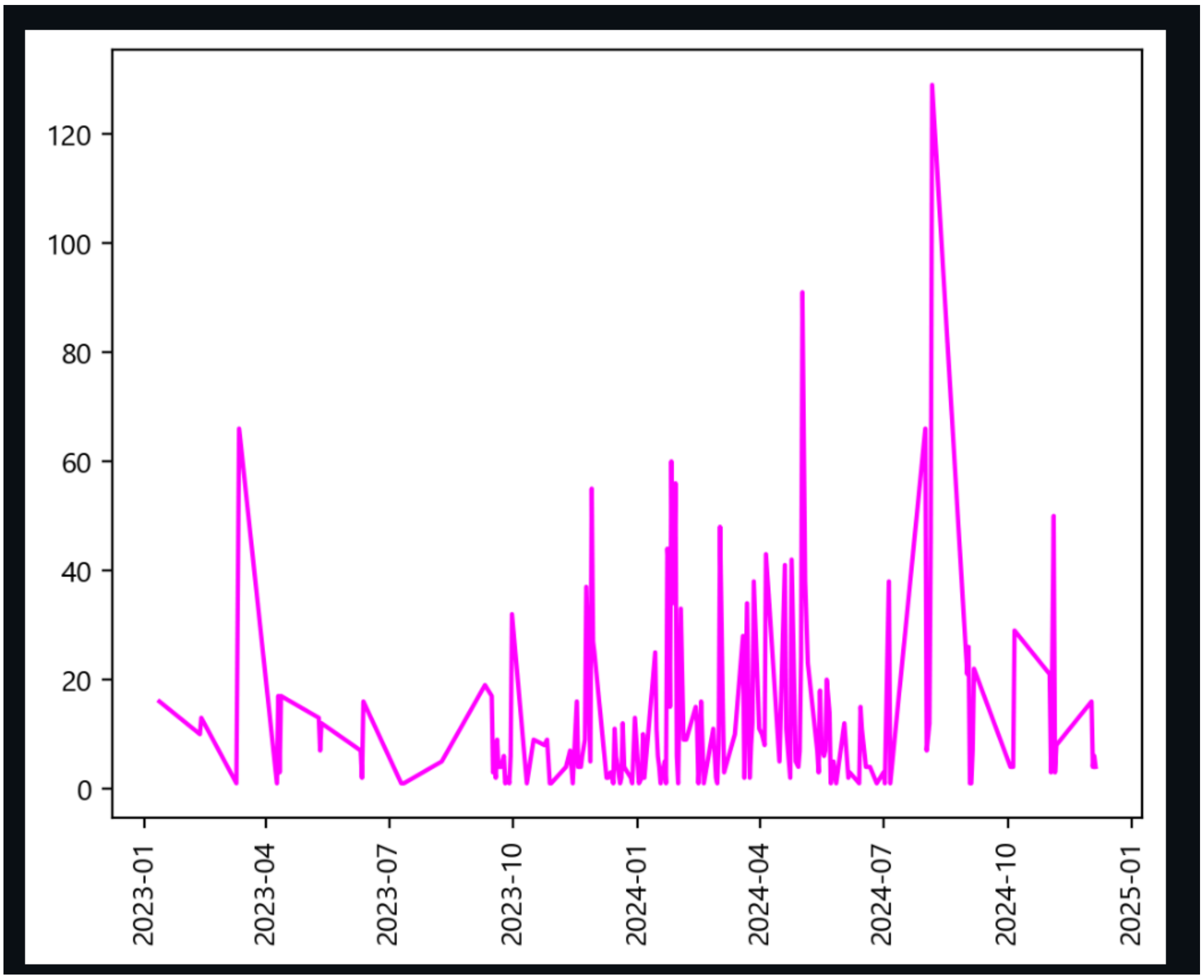
Overall

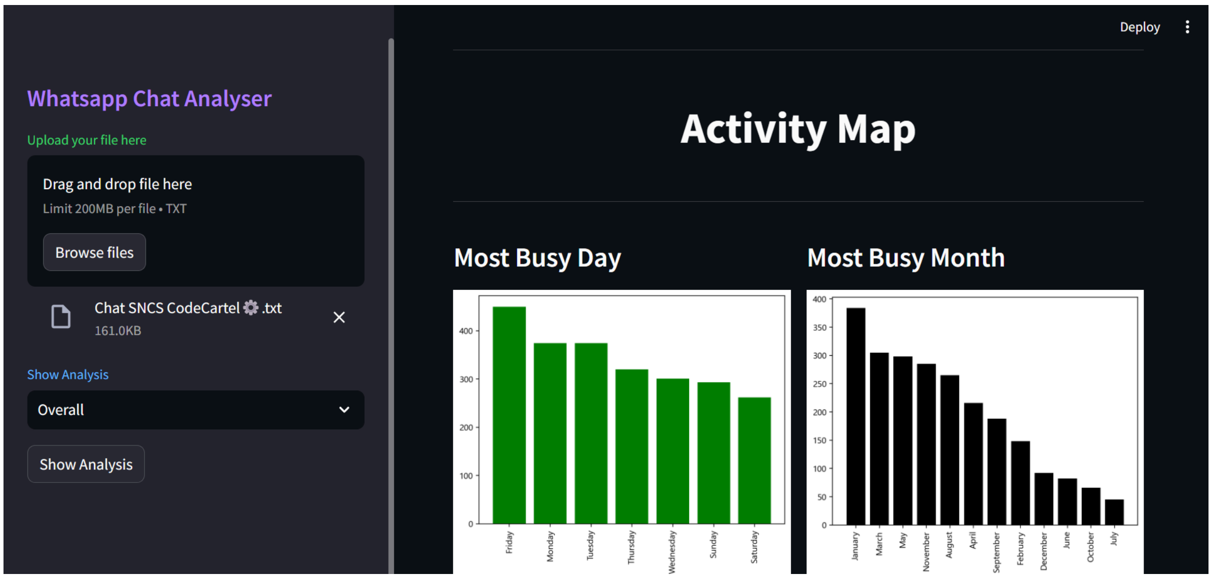
Show Analysis

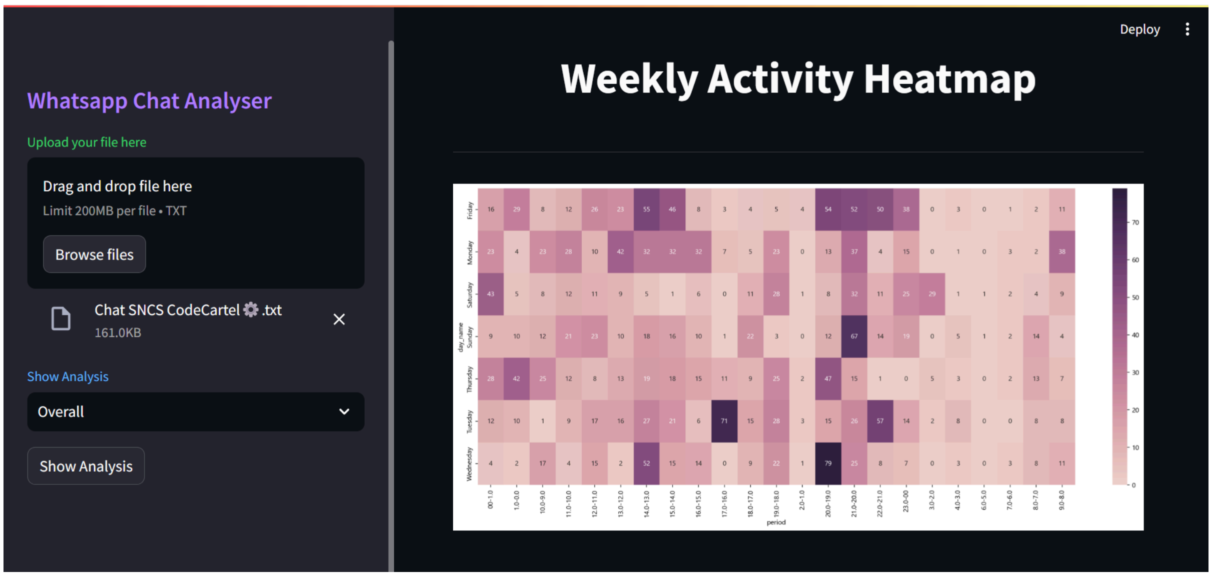
Monthly Timeline Analysis




Deploy
Whatsapp Chat Analyser
Upload your file here
Drag and drop file here
Limit 200MB per file • TXT
Browse files
Chat SNCS CodeCartel ⚙ .txt
161.0KB
Show Analysis
Overall
Show Analysis
Daily Timeline
120
100
80
60
40


120
100
80
60
40
20
0
2023-01
2023-04
2023-07
2023-10
2024-01
2024-04
2024-07
2024-10
2025-01


Deploy
Whatsapp Chat Analyser
Upload your file here
Drag and drop file here
Limit 200MB per file • TXT
Browse files
Chat SNCS CodeCartel ⚙️ .txt
161.0KB
Show Analysis
Overall
Show Analysis
Activity Map
Most Busy Day
Friday
Monday
Tuesday
Thursday
Wednesday
Sunday
Saturday
Most Busy Month
January
March
May
November
August
April
September
February
December
June
October
July


Deploy
Whatsapp Chat Analyser
Upload your file here
Drag and drop file here
Limit 200MB per file • TXT
Browse files
Chat SNCS CodeCartel ⚙️ .txt
161.0KB
Show Analysis
Overall
Show Analysis
Weekly Activity Heatmap
day_name
Friday
Monday
Saturday
Sunday
Thursday
Tuesday
Wednesday
period

Scope of Work

The "Whatsapp Chat Analyser" project is designed to provide users with insights and analytics based on their WhatsApp chat data. The scope of work includes:

1. **Data Collection and Preprocessing:**
    - Extracting chat data from WhatsApp backups.
    - Cleaning and structuring the data for analysis.
    - Handling different formats of exported chat files (e.g., .txt, .json).
2. **Text Analysis:**
    - Performing sentiment analysis to gauge the overall mood of the conversations.
    - Identifying frequently used words and phrases.
    - Detecting key topics and themes discussed in the chats.
3. **Statistical Analysis:**
    - Calculating statistics such as the number of messages sent, average response time, and most active chat periods.
    - Visualizing chat activity over time using graphs and charts.
4. **User Interaction Analysis:**
    - Analyzing interaction patterns between participants.
    - Determining the most active participants in group chats.
    - Measuring the frequency and context of media (images, videos, voice notes) sharing.
5. **Visualization:**
    - Providing a user-friendly dashboard for visual representation of the analysis.
    - Generating reports with visual summaries of the chat data.
6. **Customization and Extensibility:**
    - Allowing users to filter data based on specific time periods or participants.
    - Providing options to export the analysis results in various formats (e.g., PDF, CSV).

Drawbacks

1. **Privacy Concerns:**
    - Handling personal and sensitive data which could lead to privacy issues if not managed securely.
    - The necessity for explicit user consent before analyzing personal chat data.
2. **Data Accuracy and Integrity:**
    - Inconsistencies in chat exports due to different versions of WhatsApp or different devices.
    - Possible inaccuracies in sentiment analysis due to the nuances of human language and context.
3. **Complexity in Text Analysis:**
    - Difficulty in accurately interpreting sarcasm, slang, and regional language variations.
    - Challenges in processing multimedia content (images, videos) which are not text-based.
4. **Resource Intensive:**
    - The analysis, especially for large datasets, can be computationally intensive and time-consuming.
    - High memory and processing power requirements for real-time or large-scale analysis.
5. **User Interface and Experience:**
    - Designing an intuitive and easy-to-use interface that caters to non-technical users.
    - Ensuring the visualizations are meaningful and effectively convey the analysis results.
6. **Scalability Issues:**
    - The system might struggle to handle very large datasets efficiently.
    - Limited by the processing power and memory of the user's machine.

By outlining the scope of work and acknowledging the potential drawbacks, this documentation aims to provide a comprehensive overview of the "Whatsapp Chat Analyser" project, ensuring clear communication of its capabilities and limitations.

Conclusion

The "Whatsapp Chat Analyser" project successfully demonstrates the potential of utilizing Python for text and data analysis within the context of WhatsApp conversations. By extracting and preprocessing chat data, the project provides valuable insights into user interaction patterns, sentiment analysis, and chat activity through various statistical and visual methods. Despite some challenges related to privacy concerns, data accuracy, and computational resources, the project highlights the feasibility and utility of automated chat analysis. It offers a comprehensive tool for users to gain a deeper understanding and visualization of their communication habits, paving the way for further enhancements and applications in personal and professional domains.