```
Start coding or generate with AI.
```

```
import re
import pandas as pd
```

```
file_name = 'WhatsApp Chat with Dbms project.txt'

try:
    with open(file_name, 'r', encoding='utf-8') as f:
        content = f.read()
    print(f"Content of '{file_name}':\n")
    # Print the first 500 characters, or the whole content if it's shorter
    print(content[:500] + ('...' if len(content) > 500 else ''))
except FileNotFoundError:
    print(f"Error: The file '{file_name}' was not found. Please make sure it's uploaded or the path is correct.")
except Exception as e:
    print(f"An error occurred while reading the file: {e}")
```

```
Content of 'WhatsApp Chat with Dbms project.txt':

7/25/24, 9:04 PM - Messages and calls are end-to-end encrypted. Only people in this chat can read, listen to, or share them. L
7/25/24, 9:04 PM - Soham Narvankar created group "Dbms project"
7/25/24, 9:04 PM - Soham Narvankar added you
7/25/24, 11:03 PM - Sarthak CMPN: <Media omitted>
7/26/24, 12:15 PM - Atharva Jadhav CMPN B: <Media omitted>
7/26/24, 12:15 PM - Atharva Jadhav CMPN B: added clasroom
7/26/24, 4:34 PM - Piyush 🎂🎂🐂🐂CMPN B: <Media omitted>
7/26/24, 4:34 PM - Piyush 🎂🎂🐂🐂CMP...
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True
```

```
pattern = r'\d{1,2}/\d{1,2}/\d{2,4},\s\d{1,2}:\d{2}\s[AP]M\s-\s'
```

```
messages = re.split(pattern, content)[1:]
dates = re.findall(pattern, content)
```

```
df = pd.DataFrame({'user_message': messages, 'message_date': dates})
df.head()
```

	user_message	message_date
0	Messages and calls are end-to-end encrypted. O...	7/25/24, 9:04 PM -
1	Soham Narvankar created group "Dbms project"\n	7/25/24, 9:04 PM -
2	Soham Narvankar added you\n	7/25/24, 9:04 PM -
3	Sarthak CMPN: <Media omitted>\n	7/25/24, 11:03 PM -
4	Atharva Jadhav CMPN B: <Media omitted>\n	7/26/24, 12:15 PM -

Next steps: Generate code with df | New interactive sheet

```
df['message_date'] = pd.to_datetime(df['message_date'], format='%m/%d/%y, %I:%M %p - ')
df.head()
```

	user_message	message_date
0	Messages and calls are end-to-end encrypted. O...	2024-07-25 21:04:00
1	Soham Narvankar created group "Dbms project"\n	2024-07-25 21:04:00
2	Soham Narvankar added you\n	2024-07-25 21:04:00
3	Sarthak CMPN: <Media omitted>\n	2024-07-25 23:03:00
4	Atharva Jadhav CMPN B: <Media omitted>\n	2024-07-26 12:15:00

Next steps: Generate code with df | New interactive sheet

```
users = []
messages = []
for message in df['user_message']:
    entry = re.split(r'([\w\W]+?):\s', message)
    if entry[1:]:  # user name
        users.append(entry[1])
        messages.append(" ".join(entry[2:]))
    else:
        users.append('group_notification')
        messages.append(entry[0])

df['user'] = users
df['message'] = messages
df.drop(columns=['user_message'], inplace=True)
```

```
df['only_date'] = df['message_date'].dt.date
df['year'] = df['message_date'].dt.year
df['month_num'] = df['message_date'].dt.month
df['month'] = df['message_date'].dt.month_name()
df['day'] = df['message_date'].dt.day
df['day_name'] = df['message_date'].dt.day_name()
df['hour'] = df['message_date'].dt.hour
df['minute'] = df['message_date'].dt.minute

display(df.head())
```

	message_date	user	message	only_date	year	month_num	month	day	day_name	hour	minute
0	2024-07-25 21:04:00	group_notification	Messages and calls are end-to-end encrypted. O...	2024-07-25	2024	7	July	25	Thursday	21	4
1	2024-07-25 21:04:00	group_notification	Soham Narvankar created group "Dbms project"\n	2024-07-25	2024	7	July	25	Thursday	21	4
2	2024-07-25 21:04:00	group_notification	Soham Narvankar added you\n	2024-07-25	2024	7	July	25	Thursday	21	4
3	2024-07-25 23:03:00	Sarthak CMPN	<Media omitted>\n	2024-07-25	2024	7	July	25	Thursday	23	3

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

Analysis Part

```
!pip install emoji wordcloud
import re
from collections import Counter
from wordcloud import WordCloud
import emoji
import pandas as pd

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

    num_messages = df.shape[0]

    words = []
    for message in df['message']:
        words.extend(message.split())

    num_media_messages = df[df['message'] == '<Media omitted>\n'].shape[0]

    links = []
```

```
        url_pattern = r'(https?://\S+)'
        for message in df['message']:
            links.extend(re.findall(url_pattern, message))

        return num_messages, len(words), num_media_messages, len(links)

    def most_busy_users(df):
        x = df['user'].value_counts().head()
        df_percent = round((df['user'].value_counts() / df.shape[0]) * 100, 2).reset_index().rename(
            columns={'index': 'name', 'user': 'percent'})
        return x, df_percent

    def create_wordcloud(selected_user, df):
        try:
            f = open('stop_hinglish.txt', 'r', encoding='utf-8')
            stop_words_content = f.read()
            stop_words = stop_words_content.split('\n')
        except FileNotFoundError:
            print("Error: 'stop_hinglish.txt' not found. Please make sure it's uploaded or the path is correct.")
            return None
        except Exception as e:
            print(f"An error occurred while reading stop_hinglish.txt: {e}")
            return None

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

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

        def remove_stop_words(message):
            y = []
            for word in message.lower().split():
                if word not in stop_words:
                    y.append(word)
            return " ".join(y)

        if not temp.empty:
            wc = WordCloud(width=500, height=500, min_font_size=10, background_color='white')
            temp['message'] = temp['message'].apply(remove_stop_words)
            df_wc = wc.generate(temp['message'].str.cat(sep=" "))
            return df_wc
        else:
            return None

    def most_common_words(selected_user, df):
        try:
            f = open('stop_hinglish.txt', 'r', encoding='utf-8')
            stop_words_content = f.read()
            stop_words = stop_words_content.split('\n')
        except FileNotFoundError:
            print("Error: 'stop_hinglish.txt' not found. Please make sure it's uploaded or the path is correct.")
            return pd.DataFrame()
        except Exception as e:
            print(f"An error occurred while reading stop_hinglish.txt: {e}")
            return pd.DataFrame()

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

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

        words = []
        for message in temp['message']:
            for word in message.lower().split():
                if word not in stop_words:
                    words.append(word)

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

    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)])

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

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

def daily_timeline(selected_user, df):
    if selected_user != 'Overall':
        df = df[df['user'] == selected_user]

    daily_timeline = df.groupby('only_date').count()['message'].reset_index()

    return daily_timeline

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]

    # Ensure 'period' column exists for heatmap, creating it if not present.
    # This part was previously added and is essential for the function.
    if 'period' not in df.columns:
        df['period'] = df['hour'].apply(lambda x: 'morning' if x < 12 else ('afternoon' if x < 18 else 'night'))

    user_heatmap = df.pivot_table(index='day_name', columns='period', values='message', aggfunc='count').fillna(0)

    return user_heatmap
```

```
Requirement already satisfied: emoji in /usr/local/lib/python3.12/dist-packages (2.15.0)
Requirement already satisfied: wordcloud in /usr/local/lib/python3.12/dist-packages (1.9.6)
Requirement already satisfied: numpy>=1.19.3 in /usr/local/lib/python3.12/dist-packages (from wordcloud) (2.0.2)
Requirement already satisfied: pillow in /usr/local/lib/python3.12/dist-packages (from wordcloud) (11.3.0)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.12/dist-packages (from wordcloud) (3.10.0)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib->wordcloud) (1.3.3
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.12/dist-packages (from matplotlib->wordcloud) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib->wordcloud) (4.61
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib->wordcloud) (1.4.
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib->wordcloud) (25.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib->wordcloud) (3.3.2
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.12/dist-packages (from matplotlib->wordcloud) (2
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.7->matplotlib->wor
```

```
import ipywidgets as widgets
from IPython.display import display
```

```
user_dropdown = widgets.Dropdown(
    options=user_list,
    value='Overall',
    description='Select User:',
    disabled=False,
)

display(user_dropdown)
```

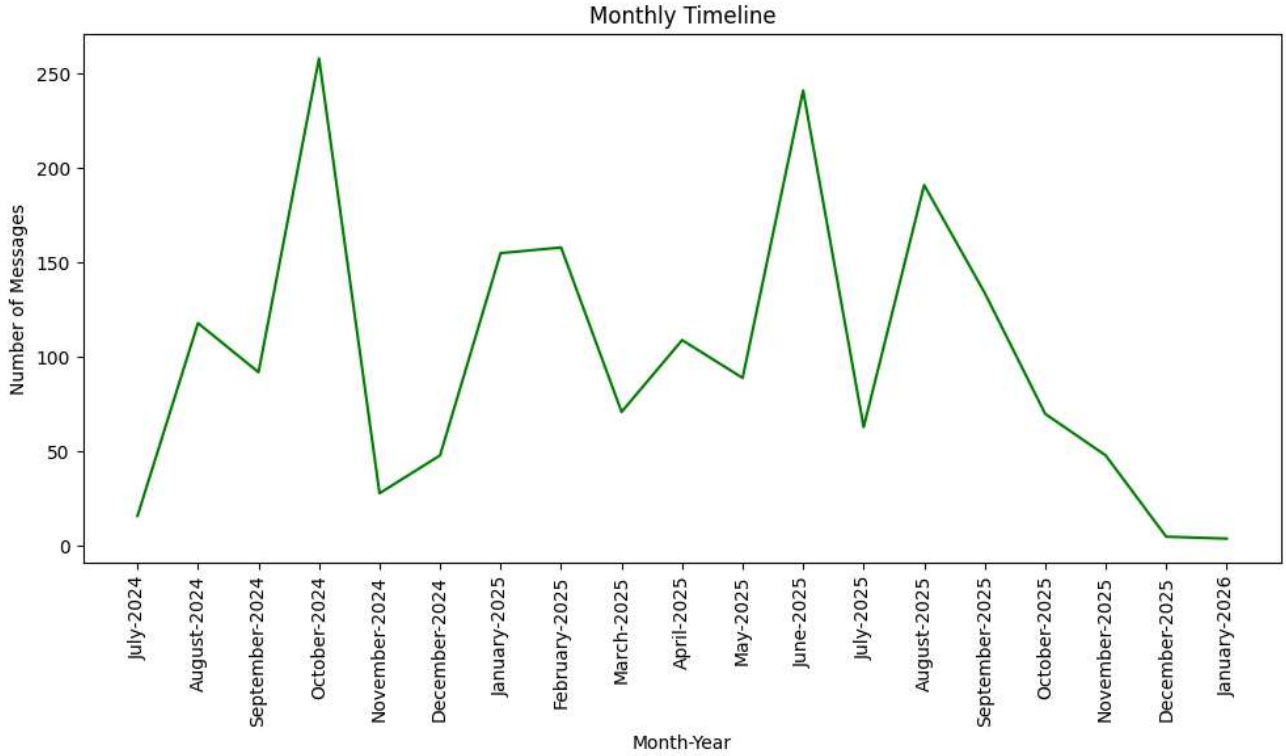
Select User: Overall

```
import matplotlib.pyplot as plt
import seaborn as sns
from IPython.display import display # For displaying DataFrames nicely

# Assuming 'user_dropdown' is available from a previous cell and 'df' is the processed DataFrame
selected_user = user_dropdown.value

# Content for stop_hinglish.txt based on the previous code state
stop_words_content_str = ", ., ..., .., ?, -, --, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, a, aadi, aaj, aap, aapne, aata, aati, aaya, a

# Create the file 'stop_hinglish.txt' with this content
try:
    with open('stop_hinglish.txt', 'w', encoding='utf-8') as f:
        f.write(stop_words_content_str.strip())
    print("Successfully created 'stop_hinglish.txt'.")
except Exception as e:
    print(f"An error occurred while creating 'stop_hinglish.txt': {e}")


# --- Top Statistics ---
print("\n--- Top Statistics ---")
num_messages, words, num_media_messages, num_links = fetch_stats(selected_user, df)
print(f"Total Messages: {num_messages}")
print(f"Total Words: {words}")
print(f"Media Shared: {num_media_messages}")
print(f"Links Shared: {num_links}")

# --- Monthly Timeline ---
print("\n--- Monthly Timeline ---")
timeline = monthly_timeline(selected_user, df)
if not timeline.empty:
    fig, ax = plt.subplots(figsize=(10, 6))
    ax.plot(timeline['time'], timeline['message'], color='green')
    plt.xticks(rotation='vertical')
    plt.title("Monthly Timeline")
    plt.xlabel("Month-Year")
    plt.ylabel("Number of Messages")
    plt.tight_layout()
    plt.show()
else:
    print("No data to display for monthly timeline.")

# --- Daily Timeline ---
print("\n--- Daily Timeline ---")
daily_timeline_df = daily_timeline(selected_user, df)
if not daily_timeline_df.empty:
    fig, ax = plt.subplots(figsize=(10, 6))
    ax.plot(daily_timeline_df['only_date'], daily_timeline_df['message'], color='black')
    plt.xticks(rotation='vertical')
    plt.title("Daily Timeline")
    plt.xlabel("Date")
    plt.ylabel("Number of Messages")
    plt.tight_layout()
    plt.show()
else:
    print("No data to display for daily timeline.")

# --- Activity Map ---
print("\n--- Activity Map ---")

# Most busy day
busy_day = week_activity_map(selected_user, df)
if not busy_day.empty:
```

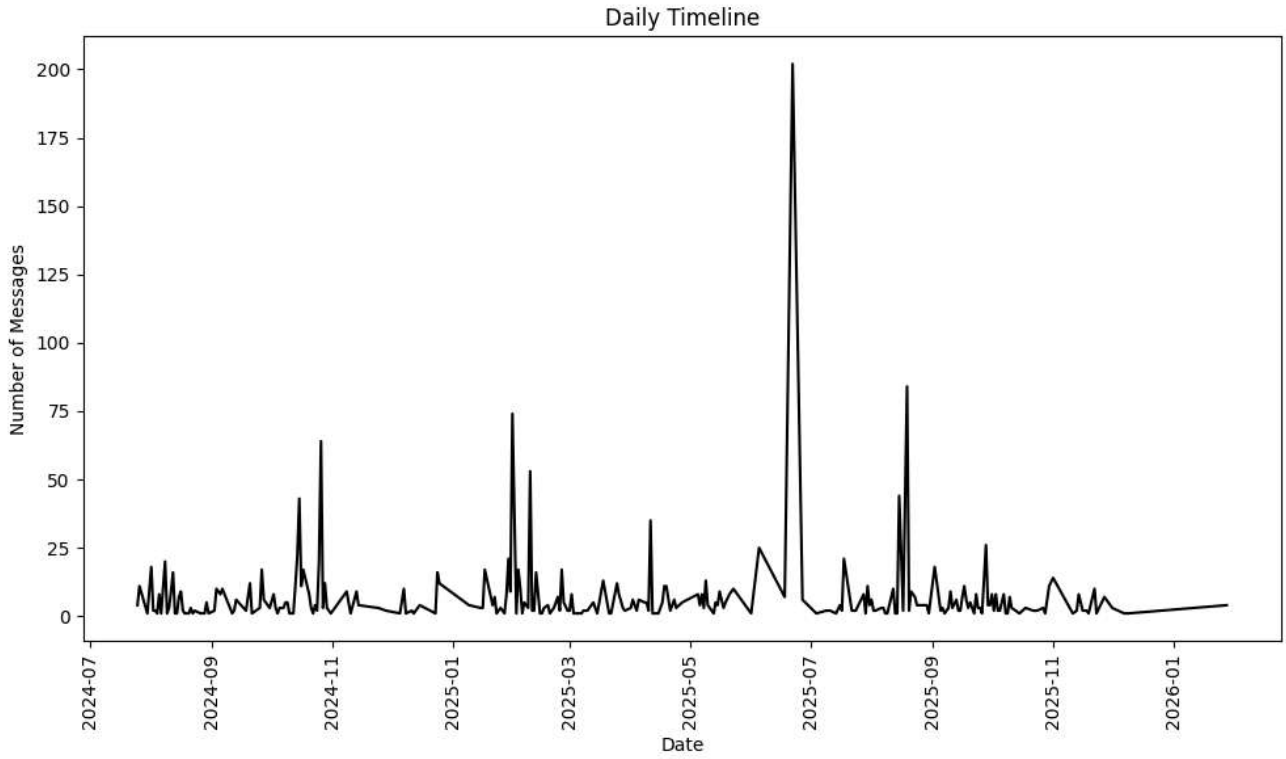
```
        print("Most busy day:")
        fig, ax = plt.subplots(figsize=(8, 5))
        ax.bar(busy_day.index, busy_day.values, color='purple')
        plt.xticks(rotation='vertical')
        plt.title("Most Busy Day")
        plt.xlabel("Day of Week")
        plt.ylabel("Number of Messages")
        plt.tight_layout()
        plt.show()
    else:
        print("No data to display for most busy day.")

    # Most busy month
    busy_month = month_activity_map(selected_user, df)
    if not busy_month.empty:
        print("\nMost busy month:")
        fig, ax = plt.subplots(figsize=(8, 5))
        ax.bar(busy_month.index, busy_month.values, color='orange')
        plt.xticks(rotation='vertical')
        plt.title("Most Busy Month")
        plt.xlabel("Month")
        plt.ylabel("Number of Messages")
        plt.tight_layout()
        plt.show()
    else:
        print("No data to display for most busy month.")

    print("\n--- Weekly Activity Map (Heatmap) ---")
    user_heatmap = activity_heatmap(selected_user, df)
    if not user_heatmap.empty:
        fig, ax = plt.subplots(figsize=(10, 7))
        sns.heatmap(user_heatmap, ax=ax, cmap='YlGnBu')
        plt.title("Weekly Activity Heatmap")
        plt.xlabel("Period of Day")
        plt.ylabel("Day of Week")
        plt.tight_layout()
        plt.show()
    else:
        print("No data to display for weekly activity heatmap.")

    # --- Most Busy Users (Group level) ---
    if selected_user == 'Overall':
        print("\n--- Most Busy Users (Overall Group) ---")
        x, new_df = most_busy_users(df)
        if not x.empty:
            fig, ax = plt.subplots(figsize=(10, 6))
            ax.bar(x.index, x.values, color='red')
            plt.xticks(rotation='vertical')
            plt.title("Top 5 Most Busy Users")
            plt.xlabel("User")
            plt.ylabel("Number of Messages")
            plt.tight_layout()
            plt.show()
        else:
            print("No data to display for top busy users.")

        if not new_df.empty:
            print("\nPercentage of Messages by User:")
            display(new_df)
        else:
            print("No data to display for message percentage by user.")

    # --- WordCloud ---
    print("\n--- Wordcloud ---")
    df_wc = create_wordcloud(selected_user, df)
    if df_wc:
        fig, ax = plt.subplots(figsize=(10, 8))
        ax.imshow(df_wc)
        plt.axis('off')
        plt.title("Word Cloud")
        plt.show()
    else:
        print("Could not generate word cloud. Check if 'stop_hinglish.txt' exists and enough message data is available.")

    # --- Most Common Words ---
    print("\n--- Most Common Words ---")
```

```
    most_common_df = most_common_words(selected_user, df)
    if not most_common_df.empty:
        fig, ax = plt.subplots(figsize=(12, 8))
        ax.barh(most_common_df[0], most_common_df[1], color='skyblue')
        ax.invert_yaxis() # To display the most common word at the top
        plt.xticks(rotation='vertical')
        plt.title('Most Common Words')
        plt.xlabel("Count")
        plt.ylabel("Word")
        plt.tight_layout()
        plt.show()
    else:
        print("No data to display for most common words.")

    # --- Emoji Analysis ---
    print("\n--- Emoji Analysis ---")
    emoji_df = emoji_helper(selected_user, df)
    if not emoji_df.empty:
        print("Top Emojis:")
        display(emoji_df.head())

        if len(emoji_df) > 0:
            fig, ax = plt.subplots(figsize=(8, 8))
            ax.pie(emoji_df[1].head(), labels=emoji_df[0].head(), autopct="%0.2f%%")
            plt.title("Emoji Distribution (Top 5)")
            plt.tight_layout()
            plt.show()
        else:
            print("No emojis to display in pie chart.")
    else:
        print("No emojis found for analysis.")
```

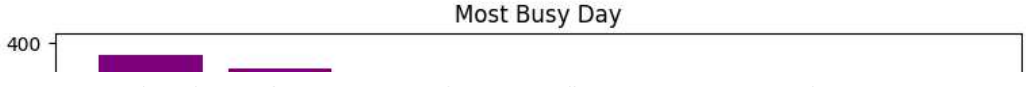
```
Successfully created 'stop_hinglish.txt'.

--- Top Statistics ---
Total Messages: 1898
Total Words: 25079
Media Shared: 1147
Links Shared: 106

--- Monthly Timeline ---
```

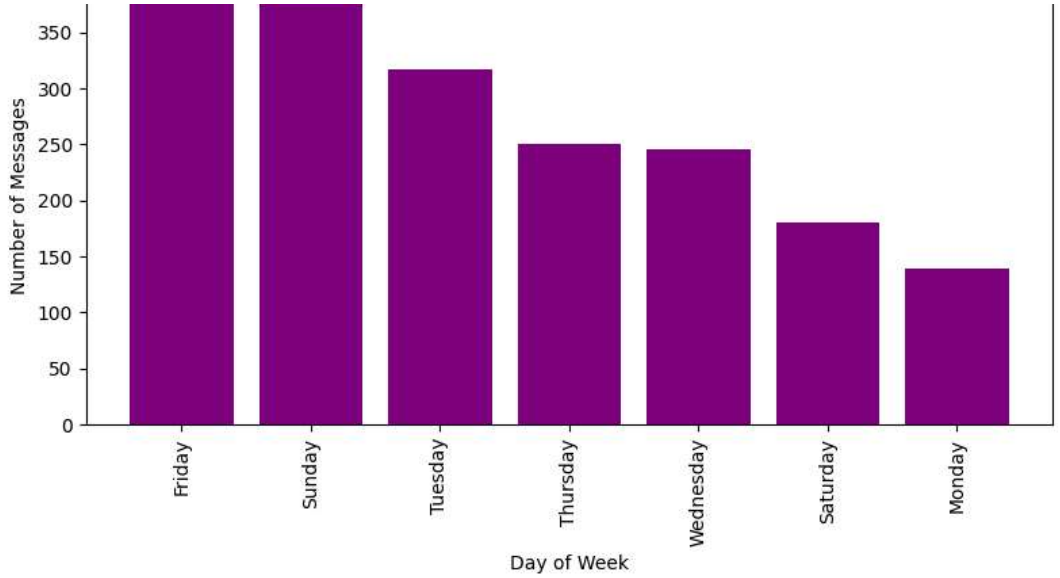



```
--- Daily Timeline ---
```

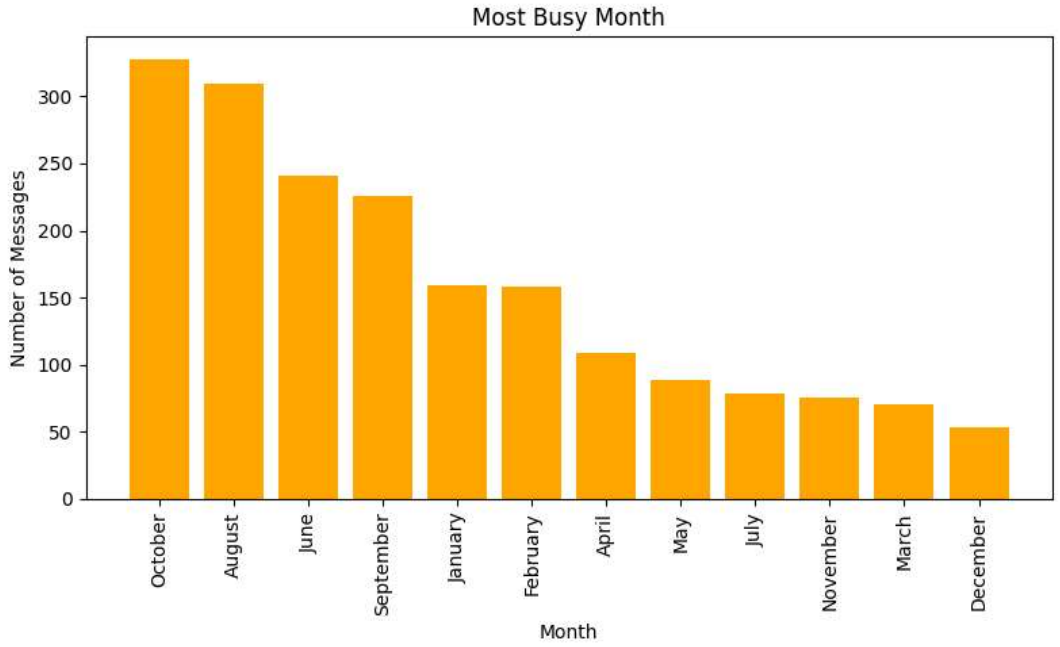



```
--- Activity Map ---
Most busy day:
```

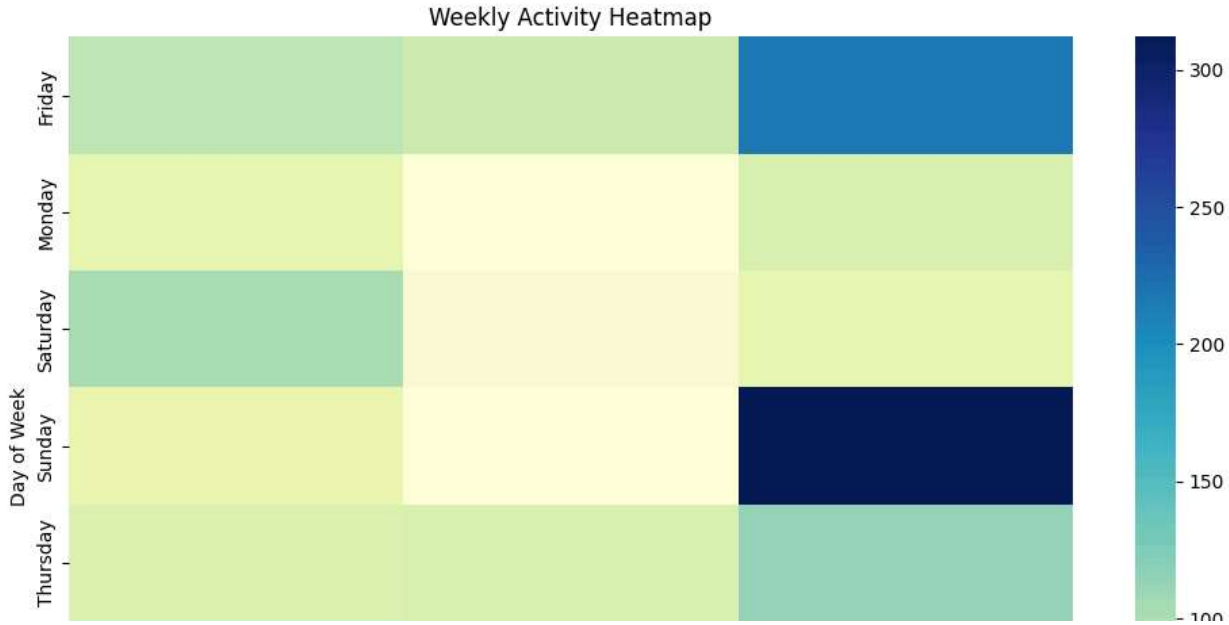


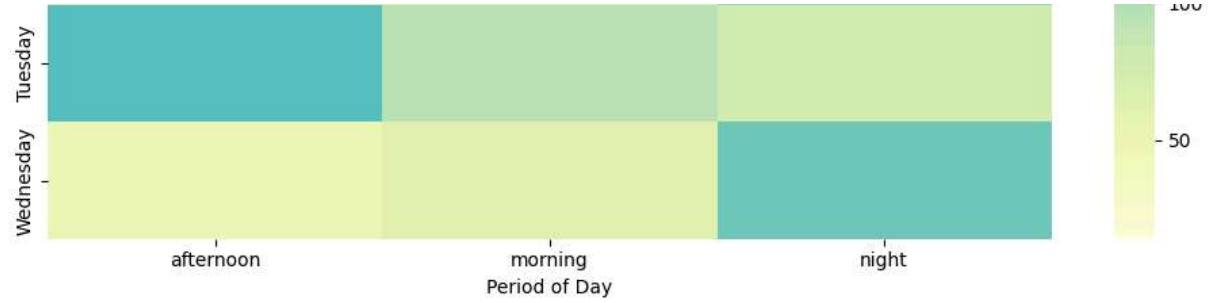



```
Most busy month:
```

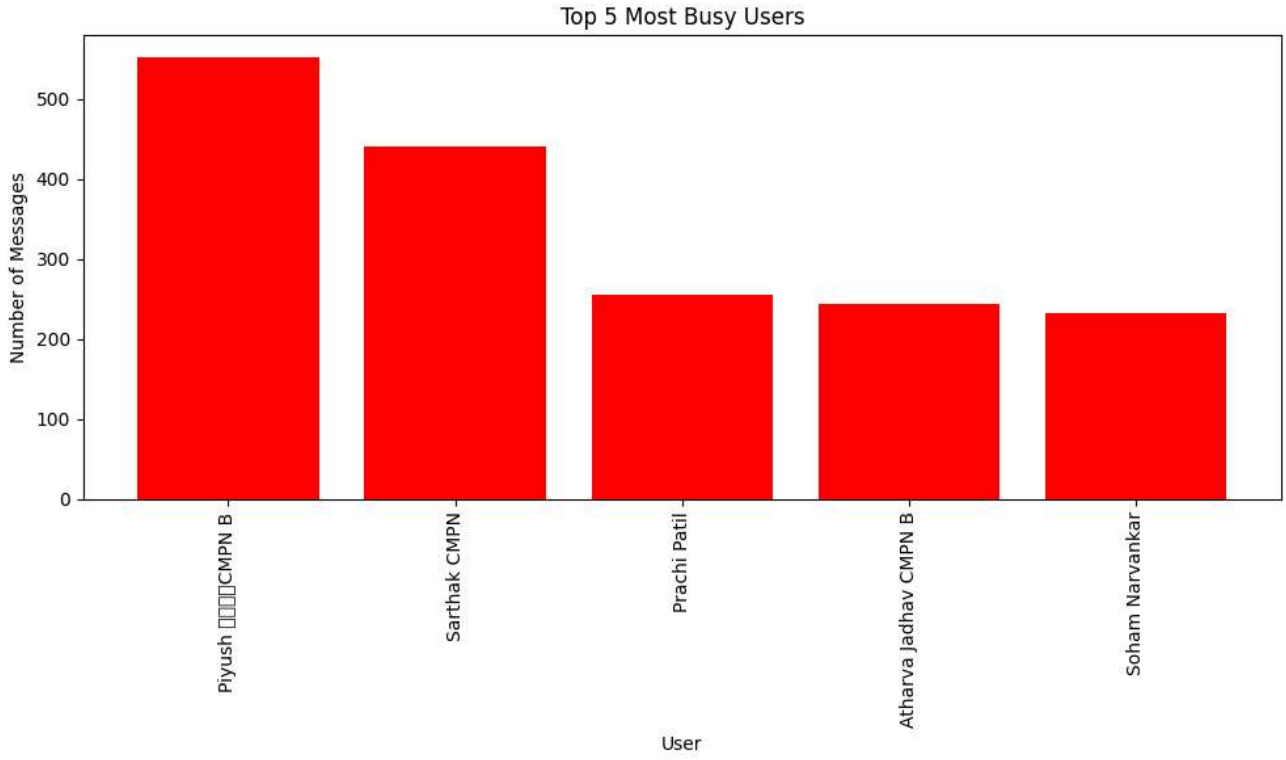



```
--- Weekly Activity Map (Heatmap) ---
```





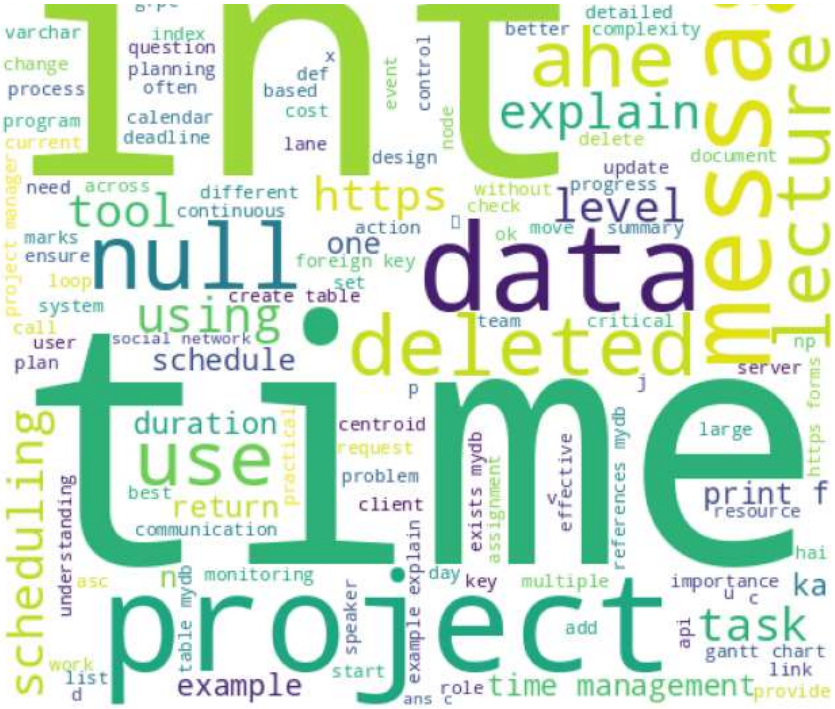

```
--- Most Busy Users (Overall Group) ---
/tmp/ipython-input-2201148171.py:115: UserWarning: Glyph 128055 (\N{PIG FACE}) missing from font(s) DejaVu Sans.
  plt.tight_layout()
/tmp/ipython-input-2201148171.py:115: UserWarning: Glyph 128003 (\N{WATER BUFFALO}) missing from font(s) DejaVu Sans.
  plt.tight_layout()
/usr/local/lib/python3.12/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 128055 (\N{PIG FACE}) missing from
  fig.canvas.print_figure(bytes_io, **kw)
/usr/local/lib/python3.12/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 128003 (\N{WATER BUFFALO}) missing
  fig.canvas.print_figure(bytes_io, **kw)
```



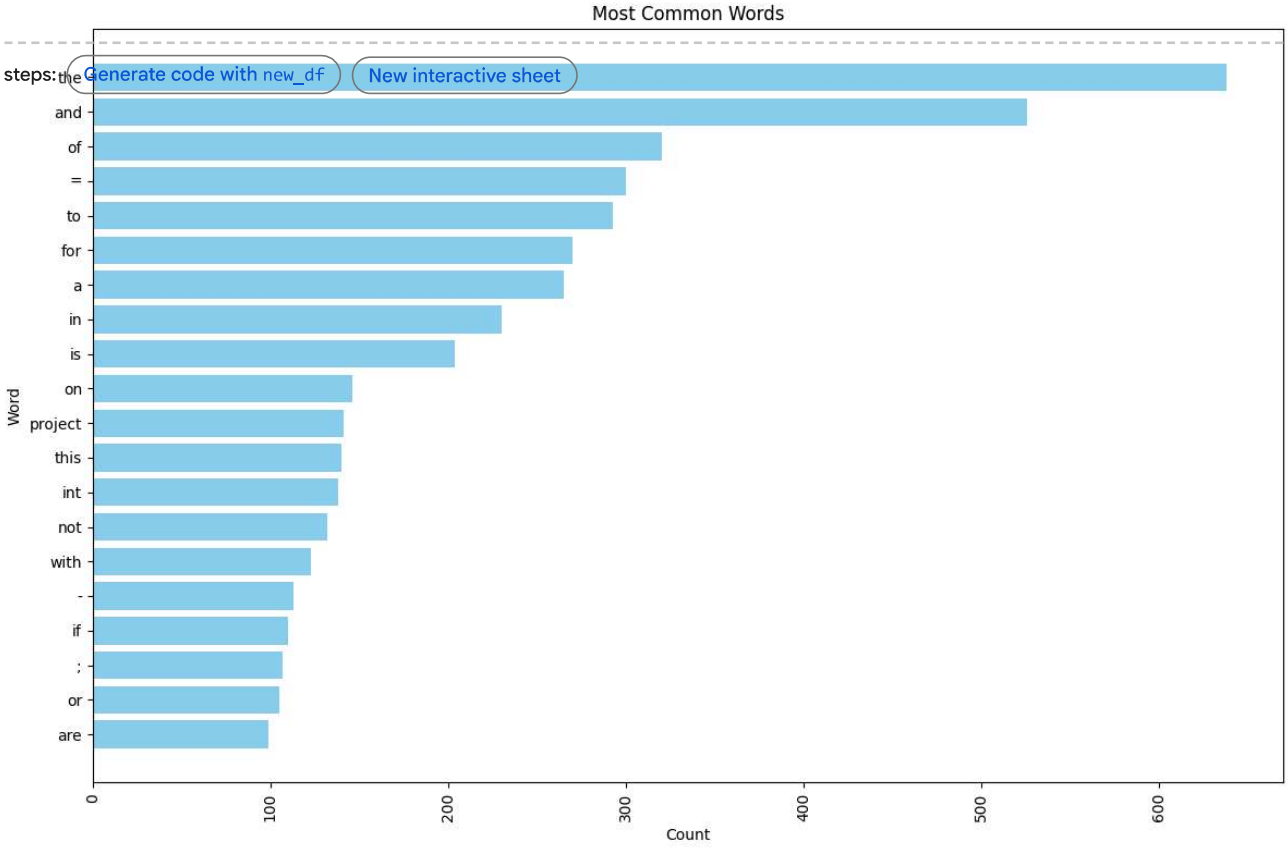
Percentage of Messages by User:

		percent	count
0	Piyush 🐷🐷🐂🐂CMPN B	29.08	
1	Sarthak CMPN	23.18	
2	Prachi Patil	13.49	
3	Atharva Jadhav CMPN B	12.86	
4	Soham Narvankar	12.22	
5	Harshal Patil CMPN A	7.11	
6	Rishi CMPN A	1.48	
7	group_notification	0.58	

```
--- Wordcloud ---
```




```
--- Most Common Words ---
```




```
--- Emoji Analysis ---
Top Emojis:
```

	0	1
0	✅	20
1	🐷	18
2	🐂	18
3	🔹	17
4	❓	15

```
/tmp/ipython-input-2201148171.py:165: UserWarning: Glyph 9989 (\N{WHITE HEAVY CHECK MARK}) missing from font(s) DejaVu Sans
```

Start coding or generate with AI.

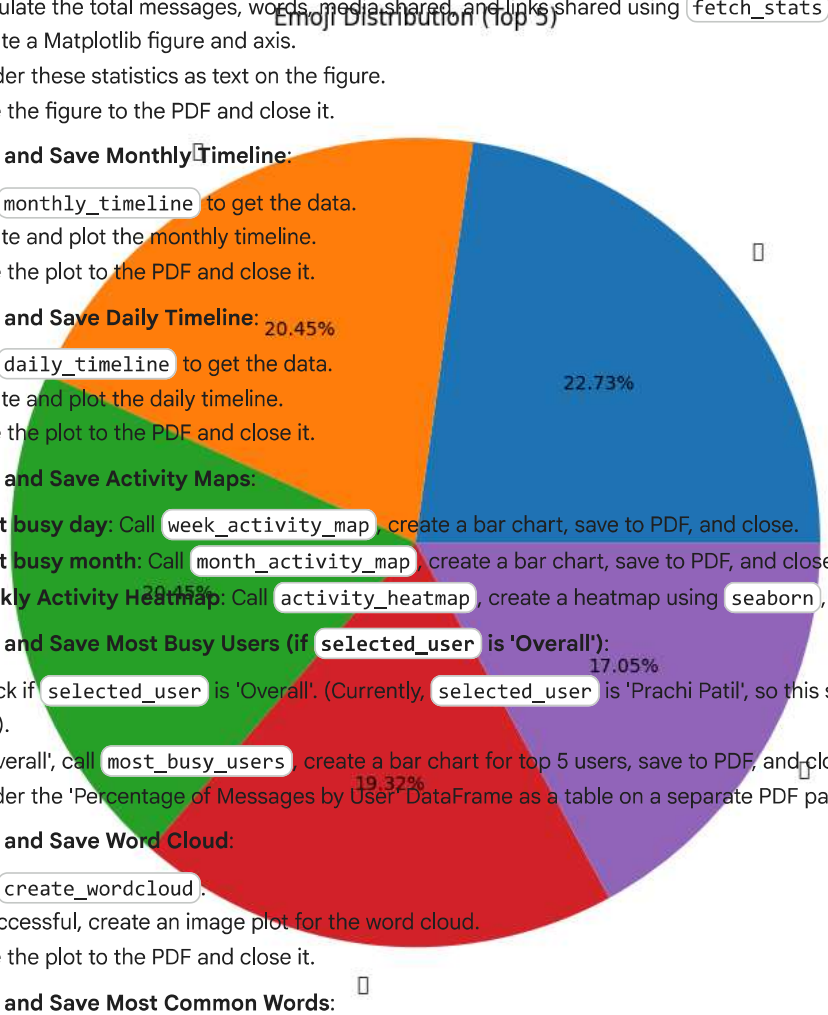
```
  plt.tight_layout()
/tmp/ipython-input-2201148171.py:165: UserWarning: Glyph 128003 (\N{WATER BUFFALO}) missing from font(s) DejaVu Sans.
  plt.tight_layout()
/tmp/ipython-input-2201148171.py:165: UserWarning: Glyph 128313 (\N{SMALL BLUE DIAMOND}) missing from font(s) DejaVu Sans.
  plt.tight_layout()
/tmp/ipython-input-2201148171.py:165: UserWarning: Glyph 10067 (\N{BLACK QUESTION MARK ORNAMENT}) missing from font(s) DejaVu
  plt.tight_layout()
/usr/local/lib/python3.12/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 9989 (\N{WHITE HEAVY CHECK MARK})
  fig.canvas.print_figure(bytes_io, **kw)
/usr/local/lib/python3.12/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 128055 (\N{PIG FACE}) missing from
  fig.canvas.print_figure(bytes_io, **kw)
/usr/local/lib/python3.12/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 128003 (\N{WATER BUFFALO}) missing
  fig.canvas.print_figure(bytes_io, **kw)
/usr/local/lib/python3.12/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 128313 (\N{SMALL BLUE DIAMOND}) mi
  fig.canvas.print_figure(bytes_io, **kw)
/usr/local/lib/python3.12/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 10067 (\N{BLACK QUESTION MARK ORNA
  fig.canvas.print_figure(bytes_io, **kw)
```

Task

The user wants to generate a detailed PDF report summarizing the WhatsApp chat analysis. This report should include various statistical plots and tables derived from the chat data.

To achieve this, I'll execute the following steps in the next code cell:

1. **Import necessary libraries**: `matplotlib.backends.backend_pdf.PdfPages` for PDF generation, `matplotlib.pyplot` for plotting, and `seaborn` for enhanced visualizations.
2. **Initialize PDF**: Create a `PdfPages` object named 'whatsapp_chat_analysis_report.pdf'.
3. **Generate and Save Top Statistics**:
   - Calculate the total messages, words, media shared, and links shared using `fetch_stats`.
   - Create a Matplotlib figure and axis.
   - Render these statistics as text on the figure.
   - Save the figure to the PDF and close it.
4. **Generate and Save Monthly Timeline**:
   - Call `monthly_timeline` to get the data.
   - Create and plot the monthly timeline.
   - Save the plot to the PDF and close it.
5. **Generate and Save Daily Timeline**:
   - Call `daily_timeline` to get the data.
   - Create and plot the daily timeline.
   - Save the plot to the PDF and close it.
6. **Generate and Save Activity Maps**:
   - **Most busy day**: Call `week_activity_map`, create a bar chart, save to PDF, and close.
   - **Most busy month**: Call `month_activity_map`, create a bar chart, save to PDF, and close.
   - **Weekly Activity Heatmap**: Call `activity_heatmap`, create a heatmap using `seaborn`, save to PDF, and close.
7. **Generate and Save Most Busy Users (if `selected_user` is 'Overall')**:
   - Check if `selected_user` is 'Overall'. (Currently, `selected_user` is 'Prachi Patil', so this section will be skipped as per the plan).
   - If 'Overall', call `most_busy_users`, create a bar chart for top 5 users, save to PDF, and close.
   - Render the 'Percentage of Messages by User' DataFrame as a table on a separate PDF page, then close the figure.
8. **Generate and Save Word Cloud**:
   - Call `create_wordcloud`.
   - If successful, create an image plot for the word cloud.
   - Save the plot to the PDF and close it.
9. **Generate and Save Most Common Words**:
   - Call `most_common_words`.
   - Create a horizontal bar chart of the most common words.
   - Save the plot to the PDF and close it.
10. **Generate and Save Emoji Analysis**:
    - Call `emoji_helper`.
    - Render the top emojis DataFrame as a table on a PDF page.
    - Create a pie chart for the top emojis, save to PDF, and close it.
11. **Finalize PDF**: Close the `pdf_pages` object to complete the PDF file creation.
12. **Provide Output**: Print a confirmation message indicating the PDF report has been generated and saved.




```
from matplotlib.backends.backend_pdf import PdfPages
import matplotlib.pyplot as plt
import seaborn as sns
```

```
import pandas as pd # Import pandas for DataFrame display in PDF

# Re-fetch selected_user in case it was changed via the dropdown in a previous cell run
# This ensures we use the correct user for the PDF generation.
# If the dropdown is not run again, selected_user will retain its value from the last run.
# For the purpose of this execution, we'll assume the state of selected_user from the kernel.
# In the kernel state, selected_user is 'Prachi Patil'.

# Initialize PDF document
pdf_pages = PdfPages('whatsapp_chat_analysis_report.pdf')

# --- 1. Generate and Save Top Statistics ---
num_messages, words, num_media_messages, num_links = fetch_stats(selected_user, df)

fig_stats, ax_stats = plt.subplots(figsize=(8, 6))
ax_stats.axis('off') # Hide axes for text display

stats_text = f"""
WhatsApp Chat Analysis Report for: {selected_user}

Total Messages: {num_messages}
Total Words: {words}
Media Shared: {num_media_messages}
Links Shared: {num_links}
"""
ax_stats.text(0.1, 0.9, stats_text, fontsize=12, verticalalignment='top',
              bbox=dict(boxstyle='round,pad=0.5', fc='wheat', alpha=0.5))
ax_stats.set_title(f'Top Statistics for {selected_user}', fontsize=14)
pdf_pages.savefig(fig_stats)
plt.close(fig_stats)


# --- 2. Generate and Save Monthly Timeline ---
timeline = monthly_timeline(selected_user, df)
if not timeline.empty:
    fig_monthly, ax_monthly = plt.subplots(figsize=(10, 6))
    ax_monthly.plot(timeline['time'], timeline['message'], color='green')
    plt.xticks(rotation='vertical')
    ax_monthly.set_title(f"Monthly Timeline for {selected_user}")
    ax_monthly.set_xlabel("Month-Year")
    ax_monthly.set_ylabel("Number of Messages")
    plt.tight_layout()
    pdf_pages.savefig(fig_monthly)
    plt.close(fig_monthly)
else:
    print(f"No data to display for monthly timeline for {selected_user}.")


# --- 3. Generate and Save Daily Timeline ---
daily_timeline_df = daily_timeline(selected_user, df)
if not daily_timeline_df.empty:
    fig_daily, ax_daily = plt.subplots(figsize=(10, 6))
    ax_daily.plot(daily_timeline_df['only_date'], daily_timeline_df['message'], color='black')
    plt.xticks(rotation='vertical')
    ax_daily.set_title(f"Daily Timeline for {selected_user}")
    ax_daily.set_xlabel("Date")
    ax_daily.set_ylabel("Number of Messages")
    plt.tight_layout()
    pdf_pages.savefig(fig_daily)
    plt.close(fig_daily)
else:
    print(f"No data to display for daily timeline for {selected_user}.")
```

```
# --- 4. Generate and Save Activity Maps ---

# Most busy day
busy_day = week_activity_map(selected_user, df)
if not busy_day.empty:
    fig_busy_day, ax_busy_day = plt.subplots(figsize=(8, 5))
    ax_busy_day.bar(busy_day.index, busy_day.values, color='purple')
    plt.xticks(rotation='vertical')
    ax_busy_day.set_title(f"Most Busy Day for {selected_user}")
    ax_busy_day.set_xlabel("Day of Week")
    ax_busy_day.set_ylabel("Number of Messages")
    plt.tight_layout()
    pdf_pages.savefig(fig_busy_day)
    plt.close(fig_busy_day)
else:
    print(f"No data to display for most busy day for {selected_user}.")

# Most busy month
busy_month = month_activity_map(selected_user, df)
if not busy_month.empty:
    fig_busy_month, ax_busy_month = plt.subplots(figsize=(8, 5))
    ax_busy_month.bar(busy_month.index, busy_month.values, color='orange')
    plt.xticks(rotation='vertical')
    ax_busy_month.set_title(f"Most Busy Month for {selected_user}")
    ax_busy_month.set_xlabel("Month")
    ax_busy_month.set_ylabel("Number of Messages")
    plt.tight_layout()
    pdf_pages.savefig(fig_busy_month)
    plt.close(fig_busy_month)
else:
    print(f"No data to display for most busy month for {selected_user}.")

# Weekly Activity Heatmap
user_heatmap = activity_heatmap(selected_user, df)
if not user_heatmap.empty:
    fig_heatmap, ax_heatmap = plt.subplots(figsize=(10, 7))
    sns.heatmap(user_heatmap, ax=ax_heatmap, cmap='YlGnBu', annot=True, fmt='g')
    ax_heatmap.set_title(f"Weekly Activity Heatmap for {selected_user}")
    ax_heatmap.set_xlabel("Period of Day")
    ax_heatmap.set_ylabel("Day of Week")
    plt.tight_layout()
    pdf_pages.savefig(fig_heatmap)
    plt.close(fig_heatmap)
else:
    print(f"No data to display for weekly activity heatmap for {selected_user}.")


# --- 5. Generate and Save Most Busy Users (Group level - only if selected_user is 'Overall') ---
if selected_user == 'Overall':
    x, new_df = most_busy_users(df)
    if not x.empty:
        fig_top_users, ax_top_users = plt.subplots(figsize=(10, 6))
        ax_top_users.bar(x.index, x.values, color='red')
        plt.xticks(rotation='vertical')
        ax_top_users.set_title("Top 5 Most Busy Users (Overall Group)")
        ax_top_users.set_xlabel("User")
        ax_top_users.set_ylabel("Number of Messages")
        plt.tight_layout()
        pdf_pages.savefig(fig_top_users)
        plt.close(fig_top_users)
    else:
        print("No data to display for top busy users.")

    if not new_df.empty:
```

```
        fig_percent, ax_percent = plt.subplots(figsize=(8, len(new_df) * 0.5)) # Adjust figure size dynamically
        ax_percent.axis('off')
        table_percent = ax_percent.table(cellText=new_df.values, colLabels=new_df.columns, loc='center')
        table_percent.auto_set_font_size(False)
        table_percent.set_fontsize(10)
        table_percent.scale(1.2, 1.2) # Adjust scale for better readability
        ax_percent.set_title("Percentage of Messages by User (Overall Group)", fontsize=14)
        plt.tight_layout()
        pdf_pages.savefig(fig_percent)
        plt.close(fig_percent)
    else:
        print("No data to display for message percentage by user.")


# --- 6. Generate and Save Word Cloud ---
df_wc = create_wordcloud(selected_user, df)
if df_wc:
    fig_wc, ax_wc = plt.subplots(figsize=(10, 8))
    ax_wc.imshow(df_wc, interpolation='bilinear')
    ax_wc.axis('off')
    ax_wc.set_title(f"Word Cloud for {selected_user}")
    pdf_pages.savefig(fig_wc)
    plt.close(fig_wc)
else:
    print(f"Could not generate word cloud for {selected_user}. Check if 'stop_hinglish.txt' exists and enough message data is ava


# --- 7. Generate and Save Most Common Words ---
most_common_df = most_common_words(selected_user, df)
if not most_common_df.empty:
    fig_common_words, ax_common_words = plt.subplots(figsize=(12, 8))
    # Horizontal bar chart: most common word at the top
    ax_common_words.barh(most_common_df[0], most_common_df[1], color='skyblue')
    ax_common_words.invert_yaxis()
    plt.xticks(rotation='vertical')
    ax_common_words.set_title(f'Most Common Words for {selected_user}')
    ax_common_words.set_xlabel("Count")
    ax_common_words.set_ylabel("Word")
    plt.tight_layout()
    pdf_pages.savefig(fig_common_words)
    plt.close(fig_common_words)
else:
    print(f"No data to display for most common words for {selected_user}.")


# --- 8. Generate and Save Emoji Analysis ---
emoji_df = emoji_helper(selected_user, df)
if not emoji_df.empty:
    # Render top emojis table
    fig_emoji_table, ax_emoji_table = plt.subplots(figsize=(6, len(emoji_df) * 0.4))
    ax_emoji_table.axis('off')
    table_emoji = ax_emoji_table.table(cellText=emoji_df.head().values, colLabels=['Emoji', 'Count'], loc='center')
    table_emoji.auto_set_font_size(False)
    table_emoji.set_fontsize(10)
    table_emoji.scale(1.2, 1.2)
    ax_emoji_table.set_title(f"Top Emojis for {selected_user}", fontsize=14)
    plt.tight_layout()
    pdf_pages.savefig(fig_emoji_table)
    plt.close(fig_emoji_table)

    # Pie chart for top emojis (if enough emojis to plot)
    if len(emoji_df) > 0:
        # Limit to top 5 for better visualization in pie chart
        top_5_emojis = emoji_df.head(5)
```

```
        fig_emoji_pie, ax_emoji_pie = plt.subplots(figsize=(8, 8))
        ax_emoji_pie.pie(top_5_emojis[1], labels=top_5_emojis[0], autopct="%0.2f%%", startangle=90)
        ax_emoji_pie.set_title(f"Emoji Distribution (Top {len(top_5_emojis)}) for {selected_user}")
        ax_emoji_pie.axis('equal') # Equal aspect ratio ensures that pie is drawn as a circle.
        plt.tight_layout()
        pdf_pages.savefig(fig_emoji_pie)
        plt.close(fig_emoji_pie)
    else:
        print(f"Not enough emojis to display in pie chart for {selected_user}.")
else:
    print(f"No emojis found for analysis for {selected_user}.")


# --- Finalize PDF ---
pdf_pages.close()
print(f"\nWhatsApp chat analysis report for '{selected_user}' has been generated and saved as 'whatsapp_chat_analysis_report.pdf'
```

Prepare PDF Document

Subtask:

Initialize a single PDF document for saving all the generated plots and tables.

Reasoning: To initialize a single PDF document for saving all the generated plots and tables, I need to import the `PdfPages` class from `matplotlib.backends.backend_pdf` and create an instance of it with the specified filename 'whatsapp_chat_analysis_report.pdf'.

```
from matplotlib.backends.backend_pdf import PdfPages

pdf_pages = PdfPages('whatsapp_chat_analysis_report.pdf')
print("PDF document 'whatsapp_chat_analysis_report.pdf' initialized.")
```

```
PDF document 'whatsapp_chat_analysis_report.pdf' initialized.
```

```
Start coding or generate with AI.
```

Reasoning: The PDF document has been initialized. The next logical step, as per the overall plan, is to calculate the top statistics (total messages, words, media shared, and links shared) using the `fetch_stats` function and then render these statistics as text onto a Matplotlib figure, which will then be saved to the initialized PDF.

```
import matplotlib.pyplot as plt

# Assuming selected_user and df are available from previous cells
num_messages, words, num_media_messages, num_links = fetch_stats(selected_user, df)

fig_stats, ax_stats = plt.subplots(figsize=(8, 6))
ax_stats.axis('off') # Hide axes for text display

stats_text = f"""
WhatsApp Chat Analysis Report for: {selected_user}

Total Messages: {num_messages}
Total Words: {words}
Media Shared: {num_media_messages}
Links Shared: {num_links}
"""
ax_stats.text(0.1, 0.9, stats_text, fontsize=12, verticalalignment='top',
              bbox=dict(boxstyle='round,pad=0.5', fc='wheat', alpha=0.5))
ax_stats.set_title(f'Top Statistics for {selected_user}', fontsize=14)
pdf_pages.savefig(fig_stats)
plt.close(fig_stats)

print(f"Top statistics for '{selected_user}' saved to PDF.")
```

```
Top statistics for 'Overall' saved to PDF.
```

```
Start coding or generate with AI.
```

Reasoning: With the top statistics added to the PDF, the next step, according to the plan, is to generate and save the monthly timeline plot. This involves calling the `monthly_timeline` function, creating a Matplotlib plot from the returned data, saving it to the `pdf_pages` object, and then closing the plot to free up memory.

```
import matplotlib.pyplot as plt

# Generate and Save Monthly Timeline
timeline = monthly_timeline(selected_user, df)
if not timeline.empty:
    fig_monthly, ax_monthly = plt.subplots(figsize=(10, 6))
    ax_monthly.plot(timeline['time'], timeline['message'], color='green')
    plt.xticks(rotation='vertical')
    ax_monthly.set_title(f"Monthly Timeline for {selected_user}")
    ax_monthly.set_xlabel("Month-Year")
    ax_monthly.set_ylabel("Number of Messages")
    plt.tight_layout()
    pdf_pages.savefig(fig_monthly)
    plt.close(fig_monthly)
    print(f"Monthly timeline for '{selected_user}' saved to PDF.")
else:
    print(f"No data to display for monthly timeline for '{selected_user}'.")
```

```
Monthly timeline for 'Overall' saved to PDF.
```

```
Start coding or generate with AI.
```

Reasoning: Following the generation of the monthly timeline, the next logical step is to generate and save the daily timeline plot to the PDF, as outlined in the overall plan. This involves calling the `daily_timeline` function, creating a Matplotlib plot from the returned data, saving it to the `pdf_pages` object, and then closing the plot to free up memory.

```
import matplotlib.pyplot as plt

# Generate and Save Daily Timeline
daily_timeline_df = daily_timeline(selected_user, df)
if not daily_timeline_df.empty:
    fig_daily, ax_daily = plt.subplots(figsize=(10, 6))
    ax_daily.plot(daily_timeline_df['only_date'], daily_timeline_df['message'], color='black')
    plt.xticks(rotation='vertical')
    ax_daily.set_title(f"Daily Timeline for {selected_user}")
    ax_daily.set_xlabel("Date")
    ax_daily.set_ylabel("Number of Messages")
    plt.tight_layout()
    pdf_pages.savefig(fig_daily)
    plt.close(fig_daily)
    print(f"Daily timeline for '{selected_user}' saved to PDF.")
else:
    print(f"No data to display for daily timeline for '{selected_user}'.")
```

```
Daily timeline for 'Overall' saved to PDF.
```

```
Start coding or generate with AI.
```

Reasoning: With the daily timeline added to the PDF, the next logical step, as per the overall plan, is to generate and save the 'Most busy day' activity map. This involves calling the `week_activity_map` function, creating a Matplotlib bar chart from the returned data, saving it to the `pdf_pages` object, and then closing the plot to free up memory.

```
import matplotlib.pyplot as plt

# Most busy day
busy_day = week_activity_map(selected_user, df)
if not busy_day.empty:
    fig_busy_day, ax_busy_day = plt.subplots(figsize=(8, 5))
    ax_busy_day.bar(busy_day.index, busy_day.values, color='purple')
    plt.xticks(rotation='vertical')
    ax_busy_day.set_title(f"Most Busy Day for {selected_user}")
    ax_busy_day.set_xlabel("Day of Week")
    ax_busy_day.set_ylabel("Number of Messages")
```