

Import the necessary libraries

```
In [1]: import re
import datetime
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from wordcloud import WordCloud, STOPWORDS
import emoji
import itertools
from collections import Counter
import warnings

%matplotlib inline
warnings.filterwarnings('ignore')
```

Preparation and reading data

```

In [2]: def rawToDf(file, key):
        '''Converts raw .txt file into a Data Frame'''

        split_formats = {
            '12hr' : '\d{1,2}/\d{1,2}/\d{2,4},\s\d{1,2}:\d{2}\s[APap][mM]\s-\s',
            '24hr' : '\d{1,2}/\d{1,2}/\d{2,4},\s\d{1,2}:\d{2}\s-\s',
            'custom' : ''
        }
        datetime_formats = {
            '12hr' : '%d/%m/%Y, %I:%M %p - ',
            '24hr' : '%d/%m/%Y, %H:%M - ',
            'custom': ''
        }

        with open(file, 'r', encoding='utf-8') as raw_data:
            # print(raw_data.read())
            raw_string = ' '.join(raw_data.read().split('\n')) # converting the list
            user_msg = re.split(split_formats[key], raw_string) [1:] # splits at all
            date_time = re.findall(split_formats[key], raw_string) # finds all the d

            df = pd.DataFrame({'date_time': date_time, 'user_msg': user_msg}) # expon

        # converting date-time pattern which is of type String to type datetime,
        # format is to be specified for the whole string where the placeholders are e
        df['date_time'] = pd.to_datetime(df['date_time'], format=datetime_formats[key]

        # split user and msg
        usernames = []
        msgs = []
        for i in df['user_msg']:
            a = re.split('([\w\W]+?):\s', i) # lazy pattern match to first {user_name
            if(a[1:]): # user typed messages
                usernames.append(a[1])
                msgs.append(a[2])
            else: # other notifications in the group(eg: someone was added, some left
                usernames.append("group_notification")
                msgs.append(a[0])

        # creating new columns
        df['user'] = usernames
        df['message'] = msgs

        # dropping the old user_msg col.
        df.drop('user_msg', axis=1, inplace=True)

        return df

```

Since WhatsApp texts are multi-line, you cannot just read the file line by line and get each message that you want. Instead, you need a way to identify if a line is a new message or part of an old message. You could use regular expressions.

While reading each line, I split it based on a comma and take the first item returned from the split() function. If the line is a new message, the first item would be a valid date, and it will be appended as a new message to the list of messages. If it's not, the message is part of the previous message, and hence, will be appended to the end of the previous message as one continuous message.

```
In [3]: df = rawToDf('whatsapp-chat-data.txt', '12hr')
```

Pre-Processing

The dataset contains 3 rows, and 13655 respective entries.

```
In [4]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13655 entries, 0 to 13654
Data columns (total 3 columns):
#   Column      Non-Null Count  Dtype
---  -
0   date_time   13655 non-null  datetime64[ns]
1   user        13655 non-null  object
2   message     13655 non-null  object
dtypes: datetime64[ns](1), object(2)
memory usage: 320.2+ KB
```

```
In [5]: df.sample(10)
```

```
Out[5]:
```

	date_time	user	message
2006	2020-02-24 16:56:00	Harsh Kapadia (TSEC IT, SE)	Hey is this only for CP?
9744	2020-08-11 21:50:00	Tanay Kamath (TSEC, CS)	Oh
5481	2020-04-29 21:32:00	Saurav Upoor (TSEC CS, SE)	Then compilation errors xp
12748	2020-09-20 15:54:00	Tanay Kamath (TSEC, CS)	But try not to send stuff related to controver...
3815	2020-03-24 20:49:00	Dheeraj Lalwani (TSEC, CS)	Yas
13169	2020-09-27 14:40:00	+91 81696 22410	🔥🔥 Thanks for sharing this
1790	2020-02-23 09:03:00	+91 93243 92133	Message copy paste karna
1957	2020-02-23 19:40:00	Dheeraj Lalwani (TSEC, CS)	we can record those lecs and create a youtube ...
2687	2020-03-05 00:55:00	Kartik Soneji (TSEC, CS)	This message was deleted
3220	2020-03-15 21:37:00	+91 88284 70904	She's from Bulgaria ig

An important observation:- One might wonder there are no NaNs, but as you can see, there are some rows, where messages are an empty string. This can be because, the message might just contain emojis. To see, how many such rows are there, we can use-

```
In [6]: df[df['message'] == ""].shape[0]
```

```
Out[6]: 538
```

Adding extra helper columns for analysis and visualization

```
In [7]: df['day'] = df['date_time'].dt.strftime('%a')
df['month'] = df['date_time'].dt.strftime('%b')
df['year'] = df['date_time'].dt.year
df['date'] = df['date_time'].apply(lambda x: x.date())
```

Now that we have a clean DataFrame to work with, it's time to perform analysis on it.

Final Data Frame

```
In [8]: df
```

```
Out[8]:
```

	date_time	user	message	day	month	year	date
0	2020-01-26 16:19:00	group_notification	Messages and calls are end-to-end encrypted. N...	Sun	Jan	2020	2020-01-26
1	2020-01-24 20:25:00	group_notification	Tanay Kamath (TSEC, CS) created group "CODERS🤖...	Fri	Jan	2020	2020-01-24
2	2020-01-26 16:19:00	group_notification	You joined using this group's invite link	Sun	Jan	2020	2020-01-26
3	2020-01-26 16:20:00	group_notification	+91 99871 38558 joined using this group's invi...	Sun	Jan	2020	2020-01-26
4	2020-01-26 16:20:00	group_notification	+91 91680 38866 joined using this group's invi...	Sun	Jan	2020	2020-01-26
...
13650	2020-10-02 02:05:00	Darshan Rander (TSEC, IT)	MCQs mark kiya	Fri	Oct	2020	2020-10-02
13651	2020-10-02 02:05:00	Darshan Rander (TSEC, IT)	Sign-in kiya😄😄	Fri	Oct	2020	2020-10-02
13652	2020-10-02 02:11:00	Tanay Kamath (TSEC, CS)	Incognito se na?	Fri	Oct	2020	2020-10-02
13653	2020-10-02 02:28:00	Darshan Rander (TSEC, IT)	Yup	Fri	Oct	2020	2020-10-02
13654	2020-10-02 10:13:00	Dheeraj Lalwani (TSEC, CS)	guys, please do me a favor and vote in this po...	Fri	Oct	2020	2020-10-02

13655 rows × 7 columns

Exploratory Data Analysis(EDA)

1. Overall frequency of total messages on the group.

2. Top 10 most active days.

3. Top 10 active users on the group (with a twist).

4. Top 10 users most sent media.

5. Most active hours and days.

6. Most used words - WordCloud

Heatmaps of weekdays and months.

Most active hours, weekdays, and months.

Silent members in the group.

1. Overall frequency of total messages on the group.

We will perform this to get a good look on the data distribution over the period of time. I will plot a simple line graph to see the frequency of messages over the months.

```
In [9]: df1 = df.copy()           # I will be using a copy of the original data frame everything
df1['message_count'] = [1] * df1.shape[0] # adding extra helper column --> n
df1.drop(columns='year', inplace=True)    # dropping unnecessary columns, us
df1 = df1.groupby('date').sum().reset_index() # grouping by date; since plot is
df1
```

Out[9]:

	date	message_count
0	2020-01-24	1
1	2020-01-26	105
2	2020-01-27	90
3	2020-01-28	126
4	2020-01-29	118
...
237	2020-09-28	144
238	2020-09-29	49
239	2020-09-30	167
240	2020-10-01	91
241	2020-10-02	22

242 rows × 2 columns

```
In [10]: # Improving Default Styles using Seaborn
sns.set_style("darkgrid")

# For better readability;
import matplotlib
matplotlib.rcParams['font.size'] = 20
matplotlib.rcParams['figure.figsize'] = (27, 6)      # Same as `plt.figure(figsize=(27, 6))`

# A basic plot
plt.plot(df1.date, df1.message_count)
plt.title('Messages sent per day over a time period');

# Could have used Seaborn's Lineplot as well.
# sns.lineplot(df1.date, df1.message_count);

# Saving the plots
plt.savefig('msg_plots.svg', format = 'svg')
```



2. Top 10 most active days.

```
In [11]: top10days = df1.sort_values(by="message_count", ascending=False).head(10) # Sort by message count
top10days.reset_index(inplace=True) # reset index in order.
top10days.drop(columns="index", inplace=True) # dropping original indices.
top10days
```

```
Out[11]:
```

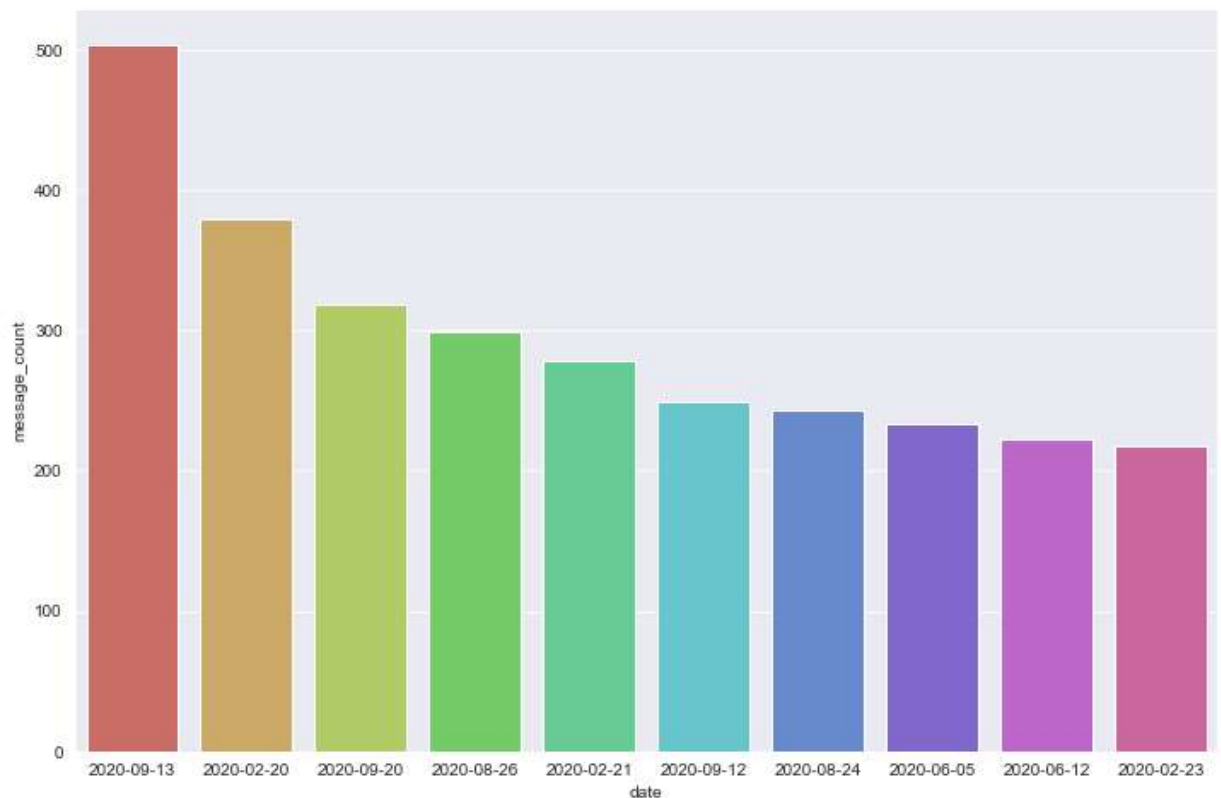
	date	message_count
0	2020-09-13	504
1	2020-02-20	379
2	2020-09-20	319
3	2020-08-26	299
4	2020-02-21	278
5	2020-09-12	249
6	2020-08-24	243
7	2020-06-05	233
8	2020-06-12	223
9	2020-02-23	218


```
In [12]: # Improving Default Styles using Seaborn
sns.set_style("darkgrid")

# For better readability;
import matplotlib
matplotlib.rcParams['font.size'] = 10
matplotlib.rcParams['figure.figsize'] = (12, 8)

# A bar plot for top 10 days
sns.barplot(top10days.date, top10days.message_count, palette="hls");

# Saving the plots
plt.savefig('top10_days.svg', format = 'svg')
```



From the graph above, we conclude that the group was very active on 25th Jan'22

3. Top 10 active users on the group.

Before, analysing that, we will see the number of silent members in the group.

```
In [13]: # Total number of people who have sent at least one message on the group;
print(f"Total number of people who have sent at least one message on the group are")

print(f"Number of people who haven't sent even a single message on the group are")
```

Total number of people who have sent at least one message on the group are 154
 Number of people who haven't sent even a single message on the group are -88

So we can see that 81 people haven't sent a single message in the group.

Now, pre-processing top 10 active users.

```
In [14]: df2 = df.copy()
df2 = df2[df2.user != "group_notification"]
top10df = df2.groupby("user")["message"].count().sort_values(ascending=False)

# Final Data Frame
top10df = top10df.head(10).reset_index()
top10df
```

Out[14]:

	user	message
0	Tanay Kamath (TSEC, CS)	2528
1	Dheeraj Lalwani (TSEC, CS)	1937
2	Darshan Rander (TSEC, IT)	1404
3	Kartik Soneji (TSEC, CS)	841
4	Harsh Kapadia (TSEC IT, SE)	790
5	Pratik K (TSEC CS, SE)	781
6	Saurav Upoor (TSEC CS, SE)	569
7	Tushar Nankani	354
8	+91 82916 21138	275
9	Farhan Irani (TSEC IT, SE)	255

```
In [15]: top10df['initials'] = ''
for i in range(10):
    top10df.initials[i] = top10df.user[i].split()[0][0] + top10df.user[i].split()[1][0]

top10df.initials[7] = "Me"      # That's me
top10df.initials[8] = "DT"
```

```
In [16]: # For better readability;
import matplotlib
matplotlib.rcParams['font.size'] = 14
matplotlib.rcParams['figure.figsize'] = (9, 5)
matplotlib.rcParams['figure.facecolor'] = '#00000000'
```

Now, I will be trying different visualization methods. Plotting a simple line graph.

```
In [17]: # Improving Default Styles using Seaborn
sns.set_style("whitegrid")

# Increasing the figure size
plt.figure(figsize=(12, 6))

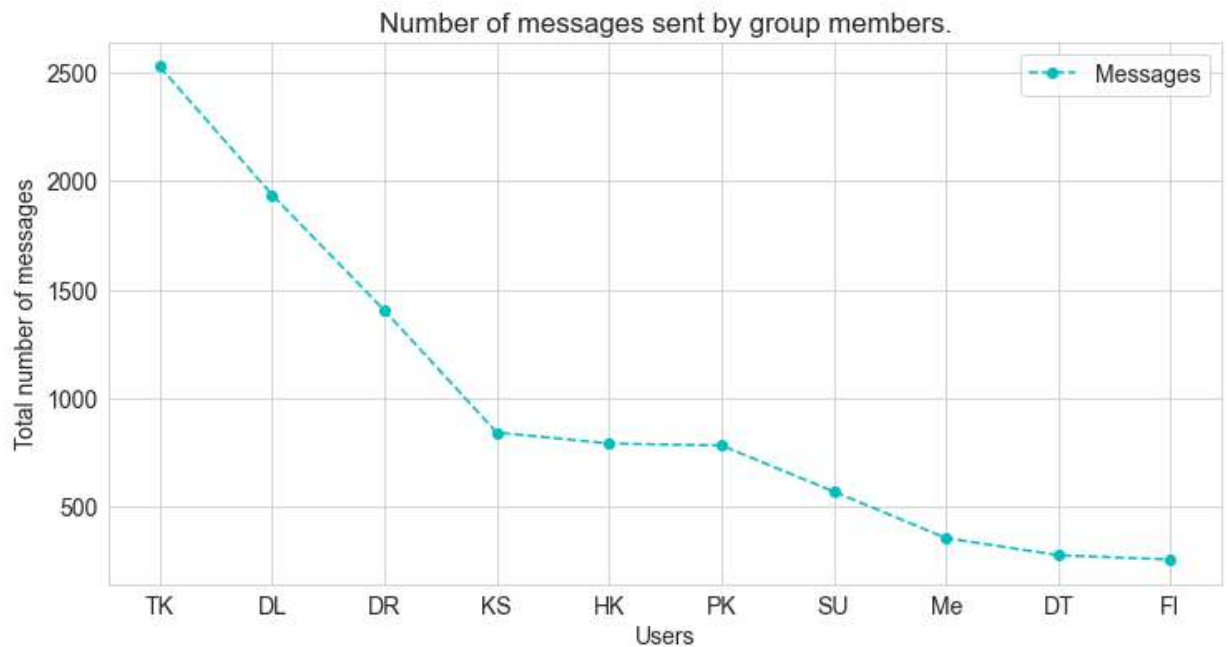
# plt.plot(top10df.initials, top10df.message, marker='o', ls='--', c='cyan')

# BETTER IMPLEMENTATION using the `fmt` argument;
plt.plot(top10df.initials, top10df.message, 'o--c')

# Labels and Title
plt.xlabel('Users')
plt.ylabel('Total number of messages')

plt.title("Number of messages sent by group members.")
plt.legend(['Messages']);

# Saving the plots
# plt.savefig('msg_plots.svg', format = 'svg')
```

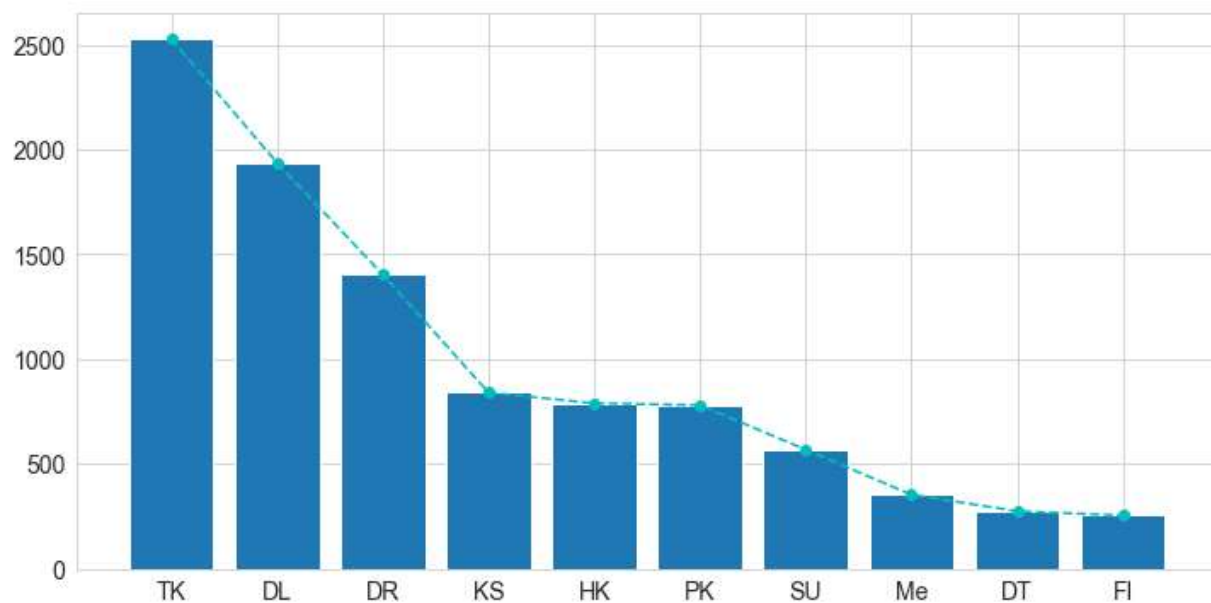


Plotting a bar chart and line graph together.

```
In [18]: # Improving Default Styles using Seaborn
sns.set_style("whitegrid")

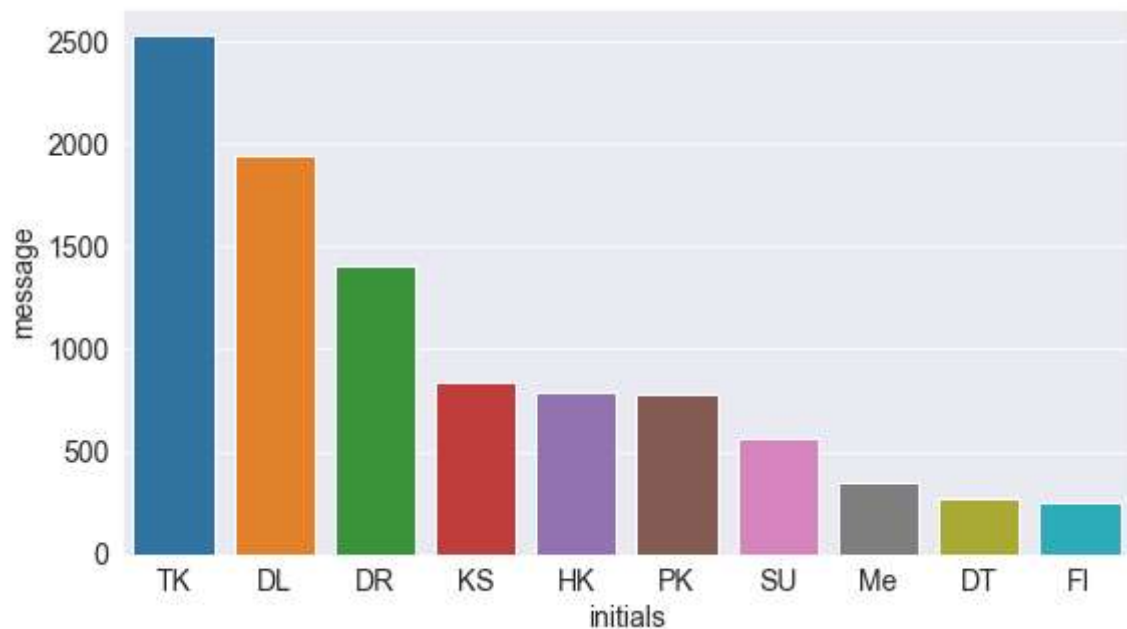
# Increasing the figure size
plt.figure(figsize=(12, 6))

plt.bar(top10df.initials, top10df.message) # basic bar chart
plt.plot(top10df.initials, top10df.message, 'o--c'); # line chart
```



Beautifying Default Styles using Seaborn

```
In [19]: # Beautifying Default Styles using Seaborn
sns.set_style("darkgrid")
sns.barplot(top10df.initials, top10df.message, data=top10df);
```



Important Note Since almost all the plots will be comparing one person with another, I'll assign a specific colour to each person so that it becomes easy to identify each person among multiple plots.

I could've used seaborn's color palette but:

Seaborn assigns default colors itself, but i wanted the color of a certain person remains the same, no matter the plot; Also, I wanted to try some different colors so I grabbed my colour palette from <https://coolours.co/> (<https://coolours.co/>). Next, I made a dictionary where each key is the name and the value for each would be their assigned colour. I create a function which reorders colours given a list of names to match the ordering of the plot. This function takes the ordered names as input and returns a reordered list of colours. This list has to be passed into the pallete argument in a seaborn plotting function.

Defining a function to tackle the problem. I'm defining the following function to maintain consistent colors for each person across all plots. Since the order will vary depending on the plot, this is passed to the function which will reorder colors in a particular order so that the color of a certain person remains the same no matter the plot. This will help maintain consistency and readability amongst the many graphs I will be plotting.

```
In [20]: def get_colors_of_certain_order(names_in_certain_order):
    '''the color of a certain person remains the same, no matter the plot'''

    order = list(names_in_certain_order)
    return_list = []

    for name in order:
        return_list.append(color_dict[name])

    return return_list
```

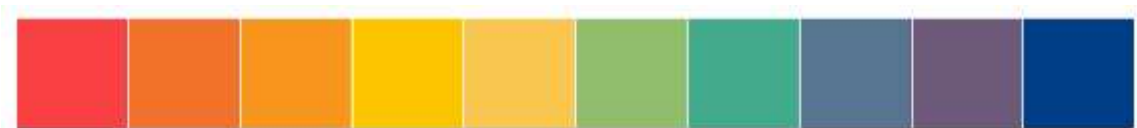
Now, we have a really nice set of colours for each person, with which we can visualize using sns.palplot.

```
In [21]: colors = ['#F94144', '#F3722C', '#F8961E', '#FDC500', '#F9C74F', '#90BE6D', '#43AA8B', '#577590', '#6D597A', '#003F88']
sns.palplot(colors)    # visualizing the colors' list

names = top10df.initials

color_dict = {}
for name, color in zip(names, colors):
    color_dict[name] = color
color_dict
```

```
Out[21]: {'TK': '#F94144',
'DL': '#F3722C',
'DR': '#F8961E',
'KS': '#FDC500',
'HK': '#F9C74F',
'PK': '#90BE6D',
'SU': '#43AA8B',
'Me': '#577590',
'DT': '#6D597A',
'FI': '#003F88'}
```



Now, finding the average message length of the 10 most active users of the group.

```
In [22]: # Adding another column for message length; using the apply method;
df2['message_length'] = df2['message'].apply(lambda x: len(x))

# Creating another dataframe for average length per user;
avg_msg_lengths = df2.groupby(df2.user).mean().reset_index().sort_values(by = 'me

# Creating helper columns;
top10df['avg_message_length'] = [0] * 10
i, j = 0, 0
while i < 10:
    if top10df['user'][i] == avg_msg_lengths['user'][j]:
        top10df['avg_message_length'][i] = avg_msg_lengths['message_length'][j]
        i += 1
        j = -1
    j += 1

# Sorting the average message lengths of the same to 10 active users;
top10df_msg = top10df.sort_values(by = "avg_message_length", ascending=False)
```

Now, we will be plotting most sent messages and respective average message lengths simultaneously, to see some interesting results.

Plotting multiple charts in a grid Matplotlib and Seaborn also support plotting multiple charts in a grid, using `plt.subplots`, which returns a set of axes that can be used for plotting.

```
In [23]: # plotting multiple charts in a grid
fig, axes = plt.subplots(1, 2, figsize=(16, 6))
sns.set_style("darkgrid")

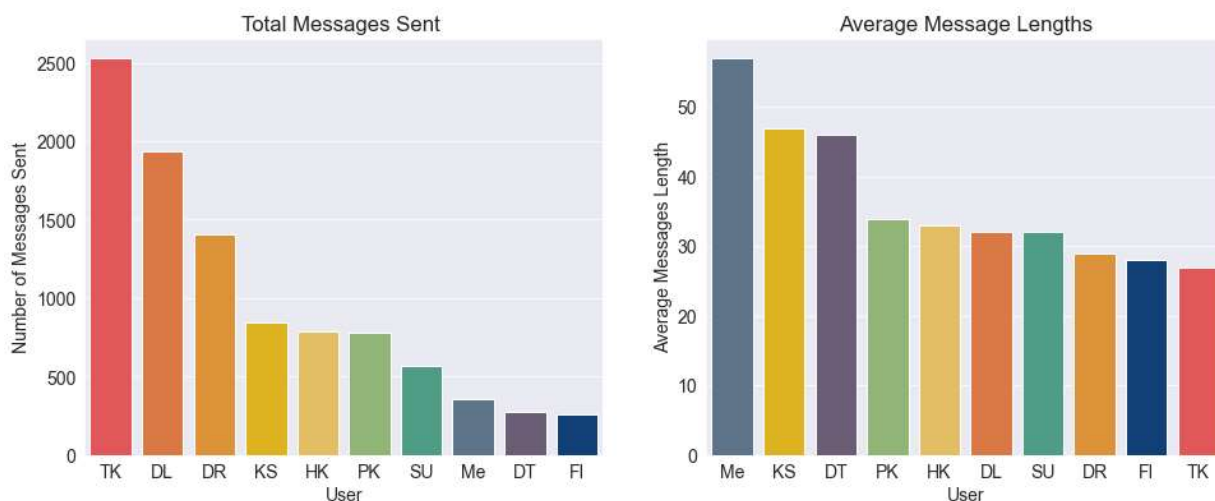
# Plot 1 - Countplot of total messages sent
sns.barplot(top10df.initials, top10df.message, data=top10df, ax = axes[0], palette=

axes[0].set_title('Total Messages Sent ')
axes[0].set_xlabel('User')
axes[0].set_ylabel('Number of Messages Sent')

# Plot 2 - Barplot of those top 10 users' average message lengths
sns.barplot(top10df_msg.initials, top10df_msg.avg_message_length, ax = axes[1], p

axes[1].set_title('Average Message Lengths')
axes[1].set_xlabel('User');
axes[1].set_ylabel('Average Messages Length');

# Saving the plots
plt.savefig('top10_msg_plots_diff.svg', format = 'svg')
```



6. Most active days, most active hours, most active months.

Pre-processing

```
In [24]: df3 = df.copy()
df3['message_count'] = [1] * df.shape[0] # helper column to keep a count.

df3['hour'] = df3['date_time'].apply(lambda x: x.hour)

grouped_by_time = df3.groupby('hour').sum().reset_index().sort_values(by = 'hour')
```

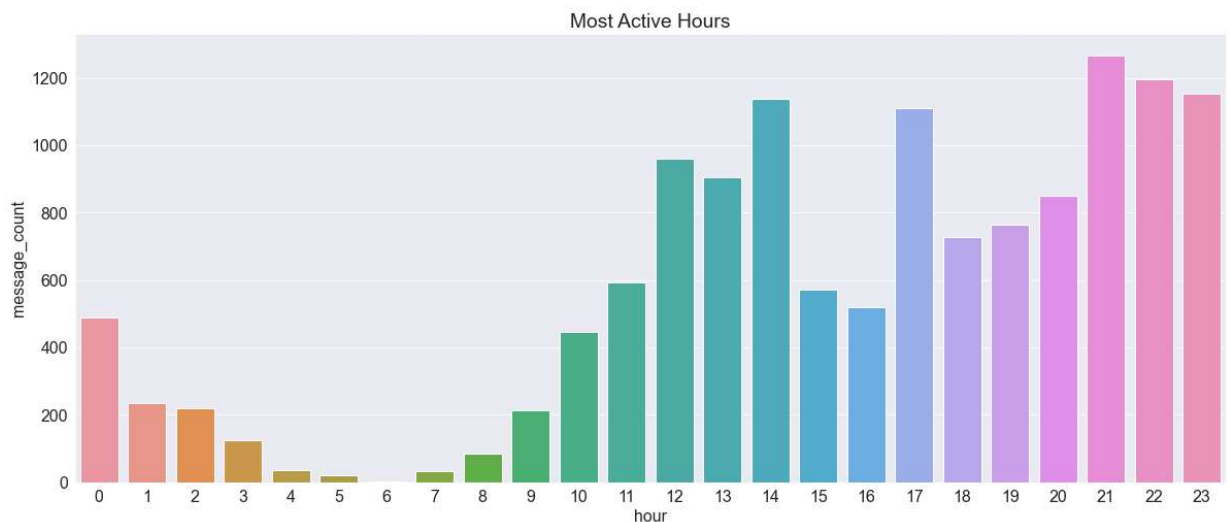


```
In [25]: # Better Readability
import matplotlib
matplotlib.rcParams['font.size'] = 16
matplotlib.rcParams['figure.figsize'] = (20, 8)

# Beautifying Default Styles using Seaborn
sns.set_style("darkgrid")

# PLOT: grouped by hour
sns.barplot(grouped_by_time.hour, grouped_by_time.message_count)
plt.title('Most Active Hours');

# Saving the plots;
plt.savefig('most_active_hours.svg', format = 'svg')
```



Pre-processing weekdays and months

```
In [26]: # specific `order` to be printed in;
days = ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun']
# grouping by day;
grouped_by_day = df3.groupby('day').sum().reset_index()[['day', 'message_count']]

# specific `order` to be printed in;
months = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep'] # till
# grouping by month;
grouped_by_month = df3.groupby('month').sum().reset_index()[['month', 'message_count']]
```

Visualization Now, we will be plotting grouped by day and respective group by month simultaneously, to see some interesting results. Using plt.subplots to plot multiple charts in a grid.

```
In [27]: fig, axs = plt.subplots(1, 2, figsize = (24, 6))

# Better Readability
import matplotlib
matplotlib.rcParams['font.size'] = 20

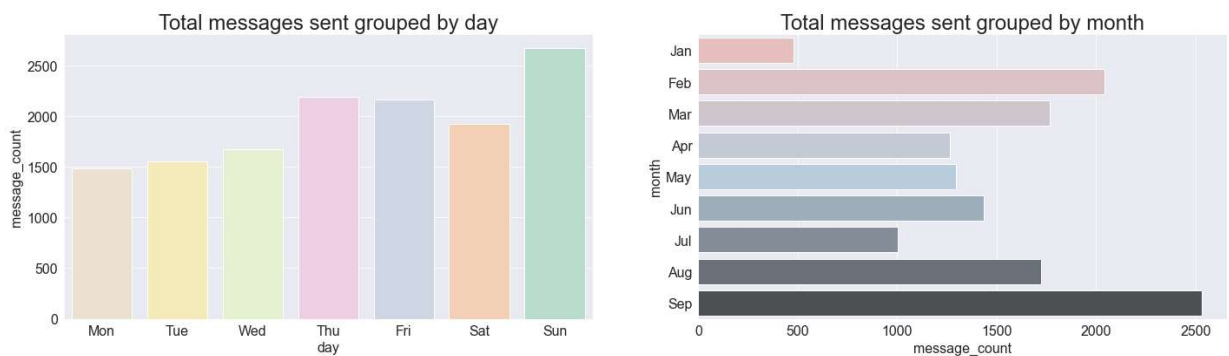
# Beautifying Default Styles using Seaborn
sns.set_style("darkgrid")

# Plotting;

# PLOT 1: Messages grouped by weekday
sns.barplot(grouped_by_day.day, grouped_by_day.message_count, order=days, ax = axs[0])
axs[0].set_title('Total messages sent grouped by day')

# PLOT 2: Messages grouped by months
sns.barplot(y = grouped_by_month.month, x=grouped_by_month.message_count, order = months, ax = axs[1])
axs[1].set_title('Total messages sent grouped by month');

# Saving the plots;
plt.savefig('days_and_month.svg', format = 'svg')
```



The group is most active on Sundays, and least active on Mondays.

Now, we will plot a heatmap, combining the above to bar plots, for a better understanding!

```

In [28]: # Better Readability
import matplotlib
matplotlib.rcParams['font.size'] = 14
matplotlib.rcParams['figure.figsize'] = (18, 6)

# Beautifying Default Styles using Seaborn,
sns.set_style("darkgrid")

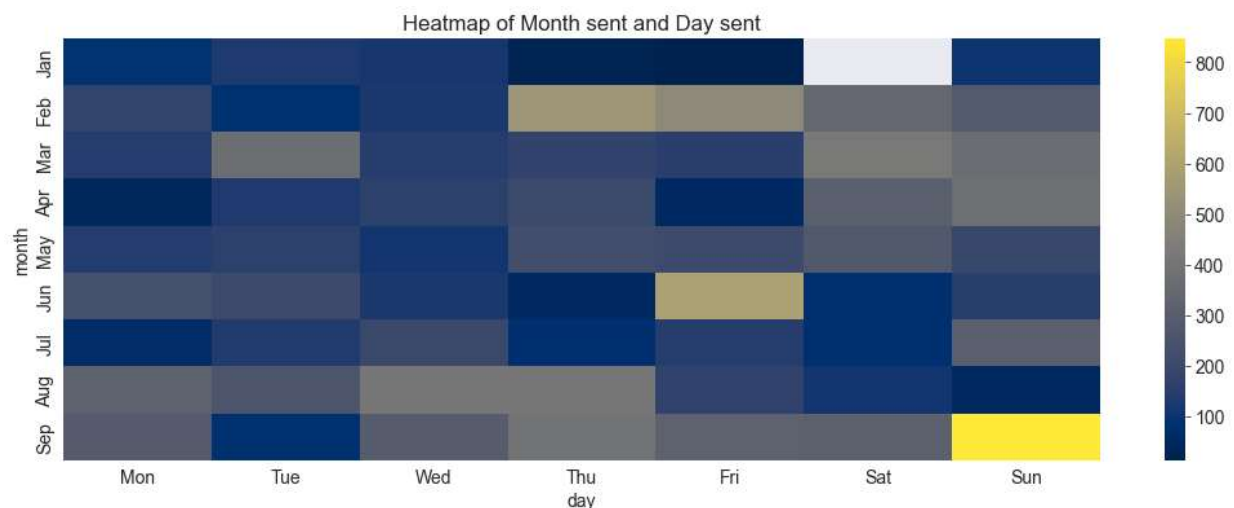
# Pre-Processing by month and day,
grouped_by_month_and_day = df3.groupby(['month', 'day']).sum().reset_index()[['month', 'day', 'count']]

# creating a pivot table,
pt = grouped_by_month_and_day.pivot_table(index = 'month', columns = 'day', values = 'count')

# PLOT: heatmap.
sns.heatmap(pt, cmap = 'cividis');
plt.title('Heatmap of Month sent and Day sent');

# Saving the plots;
plt.savefig('month_day_heatmap.svg', format = 'svg')

```



Inferences

The group is more active on weekends, throughout the months.

September has the most lighter blue shades and more yellow gradients.

This gives a combined analysis.

Most used words in the chat.

We will now decide the most used words in the group using WordCloud module in Python.

```
In [29]: comment_words = ' '

# stopwords --> Words to be avoided while forming the WordCloud,
# removed group_notifications like 'joined', 'deleted';
# removed really common words like "yeah" and "okay".
stopwords = STOPWORDS.update(['group', 'link', 'invite', 'joined', 'message', 'de

# iterate through the DataFrame.
for val in df3.message.values:

    # typecaste each val to string.
    val = str(val)

    # split the value.
    tokens = val.split()

    # Converts each token into lowercase.
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    for words in tokens:
        comment_words = comment_words + words + ' '

wordcloud = WordCloud(width = 600, height = 600,
                        background_color = 'white',
                        stopwords = stopwords,
                        min_font_size = 8).generate(comment_words)
```

```
wordcloud.to_image()
```

