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


df = pd.read_excel("/content/bike_ride_1.xlsx")
```

Double-click (or enter) to edit

```
df.shape
```

```
(696, 18)
```

```
df.head()
```

	ride_id	rideable_type	started_at	new_start_date	new_started_time	ended_at	new_end_at	new_ended_time	st
0	F6496DF223062E4D	electric_bike	13/06/2021 22:35	13-Jun-21	22:35:00	14/06/2021 00:14	14-Jun-21	00:14:00	I
1	2B218870CDC78BB2	classic_bike	05/06/2021 23:46	05-Jun-21	23:46:00	06/06/2021 00:10	06-Jun-21	00:10:00	L
2	067B43F67F5DF530	classic_bike	11/06/2021 23:30	11-Jun-21	23:30:00	12/06/2021 00:04	12-Jun-21	00:04:00	
3	E90B0906B0E6AE92	electric_bike	22/06/2021 23:38	22-Jun-21	23:38:00	23/06/2021 01:52	23-Jun-21	01:52:00	M
4	6A49443B53ED1E7D	classic_bike	12/06/2021 23:12	12-Jun-21	23:12:00	13/06/2021 01:12	13-Jun-21	01:12:00	D

```
df.duplicated().sum()
```

```
0
```

```
df.tail()
```

	ride_id	rideable_type	started_at	new_start_date	new_started_time	ended_at	new_end_at	new_ended_time
691	67AB517DE2340656	docked_bike	13/05/2022 15:58	13-May-22	15:58:00	14/05/2022 01:16	14-May-22	01:16:00
692	694D49CD4065F9CC	electric_bike	28/05/2022 23:49	28-May-22	23:49:00	29/05/2022 00:01	29-May-22	00:01:00
693	67F99CD7DCF6A5E1	docked_bike	11/05/2022 22:37	11-May-22	22:37:00	12/05/2022 09:42	12-May-22	09:42:00
694	CAA46E7729592B77	electric_bike	30/05/2022 22:34	30-May-22	22:34:00	31/05/2022 00:04	31-May-22	00:04:00
695	30D6E11912E8F436	electric_bike	23/05/2022 23:49	23-May-22	23:49:00	24/05/2022 00:04	24-May-22	00:04:00

```
# Convert the 'started_at' and 'ended_at' columns to datetime
df['started_at'] = pd.to_datetime(df['started_at'])
df['ended_at'] = pd.to_datetime(df['ended_at'])
```

```
<ipython-input-11-9be305e71226>:2: UserWarning: Parsing dates in %d/%m/%Y %H:%M format when dayfirst=False (the default)
  df['started_at'] = pd.to_datetime(df['started_at'])
<ipython-input-11-9be305e71226>:3: UserWarning: Parsing dates in %d/%m/%Y %H:%M format when dayfirst=False (the default)
  df['ended_at'] = pd.to_datetime(df['ended_at'])
```

```
# Calculate the trip duration
df['trip_duration'] = df['ended_at'] - df['started_at']

# Optionally convert trip duration to minutes
df['trip_duration_min'] = df['trip_duration'].dt.total_seconds() / 60

# Convert the 'trip_duration' column from string to timedelta
df['trip_duration'] = pd.to_timedelta(df['trip_duration'])

# Now convert timedelta to total minutes
```

```
df['trip_duration_min'] = df['trip_duration'].dt.total_seconds() / 60

# Check the result
print(df[['trip_duration', 'trip_duration_min']])
```

```
      trip_duration  trip_duration_min
0   0 days 01:39:00               99.0
1   0 days 00:24:00               24.0
2   0 days 00:34:00               34.0
3   0 days 02:14:00              134.0
4   0 days 02:00:00              120.0
..              ...                ...
691 0 days 09:18:00              558.0
692 0 days 00:12:00               12.0
693 0 days 11:05:00              665.0
694 0 days 01:30:00               90.0
695 0 days 00:15:00               15.0

[696 rows x 2 columns]
```

Start coding or generate with AI.

```
# Show the updated DataFrame
print(df[['ride_id', 'started_at', 'ended_at', 'trip_duration', 'trip_duration_min']].head())
```

```
            ride_id          started_at            ended_at   trip_duration  \
0  F6496DF223062E4D 2021-06-13 22:35:00 2021-06-14 00:14:00 0 days 01:39:00
1  2B218870CDC78BB2 2021-06-05 23:46:00 2021-06-06 00:10:00 0 days 00:24:00
2  067B43F67F5DF530 2021-06-11 23:30:00 2021-06-12 00:04:00 0 days 00:34:00
3  E90B0906B0E6AE92 2021-06-22 23:38:00 2021-06-23 01:52:00 0 days 02:14:00
4  6A49443B53ED1E7D 2021-06-12 23:12:00 2021-06-13 01:12:00 0 days 02:00:00

   trip_duration_min
0               99.0
1               24.0
2               34.0
3              134.0
4              120.0
```

```
df.head()
```

	ride_id	rideable_type	started_at	new_start_date	new_started_time	ended_at	new_end_at	new_ended_time	st
0	F6496DF223062E4D	electric_bike	2021-06-13 22:35:00	13-Jun-21	22:35:00	2021-06-14 00:14:00	14-Jun-21	00:14:00	D
1	2B218870CDC78BB2	classic_bike	2021-06-05 23:46:00	05-Jun-21	23:46:00	2021-06-06 00:10:00	06-Jun-21	00:10:00	La
2	067B43F67F5DF530	classic_bike	2021-06-11 23:30:00	11-Jun-21	23:30:00	2021-06-12 00:04:00	12-Jun-21	00:04:00	
3	E90B0906B0E6AE92	electric_bike	2021-06-22 23:38:00	22-Jun-21	23:38:00	2021-06-23 01:52:00	23-Jun-21	01:52:00	M
4	6A49443B53ED1E7D	classic_bike	2021-06-12 23:12:00	12-Jun-21	23:12:00	2021-06-13 01:12:00	13-Jun-21	01:12:00	D

```
df.isna().sum()
```

	0
ride_id	0
rideable_type	0
started_at	0
new_start_date	0
new_started_time	0
ended_at	0
new_end_at	0
new_ended_time	0
start_station_name	0
start_station_id	0
end_station_name	0
end_station_id	0
start_lat	0
start_lng	0
end_lat	0
end_lng	0
member_casual	0
week_day	0
trip_duration	0
trip_duration_min	0

```
# List of columns to drop
columns_to_drop = ['trip_duration'] # removed 'started_at' from the list

# Drop the specified columns from the DataFrame
df.drop(columns=columns_to_drop, axis=1, inplace=True)

# Verify the changes by displaying the first few rows of the DataFrame
df.head()

# calcularing peak period
# Extract the hour from the 'started_at' column

df['hour'] = df['started_at'].dt.hour


# calcularing peak period
# Extract the hour from the 'started_at' column

df['hour'] = df['started_at'].dt.hour
df.hour.value_counts()
```

hour	count
23	465
22	65
21	38
17	19
20	18
19	18
18	17
15	8
16	7
12	7
11	5
13	5
14	4
4	3
8	3
10	3
1	2
7	2
9	2
2	2
0	1
3	1
6	1

Start coding or generate with AI.

```
# Group by the hour and count the number of rides
ride_count_hour = df.groupby('hour').size()


# Find the peak period (hour with the maximum number of rides)
peak_period = ride_count_hour.idxmax()
peak_rides = ride_count_hour.max()
print(f"The peak period is hour {peak_period} with {peak_rides} rides.")
```

```
The peak period is hour 23 with 465 rides.
```

```
# Instead of grouping by hour, you could group by 15-minute, 30-minute,
df['time_window'] = df['started_at'].dt.floor('30T')  # 30-minute windows
ride_counts_by_window = df.groupby('time_window').size()
df.time_window.value_counts()
```

```
<ipython-input-20-2d1119ee4782>:2: FutureWarning: 'T' is deprecated and will be removed in a future version, please use
  df['time_window'] = df['started_at'].dt.floor('30T')  # 30-minute windows
```

time_window	count
2021-06-12 23:30:00	29
2021-06-18 23:30:00	23
2021-06-25 23:30:00	22
2021-06-05 23:30:00	20
2021-06-19 23:30:00	19
...	...
2021-06-03 19:30:00	1
2021-06-13 04:00:00	1
2021-07-14 12:30:00	1
2021-06-27 22:30:00	1
2022-05-23 23:30:00	1

357 rows × 1 columns

```
# The station with most trips, Count trips by start station
# Count trips by start station

start_station_counts = df.groupby('start_station_name').size().reset_index(name='trip_count_start')
df.start_station_name.value_counts()
```

start_station_name	count
Lakefront Trail & Bryn Mawr Ave	2
Dearborn St & Monroe St	1
Halsted St & Archer Ave	1
Franklin St & Adams St (Temp)	1
Greenwood Ave & 47th St	1
...	...
Rockwell St & Eastwood Ave	1
Wells St & Concord Ln	1
Southport Ave & Wrightwood Ave	1
Milwaukee Ave & Wabansia Ave	1
Harlem & Irving Park	1

695 rows × 1 columns

```
# Count trips by end station

end_station_counts = df.groupby('end_station_name').size().reset_index(name='trip_count_end')
df.end_station_name.value_counts()
```

end_station_name	count
Base - 2132 W Hubbard Warehouse	7
Michigan Ave & 14th St	7
Halsted St & Roscoe St	6
Wilton Ave & Belmont Ave	6
Shore Dr & 55th St	6
...	...
Washtenaw Ave & Lawrence Ave	1
Larrabee St & Armitage Ave	1
Damen Ave & Pierce Ave	1
Wells St & Evergreen Ave	1
Orange & Addison	1

427 rows × 1 columns

```
# Combine both start and end counts to get the total trips for each station
# We merge the counts for start and end stations on station name, using outer join

total_station_counts = pd.merge(start_station_counts, end_station_counts,
                                left_on='start_station_name', right_on='end_station_name',
                                how='outer')
total_station_counts.value_counts()
```

start_station_name	trip_count_start	end_station_name	trip_count_end	count
2112 W Peterson Ave	1.0	2112 W Peterson Ave	1.0	1
Morgan St & Lake St	1.0	Morgan St & Lake St	1.0	1
Ogden Ave & Congress Pkwy	1.0	Ogden Ave & Congress Pkwy	1.0	1
Oakley Ave & Touhy Ave	1.0	Oakley Ave & Touhy Ave	2.0	1
Oak Park & Wellington	1.0	Oak Park & Wellington	1.0	1
...	...	...	...	...
Eastlake Ter & Rogers Ave	1.0	Eastlake Ter & Rogers Ave	1.0	1
DuSable Museum	1.0	DuSable Museum	1.0	1
Drake Ave & Addison St	1.0	Drake Ave & Addison St	1.0	1
Dorchester Ave & 49th St	1.0	Dorchester Ave & 49th St	1.0	1
Yates Blvd & 93rd St	1.0	Yates Blvd & 93rd St	1.0	1

412 rows × 1 columns

```
# Replace NaN values with 0 (for stations that are only start or end points)

total_station_counts.fillna(0, inplace=True)
df.head()
```

	ride_id	rideable_type	started_at	new_start_date	new_started_time	ended_at	new_end_at	new_ended_time	st
0	F6496DF223062E4D	electric_bike	2021-06-13 22:35:00	13-Jun-21	22:35:00	2021-06-14 00:14:00	14-Jun-21	00:14:00	D
1	2B218870CDC78BB2	classic_bike	2021-06-05 23:46:00	05-Jun-21	23:46:00	2021-06-06 00:10:00	06-Jun-21	00:10:00	La
2	067B43F67F5DF530	classic_bike	2021-06-11 23:30:00	11-Jun-21	23:30:00	2021-06-12 00:04:00	12-Jun-21	00:04:00	
3	E90B0906B0E6AE92	electric_bike	2021-06-22 23:38:00	22-Jun-21	23:38:00	2021-06-23 01:52:00	23-Jun-21	01:52:00	M
4	6A49443B53ED1E7D	classic_bike	2021-06-12 23:12:00	12-Jun-21	23:12:00	2021-06-13 01:12:00	13-Jun-21	01:12:00	D

5 rows × 21 columns

```
# Add the trip counts for both start and end trips

total_station_counts['total_trips'] = total_station_counts['trip_count_start'] + total_station_counts['trip_count_end']
total_station_counts.head()
```

	start_station_name	trip_count_start	end_station_name	trip_count_end	total_trips
0	2112 W Peterson Ave	1.0	2112 W Peterson Ave	1.0	2.0
1	63rd St Beach	1.0	0	0.0	1.0
2	900 W Harrison St	1.0	900 W Harrison St	1.0	2.0
3	Aberdeen St & Jackson Blvd	1.0	Aberdeen St & Jackson Blvd	1.0	2.0

```
# Find the station with the maximum trips

most_trips_station = total_station_counts.loc[total_station_counts['total_trips'].idxmax()]

# Print the result
print(f"Station with most trips: {most_trips_station['start_station_name']} with {most_trips_station['total_trips']} trips."
```

```
Station with most trips: Michigan Ave & 14th St with 8.0 trips.
```

```
df.head()
```

	ride_id	rideable_type	started_at	new_start_date	new_started_time	ended_at	new_end_at	new_ended_time	st
0	F6496DF223062E4D	electric_bike	2021-06-13 22:35:00	13-Jun-21	22:35:00	2021-06-14 00:14:00	14-Jun-21	00:14:00	D
1	2B218870CDC78BB2	classic_bike	2021-06-05 23:46:00	05-Jun-21	23:46:00	2021-06-06 00:10:00	06-Jun-21	00:10:00	La
2	067B43F67F5DF530	classic_bike	2021-06-11 23:30:00	11-Jun-21	23:30:00	2021-06-12 00:04:00	12-Jun-21	00:04:00	
3	E90B0906B0E6AE92	electric_bike	2021-06-22 23:38:00	22-Jun-21	23:38:00	2021-06-23 01:52:00	23-Jun-21	01:52:00	M
4	6A49443B53ED1E7D	classic_bike	2021-06-12 23:12:00	12-Jun-21	23:12:00	2021-06-13 01:12:00	13-Jun-21	01:12:00	D

5 rows × 21 columns

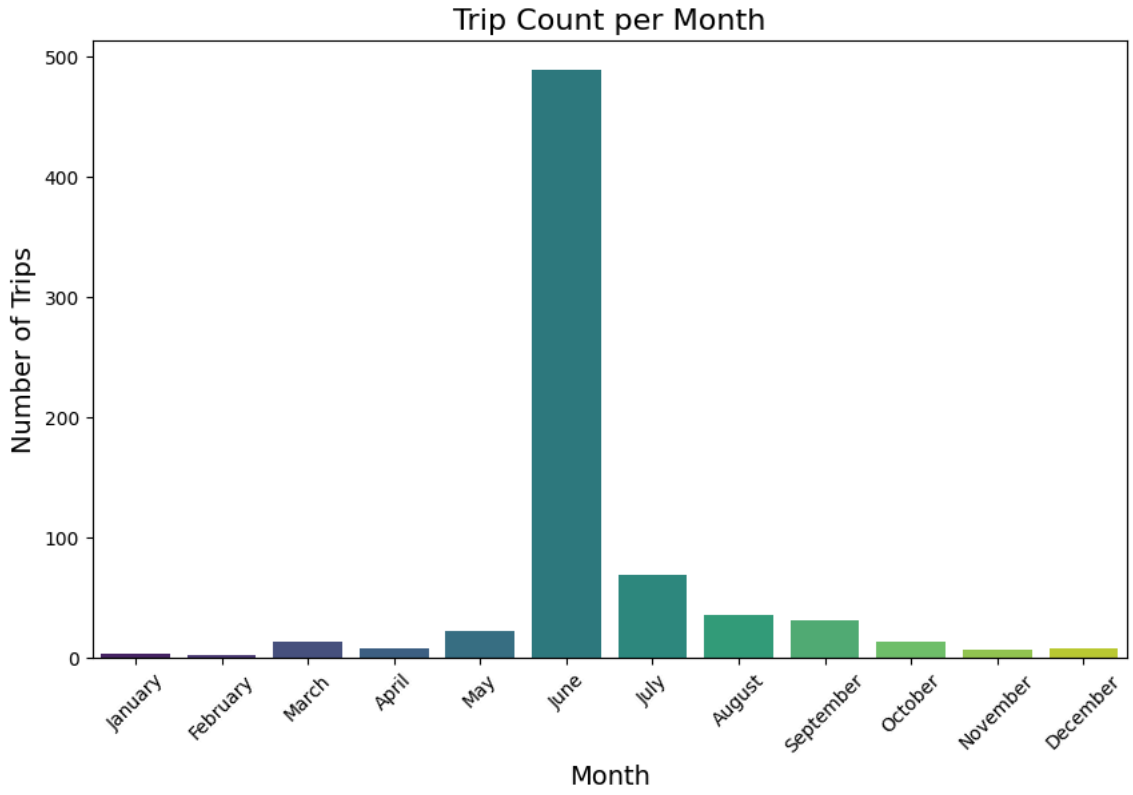
```
# Extract the month from the 'started_at' column
df['month'] = df['started_at'].dt.month

# Group by the month and count the number of trips
trips_by_month = df.groupby('month').size().reset_index(name='trip_count')
```

```
# Sort the months by the number of trips (descending order)
trips_by_month = trips_by_month.sort_values(by='trip_count', ascending=False)

# Print the DataFrame with months and their respective trip counts
print(trips_by_month)

# Find the month with the most trips
most_trips_month = trips_by_month.iloc[0]

print(f"The month with the most trips is: {most_trips_month['month']} with {most_trips_month['trip_count']} trips.")
```

```
    month  trip_count
5       6         489
6       7          68
7       8          35
8       9          31
4       5          22
2       3          13
9      10          13
3       4           7
11     12           7
10     11           6
0       1           3
1       2           2
The month with the most trips is: 6 with 489 trips.
```

```
# Extract month name from 'started_at' datetime column
df['month'] = df['started_at'].dt.month_name()

# Count the number of trips per month
monthly_trip_counts = df['month'].value_counts().reset_index()
monthly_trip_counts.columns = ['month', 'trip_count']

# Order the months to ensure the plot is in chronological order
month_order = ['January', 'February', 'March', 'April', 'May', 'June',
               'July', 'August', 'September', 'October', 'November', 'December']
monthly_trip_counts['month'] = pd.Categorical(monthly_trip_counts['month'], categories=month_order, ordered=True)

# Sort the DataFrame by month
monthly_trip_counts = monthly_trip_counts.sort_values('month')

# Plot the results using a bar plot
plt.figure(figsize=(10, 6))
sns.barplot(x='month', y='trip_count', data=monthly_trip_counts, palette='viridis')

# Add title and labels
plt.title('Trip Count per Month', fontsize=16)
plt.xlabel('Month', fontsize=14)
plt.ylabel('Number of Trips', fontsize=14)

# Display the plot with x-axis labels rotated for readability
plt.xticks(rotation=45)
plt.show()
```

```
<ipython-input-29-1eb45530b805>:18: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue`

  sns.barplot(x='month', y='trip_count', data=monthly_trip_counts, palette='viridis')
```



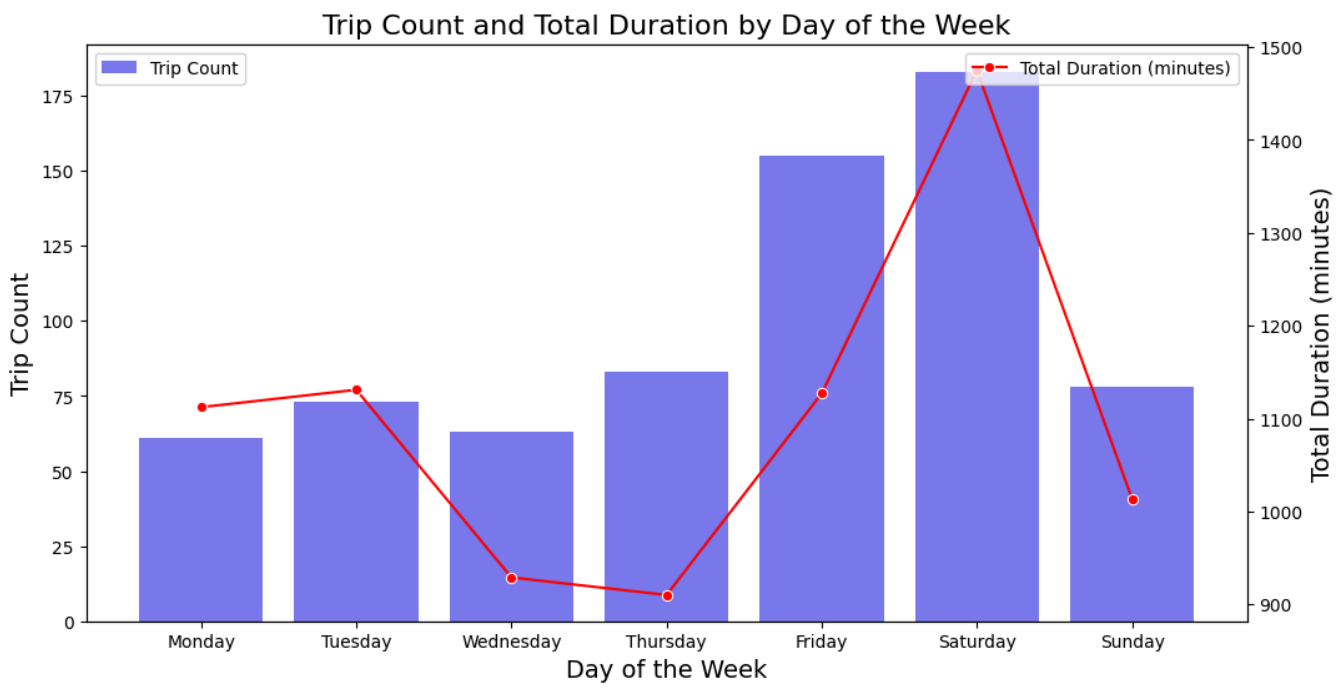
Start coding or generate with AI.

```
# Extract the day of the week from 'started_at' (0 = Monday, 6 = Sunday)
df['day_of_week'] = df['started_at'].dt.dayofweek

# Calculate the trip duration in minutes
df['trip_duration'] = (df['ended_at'] - df['started_at']).dt.total_seconds() / 60  # in minutes

# Group by day of the week and calculate total trips and total duration for each day
trips_by_day = df.groupby('day_of_week').agg(
    trip_count=('day_of_week', 'size'),
    total_duration=('trip_duration', 'sum')
).reset_index()

# Sort the result by trip count in descending order
trips_by_day = trips_by_day.sort_values(by='trip_count', ascending=False)

# Print the DataFrame showing trips and duration for each day
print(trips_by_day)

# Find the day of the week with the most trips
most_trips_day = trips_by_day.iloc[0]

# Create a mapping for more human-readable day names
day_name_mapping = {0: 'Monday', 1: 'Tuesday', 2: 'Wednesday', 3: 'Thursday', 4: 'Friday', 5: 'Saturday', 6: 'Sunday'}

# Print the result with the day name
print(f"The day of the week with the most trips is {day_name_mapping[most_trips_day['day_of_week']]} with {most_trips_day['t
```

```
   day_of_week  trip_count  total_duration
5            5         183         88499.0
4            4         155         67630.0
3            3          83         54605.0
6            6          78         60806.0
1            1          73         67862.0
2            2          63         55739.0
0            0          61         66738.0
The day of the week with the most trips is Saturday with 183.0 trips and a total duration of 88499.0 minutes.
```

Start coding or generate with AI.

```
# Extract day of the week
df['day_of_week'] = df['started_at'].dt.day_name()

# Step 2: Count the number of trips and calculate total duration by day of the week
# Assuming 'trip_duration' is in seconds
weekly_data = df.groupby('day_of_week').agg(
    trip_count=('ride_id', 'count'),
    total_duration=('trip_duration', 'sum') # Changed 'total_duration' to 'trip_duration'
).reset_index()

# Step 3: Convert total duration from seconds to minutes
weekly_data['total_duration'] = weekly_data['total_duration'] / 60  # Convert to minutes

# Step 4: Order the days of the week
day_order = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
weekly_data['day_of_week'] = pd.Categorical(weekly_data['day_of_week'], categories=day_order, ordered=True)
weekly_data = weekly_data.sort_values('day_of_week')

# Step 5: Plotting
fig, ax1 = plt.subplots(figsize=(12, 6))

# Create a bar plot for trip count
sns.barplot(x='day_of_week', y='trip_count', data=weekly_data, ax=ax1, color='b', alpha=0.6, label='Trip Count')
ax1.set_ylabel('Trip Count', fontsize=14)
ax1.set_xlabel('Day of the Week', fontsize=14)
ax1.tick_params(axis='y')

# Create a second y-axis to plot total duration
ax2 = ax1.twinx()

# Changed 'trip_duration' to 'total_duration' to reflect the new column name
sns.lineplot(x='day_of_week', y='total_duration', data=weekly_data, ax=ax2, color='r', marker='o', label='Total Duration (mi
ax2.set_ylabel('Total Duration (minutes)', fontsize=14)
ax2.tick_params(axis='y')

# Add titles and legends
plt.title('Trip Count and Total Duration by Day of the Week', fontsize=16)
ax1.legend(loc='upper left')
ax2.legend(loc='upper right')

plt.show()
```



Start coding or generate with AI.

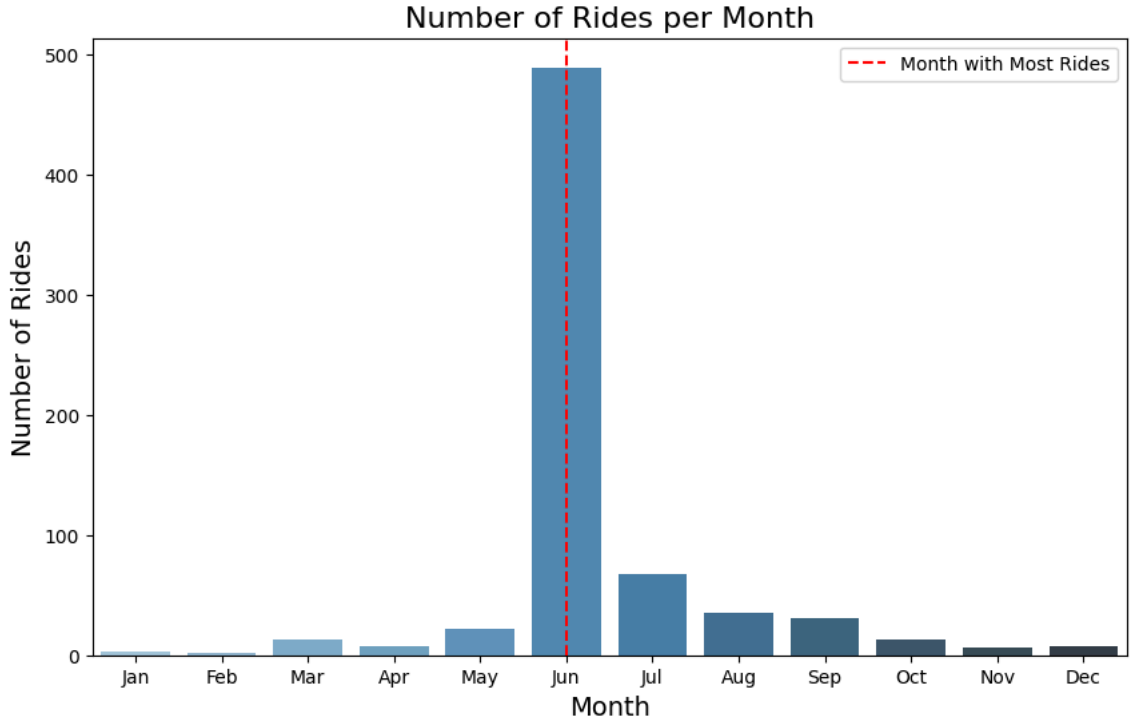
```
# Group by 'start_station_name' and 'rideable_type' and count the number of trips for each combination
bike_usage_by_location = df.groupby(['start_station_name', 'rideable_type']).size().reset_index(name='trip_count')

# Sort the data by location and the number of trips, in descending order
bike_usage_by_location = bike_usage_by_location.sort_values(['start_station_name', 'trip_count'], ascending=[True, False])

# Get the most used bike type for each location by picking the first row for each station after sorting
```

```
most_used_bike_per_location = bike_usage_by_location.groupby('start_station_name').first().reset_index()

# Print the result
print(most_used_bike_per_location)
```

```
                start_station_name  rideable_type  trip_count
0              2112 W Peterson Ave   classic_bike           1
1                   63rd St Beach  electric_bike           1
2                900 W Harrison St    docked_bike           1
3      Aberdeen St & Jackson Blvd  electric_bike           1
4         Aberdeen St & Monroe St   classic_bike           1
..                            ...            ...         ...
690    Wood St & Taylor St (Temp)   classic_bike           1
691         Woodlawn Ave & 55th St   classic_bike           1
692  Woodlawn Ave & Lake Park Ave    docked_bike           1
693           Yates Blvd & 75th St    docked_bike           1
694           Yates Blvd & 93rd St    docked_bike           1

[695 rows x 3 columns]
```

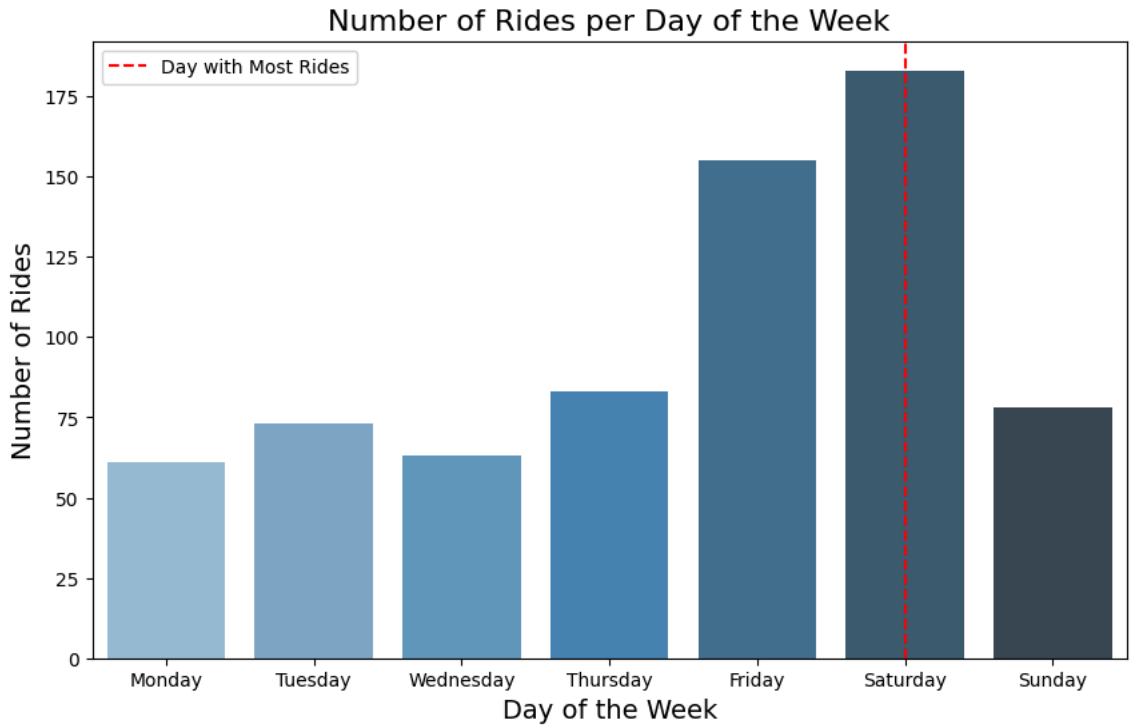
```
# Extract the month from the 'started_at' column
df['month'] = df['started_at'].dt.month

# Group by month and count the number of rides
rides_per_month = df.groupby('month').size().reset_index(name='ride_count')

# Plot the number of rides per month
plt.figure(figsize=(10,6))
sns.barplot(x='month', y='ride_count', data=rides_per_month, palette='Blues_d')

# Add title and labels
plt.title('Number of Rides per Month', fontsize=16)
plt.xlabel('Month', fontsize=14)
plt.ylabel('Number of Rides', fontsize=14)

# Highlight the month with the most rides
max_month = rides_per_month[rides_per_month['ride_count'] == rides_per_month['ride_count'].max()]
plt.axvline(x=max_month.index[0], color='red', linestyle='--', label='Month with Most Rides')

# Show legend
plt.legend()

# Show the plot
plt.xticks(ticks=range(12), labels=['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'])
plt.show()
```

```
<ipython-input-33-bea6f09e923f>:9: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue`

  sns.barplot(x='month', y='ride_count', data=rides_per_month, palette='Blues_d')
```



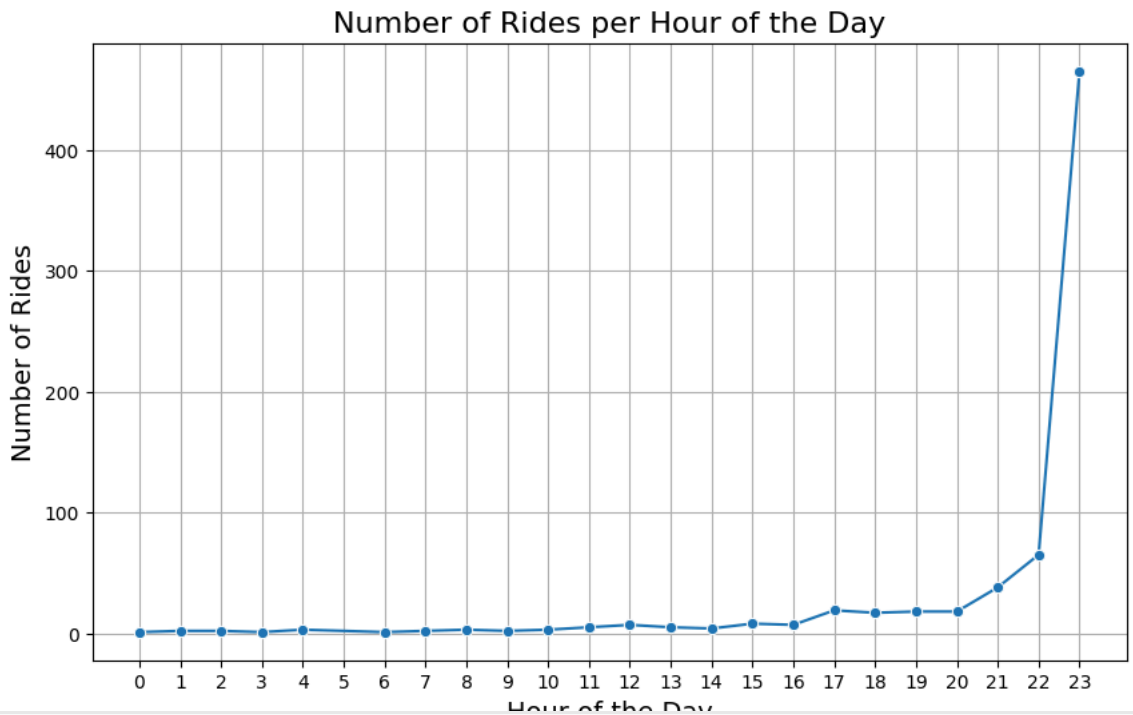
```
# Convert 'started_at' column to datetime
df['started_at'] = pd.to_datetime(df['started_at'])

# Extract the day of the week (0 = Monday, 6 = Sunday)
df['day_of_week'] = df['started_at'].dt.dayofweek

# Group by day of the week and count the number of rides
rides_per_day = df.groupby('day_of_week').size().reset_index(name='ride_count')

# Plot the number of rides per day of the week
plt.figure(figsize=(10,6))
sns.barplot(x='day_of_week', y='ride_count', data=rides_per_day, palette='Blues_d')

# Add title and labels
plt.title('Number of Rides per Day of the Week', fontsize=16)
plt.xlabel('Day of the Week', fontsize=14)
plt.ylabel('Number of Rides', fontsize=14)

# Highlight the day with the most rides
max_day = rides_per_day[rides_per_day['ride_count'] == rides_per_day['ride_count'].max()]
plt.axvline(x=max_day.index[0], color='red', linestyle='--', label='Day with Most Rides')

# Show legend
plt.legend()

# Set custom x-ticks with day names (0 = Monday, 6 = Sunday)
plt.xticks(ticks=range(7), labels=['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday'])

# Show the plot
plt.show()
```

```
<ipython-input-34-b95c940da3e4>:12: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue`

  sns.barplot(x='day_of_week', y='ride_count', data=rides_per_day, palette='Blues_d')
```



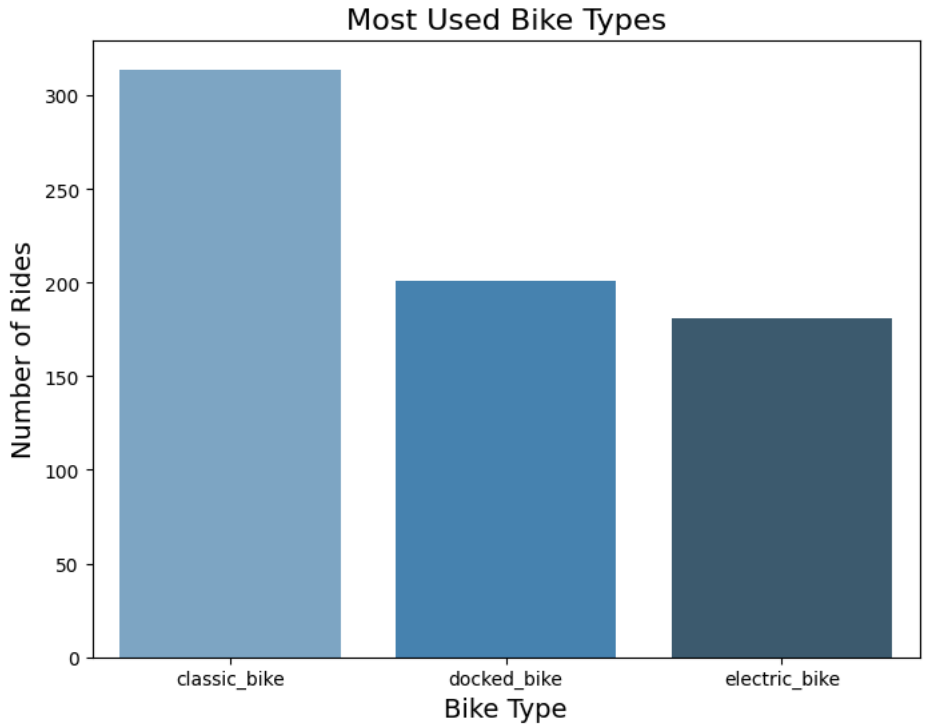

```
# Extract the hour from 'started_at' column
df['hour'] = df['started_at'].dt.hour

# Group by hour and count the number of rides for each hour
rides_per_hour = df.groupby('hour').size().reset_index(name='ride_count')

# Plot a line chart for rides per hour
plt.figure(figsize=(10,6))
sns.lineplot(x='hour', y='ride_count', data=rides_per_hour, marker='o')

# Add title and labels
plt.title('Number of Rides per Hour of the Day', fontsize=16)
plt.xlabel('Hour of the Day', fontsize=14)
plt.ylabel('Number of Rides', fontsize=14)
```

```
# Show the plot
plt.xticks(ticks=range(0, 24))  # Set x-axis ticks to show every hour
plt.grid(True)
plt.show()
```

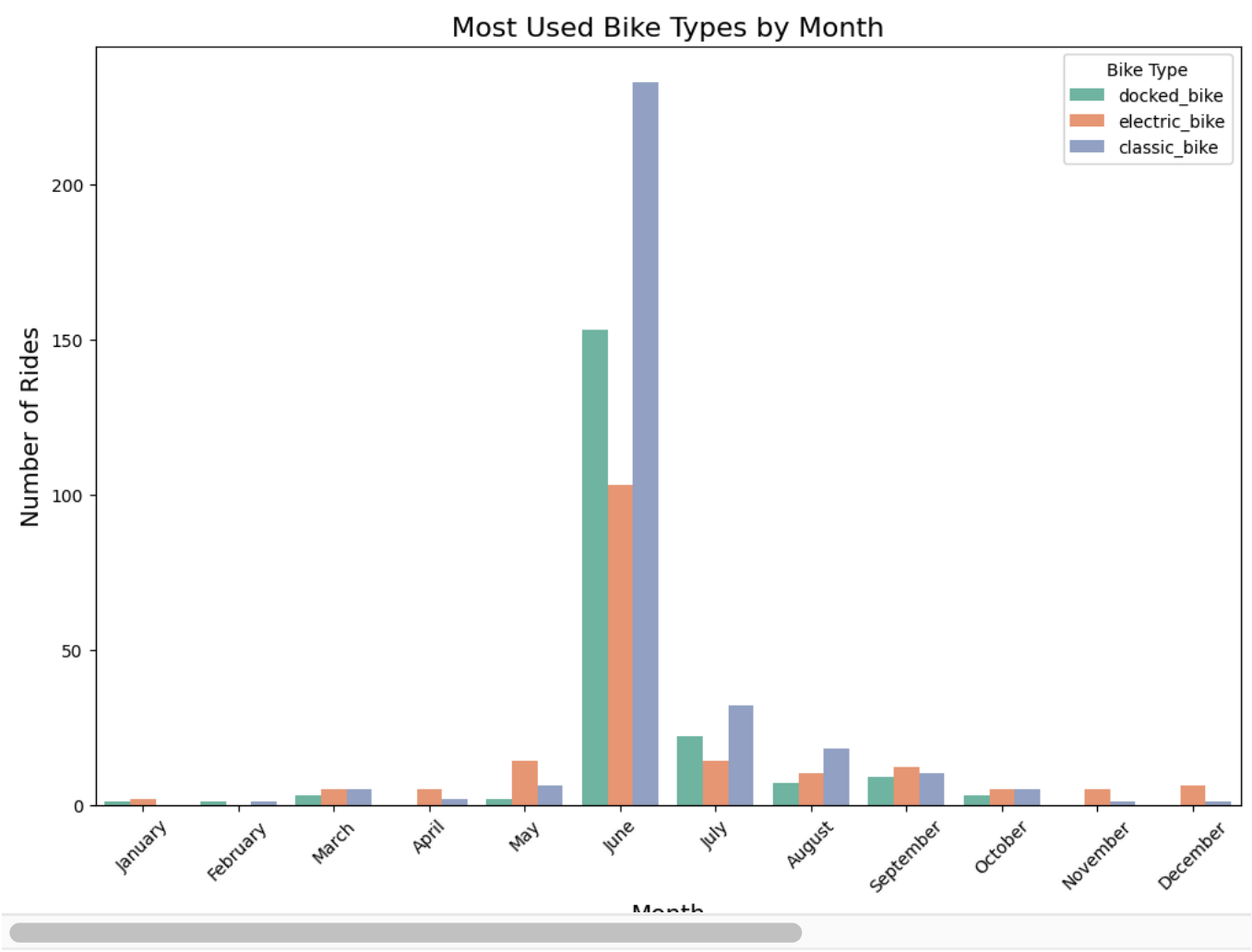



```
# Count the number of rides for each bike type
bike_type_counts = df['rideable_type'].value_counts().reset_index()
bike_type_counts.columns = ['bike_type', 'ride_count']

# Plot a bar chart for the most used bike types
plt.figure(figsize=(8,6))
sns.barplot(x='bike_type', y='ride_count', data=bike_type_counts, palette='Blues_d')

# Add title and labels
plt.title('Most Used Bike Types', fontsize=16)
plt.xlabel('Bike Type', fontsize=14)
plt.ylabel('Number of Rides', fontsize=14)

# Display the plot
plt.show()
```

```
<ipython-input-36-6b58aefeb9f8>:7: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue`

  sns.barplot(x='bike_type', y='ride_count', data=bike_type_counts, palette='Blues_d')
```

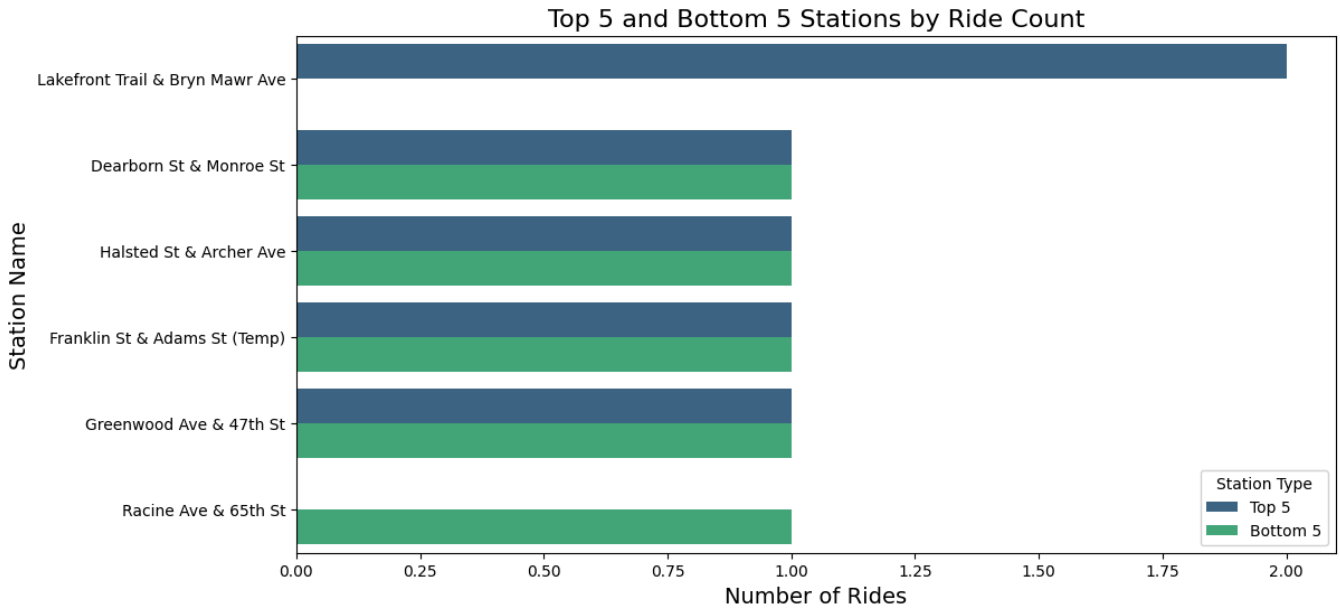



```
# Extract month from 'started_at'
df['month'] = df['started_at'].dt.month_name()

# Group by month and bike type, then count the number of rides
monthly_bike_usage = df.groupby(['month', 'rideable_type']).size().reset_index(name='ride_count')

# Order months to ensure the plot is in chronological order
month_order = ['January', 'February', 'March', 'April', 'May', 'June',
               'July', 'August', 'September', 'October', 'November', 'December']
monthly_bike_usage['month'] = pd.Categorical(monthly_bike_usage['month'], categories=month_order, ordered=True)
monthly_bike_usage = monthly_bike_usage.sort_values('month')

# Plot the results using a bar plot
plt.figure(figsize=(12, 8))
sns.barplot(x='month', y='ride_count', hue='rideable_type', data=monthly_bike_usage, palette='Set2')

# Add title and labels
plt.title('Most Used Bike Types by Month', fontsize=16)
plt.xlabel('Month', fontsize=14)
plt.ylabel('Number of Rides', fontsize=14)

# Display the plot
plt.xticks(rotation=45)  # Rotate x labels for better readability
plt.legend(title='Bike Type')
plt.show()
```



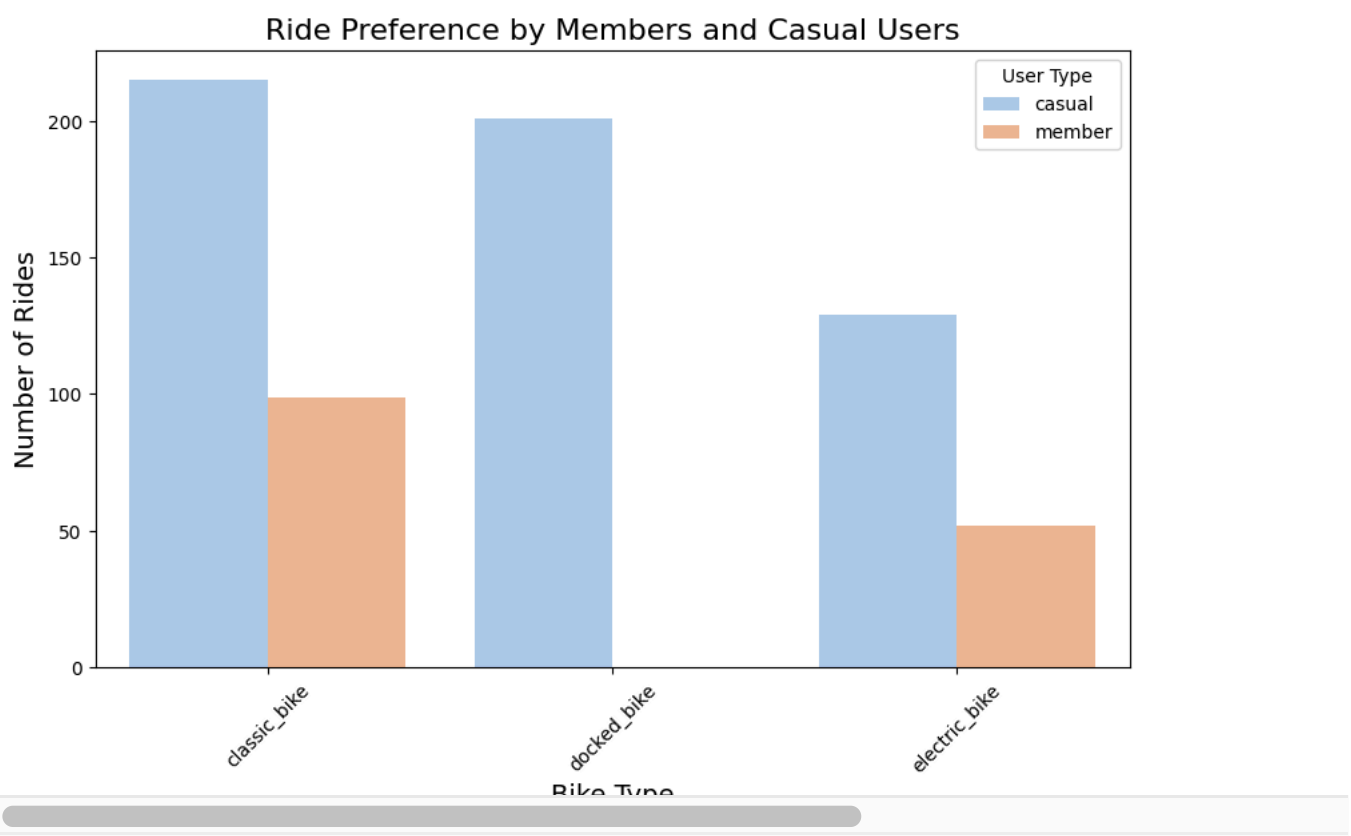
Start coding or generate with AI.

```
# Count the number of rides per station
station_counts = df['start_station_name'].value_counts().reset_index()
station_counts.columns = ['station_name', 'ride_count']

# Get the top 5 and bottom 5 stations
top_stations = station_counts.nlargest(5, 'ride_count')
bottom_stations = station_counts.nsmallest(5, 'ride_count')

# Combine the data
combined_stations = pd.concat([top_stations, bottom_stations])

# Create a color column for better distinction
combined_stations['station_type'] = ['Top 5'] * 5 + ['Bottom 5'] * 5

# Plotting
plt.figure(figsize=(12, 6))
sns.barplot(x='ride_count', y='station_name', hue='station_type', data=combined_stations, palette='viridis')

# Add titles and labels
plt.title('Top 5 and Bottom 5 Stations by Ride Count', fontsize=16)
plt.xlabel('Number of Rides', fontsize=14)
plt.ylabel('Station Name', fontsize=14)

# Show the plot
plt.legend(title='Station Type')
plt.show()
```



Start coding or generate with AI.

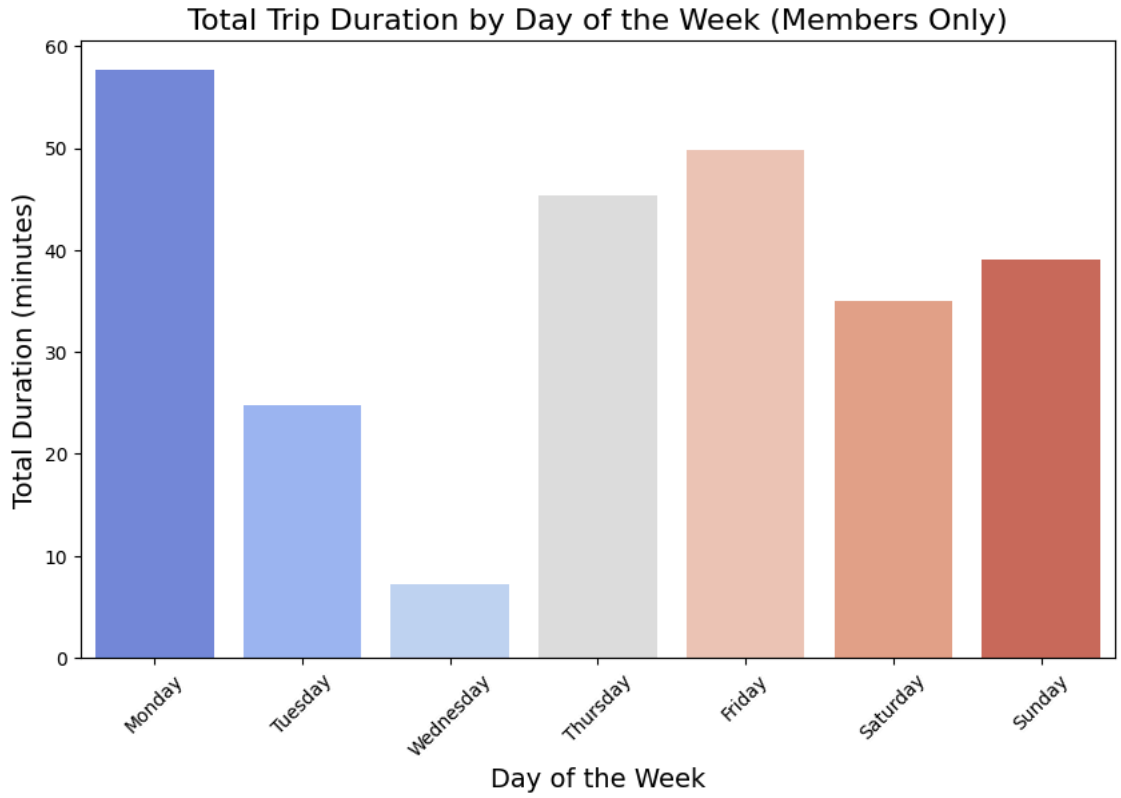
```
# Load the dataset (assuming 'rideable_type' and 'member_casual' columns exist)
#df = pd.read_csv('yourfile.csv', header=1)

# Group by rideable type and member type, then count the number of rides
ride_preference = df.groupby(['rideable_type', 'member_casual']).size().reset_index(name='ride_count')

# Create the plot
plt.figure(figsize=(10, 6))
sns.barplot(x='rideable_type', y='ride_count', hue='member_casual', data=ride_preference, palette='pastel')

# Add titles and labels
plt.title('Ride Preference by Members and Casual Users', fontsize=16)
plt.xlabel('Bike Type', fontsize=14)
plt.ylabel('Number of Rides', fontsize=14)

# Show the plot
plt.legend(title='User Type')
plt.xticks(rotation=45)  # Rotate x labels for better readability
plt.show()
```



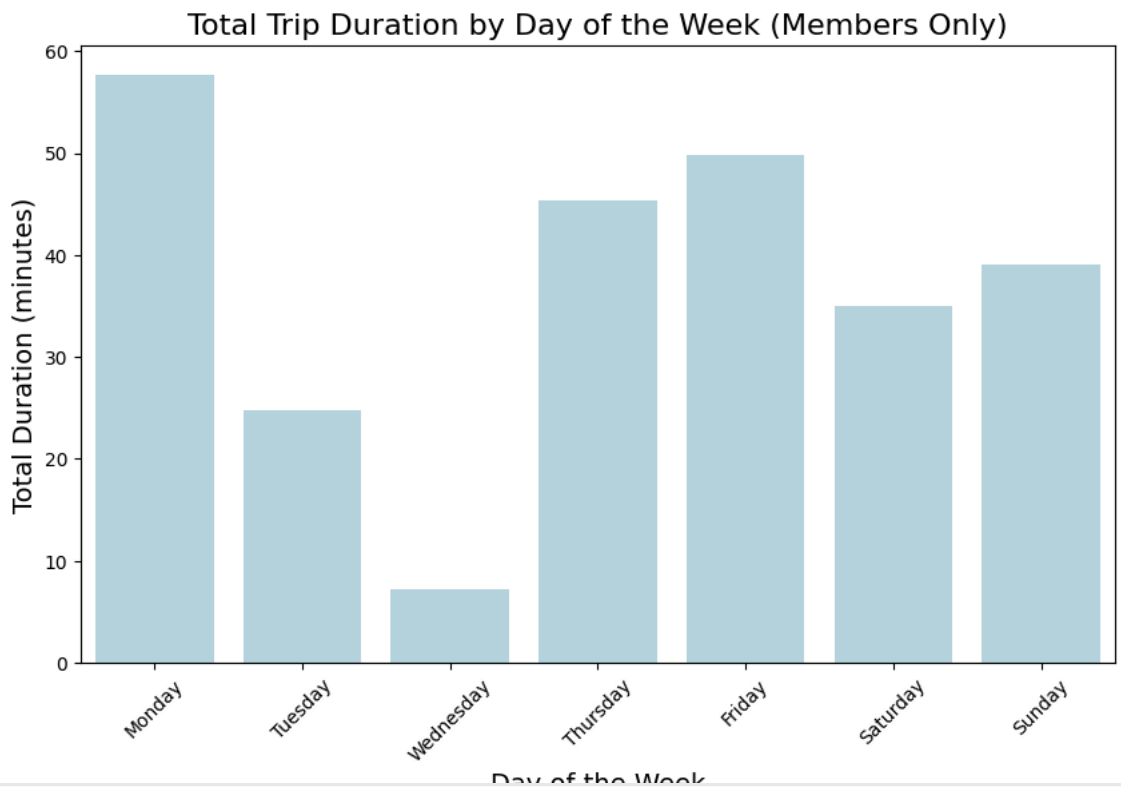

```
Start coding or generate with AI.
```

```
# Extract the day of the week
df['day_of_week'] = df['started_at'].dt.day_name()

# Filter for members only
members_df = df[df['member_casual'] == 'member']

# Calculate total trip duration for each day of the week
# Assuming 'ride_duration' is in seconds
trip_duration_by_day = members_df.groupby('day_of_week').agg(
    total_duration=('trip_duration_min', 'sum') # Changed 'ride_length' to 'ride_duration'
).reset_index()

# Convert total duration from seconds to minutes
trip_duration_by_day['total_duration'] = trip_duration_by_day['total_duration'] / 60

# Order the days of the week
day_order = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
trip_duration_by_day['day_of_week'] = pd.Categorical(trip_duration_by_day['day_of_week'], categories=day_order, ordered=True
trip_duration_by_day = trip_duration_by_day.sort_values('day_of_week')

# Plotting
plt.figure(figsize=(10, 6))
sns.barplot(x='day_of_week', y='total_duration', data=trip_duration_by_day, palette='coolwarm')

# Add titles and labels
plt.title('Total Trip Duration by Day of the Week (Members Only)', fontsize=16)
plt.xlabel('Day of the Week', fontsize=14)
plt.ylabel('Total Duration (minutes)', fontsize=14)

# Show the plot
plt.xticks(rotation=45)  # Rotate x labels for better readability
plt.show()
```

```
<ipython-input-40-03c2775f6cc1>:23: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue`

  sns.barplot(x='day_of_week', y='total_duration', data=trip_duration_by_day, palette='coolwarm')
```

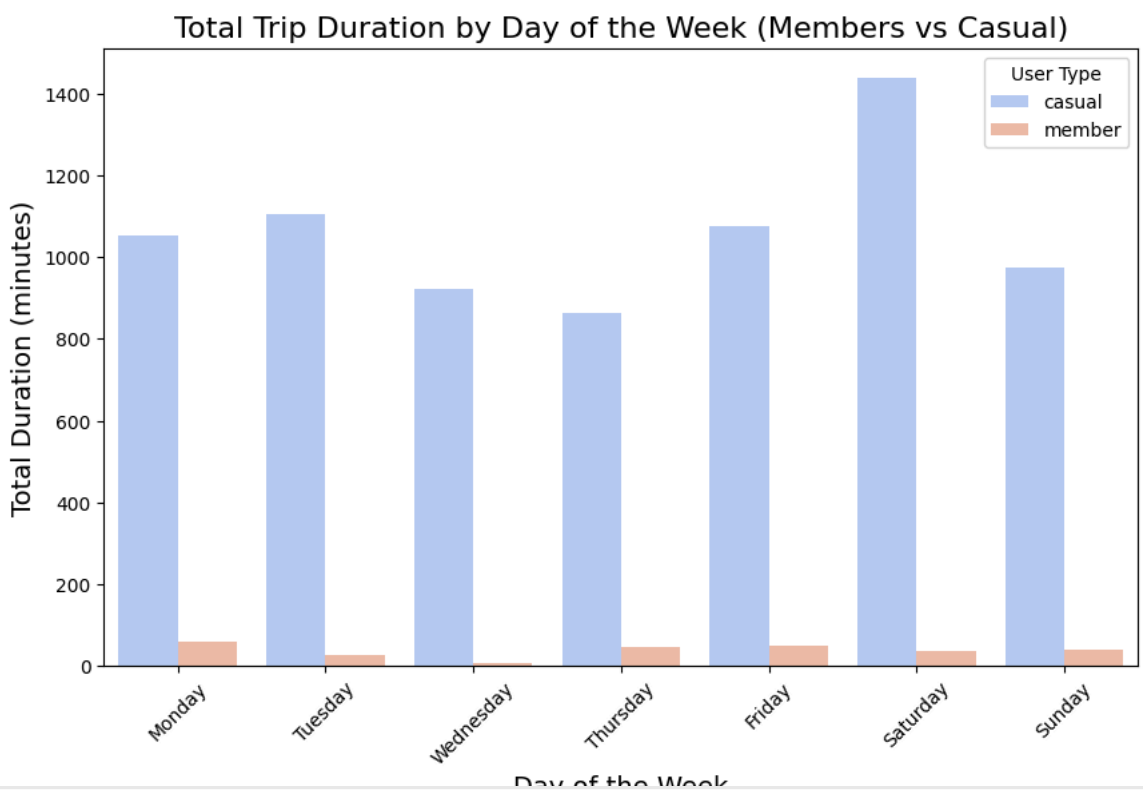



```
# Plotting without passing `palette`
plt.figure(figsize=(10, 6))
sns.barplot(x='day_of_week', y='total_duration', data=trip_duration_by_day, color='lightblue')

# Add titles and labels
plt.title('Total Trip Duration by Day of the Week (Members Only)', fontsize=16)
plt.xlabel('Day of the Week', fontsize=14)
plt.ylabel('Total Duration (minutes)', fontsize=14)

plt.xticks(rotation=45)  # Rotate x labels for better readability
plt.show()
```



Start coding or generate with AI.

```
# Order the days of the week for the x-axis
day_order = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
trip_duration_by_day['day_of_week'] = pd.Categorical(trip_duration_by_day['day_of_week'], categories=day_order, ordered=True

# Calculate total trip duration for each day of the week for all riders
trip_duration_by_day_all = df.groupby(['day_of_week', 'member_casual']).agg(
    total_duration=('trip_duration_min', 'sum')).reset_index()

# Convert total duration from seconds to minutes
trip_duration_by_day_all['total_duration'] = trip_duration_by_day_all['total_duration'] / 60

trip_duration_by_day_all['day_of_week'] = pd.Categorical(trip_duration_by_day_all['day_of_week'], categories=day_order, orde

# Plotting the total trip duration by day of the week, using 'member_casual' as hue
plt.figure(figsize=(10, 6))
sns.barplot(x='day_of_week', y='total_duration', hue='member_casual', data=trip_duration_by_day_all, palette='coolwarm') # C

# Add titles and labels
plt.title('Total Trip Duration by Day of the Week (Members vs Casual)', fontsize=16)
plt.xlabel('Day of the Week', fontsize=14)
plt.ylabel('Total Duration (minutes)', fontsize=14)

# Rotate x-axis labels for better readability
plt.xticks(rotation=45)

# Show the legend and the plot
plt.legend(title='User Type')
plt.show()
```

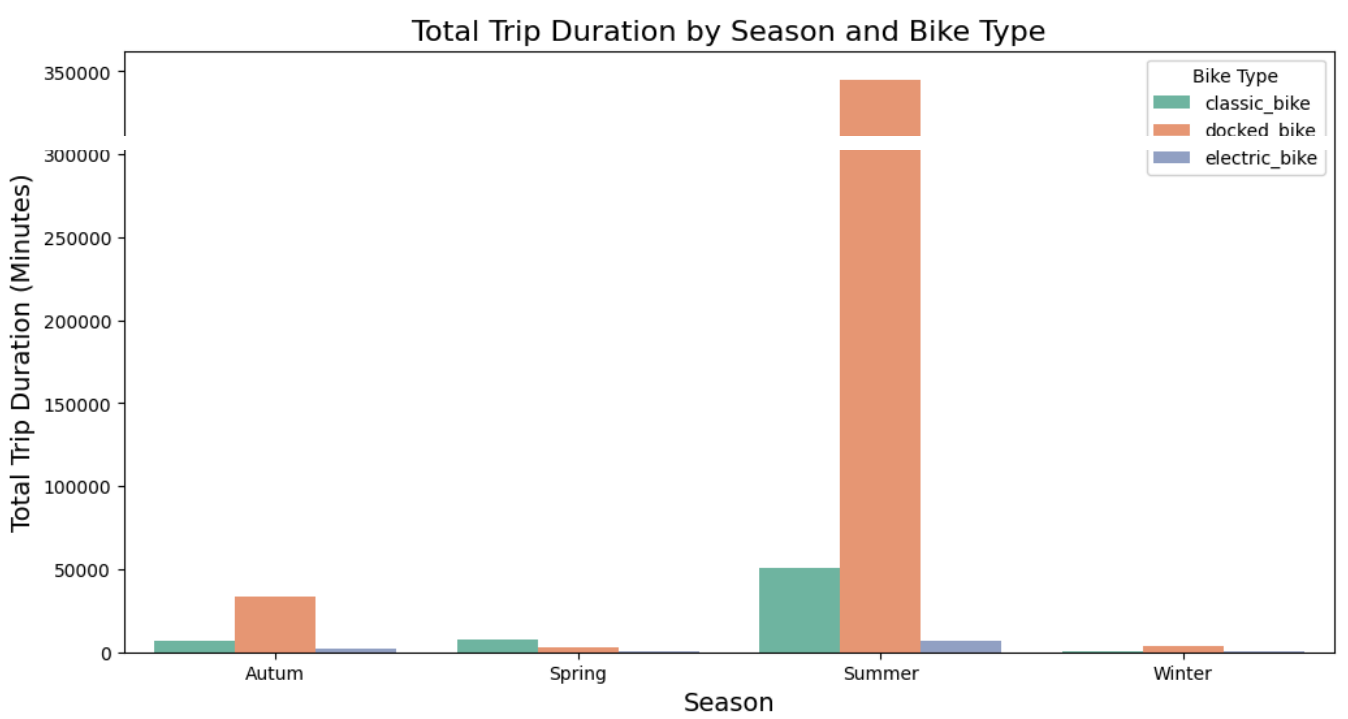



```
#Extrac month from started_at column
df['month'] = df['started_at'].dt.month


#Define seasons based on months
def get_season(month):
    if month in [12, 1, 2]:
        return 'Winter'
    elif month in [3, 4, 5]:
        return 'Spring'
    elif month in [6, 7, 8]:
        return 'Summer'
    elif month in [9, 10, 11]:
        return 'Autum'
    else:
        return 'Fall'

# Create new column by applying the function
df['season'] = df['month'].apply(get_season)


# Group by 'season' and 'rideable_type' and count the occurrences

season_usage = df.groupby(['season', 'rideable_type']).size().reset_index(name='ride_count')


# Find the most used bike type for each season
most_used_bike_per_season = season_usage.loc[season_usage.groupby('season')['ride_count'].idxmax()]


most_used_bike_per_season
```

	season	rideable_type	ride_count
2	Autum	electric_bike	22
5	Spring	electric_bike	24
6	Summer	classic_bike	283

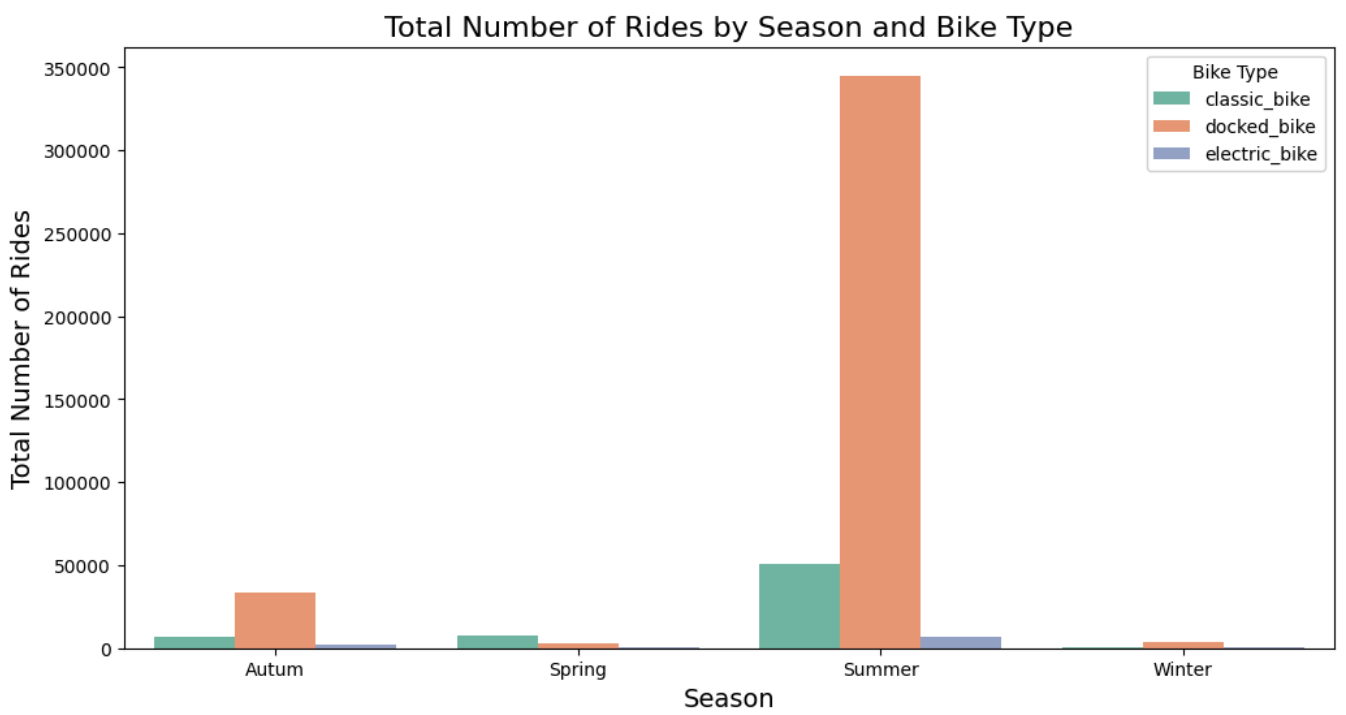
```
# Group by 'season' and 'rideable_type' and calculate total duration
season_usage = df.groupby(['season', 'rideable_type'])['trip_duration_min'].sum().reset_index(name='total_duration_minutes')

# Find the most used bike type for each season
most_used_bike_per_season = season_usage.loc[season_usage.groupby('season')['total_duration_minutes'].idxmax()]
```

```
# Plotting total trip duration by season and bike type
plt.figure(figsize=(12, 6))
sns.barplot(x='season', y='total_duration_minutes', hue='rideable_type', data=season_usage, palette='Set2')

# Add titles and labels
plt.title('Total Trip Duration by Season and Bike Type', fontsize=16)
plt.xlabel('Season', fontsize=14)
plt.ylabel('Total Trip Duration (Minutes)', fontsize=14)

# Show the legend and plot
plt.legend(title='Bike Type')
plt.show()
```



Start coding or generate with AI.

```
# Plotting total ride counts by season and bike type
plt.figure(figsize=(12, 6))
sns.barplot(x='season', y='total_duration_minutes', hue='rideable_type', data=season_usage, palette='Set2')

# Add titles and labels
plt.title('Total Number of Rides by Season and Bike Type', fontsize=16)
plt.xlabel('Season', fontsize=14)
plt.ylabel('Total Number of Rides', fontsize=14)

# Show the legend and plot
plt.legend(title='Bike Type')
plt.show()
```




```
df.head()
```

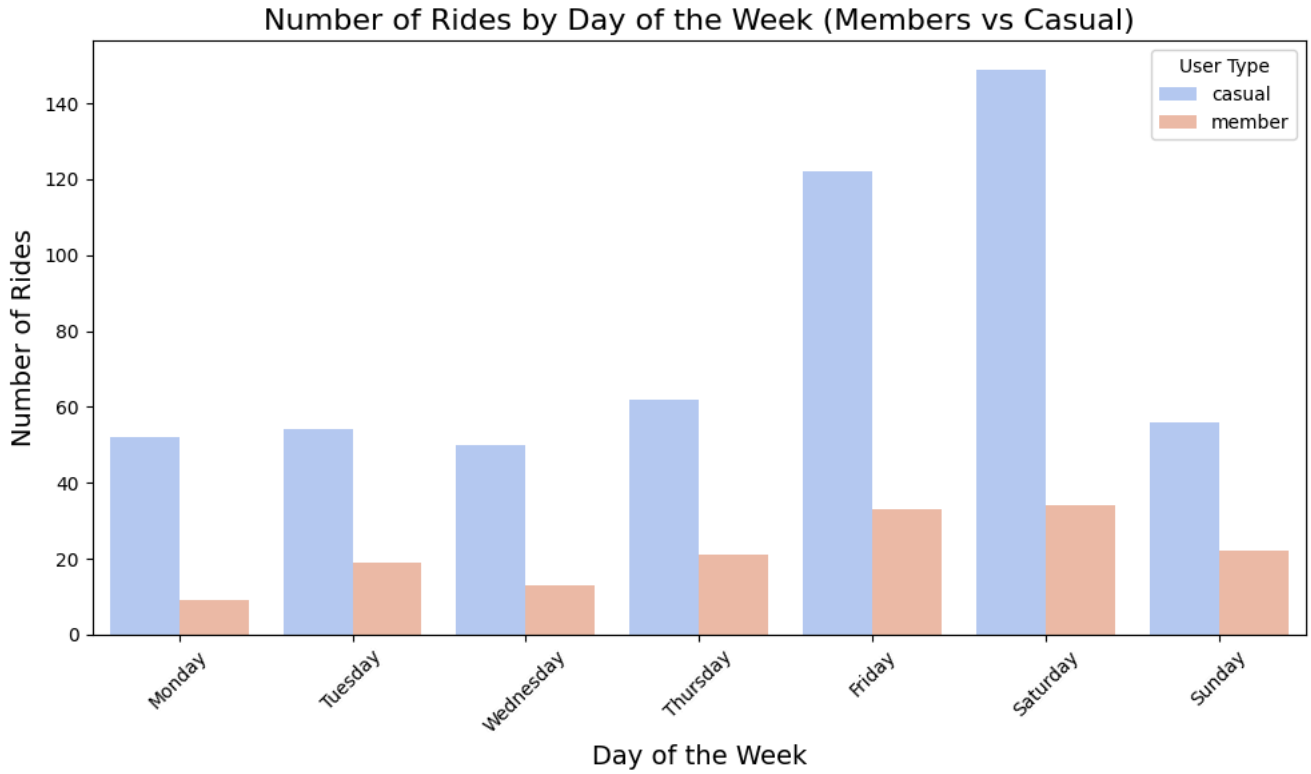
	ride_id	rideable_type	started_at	new_start_date	new_started_time	ended_at	new_end_at	new_ended_time	st
0	F6496DF223062E4D	electric_bike	2021-06-13 22:35:00	13-Jun-21	22:35:00	2021-06-14 00:14:00	14-Jun-21	00:14:00	D
1	2B218870CDC78BB2	classic_bike	2021-06-05 23:46:00	05-Jun-21	23:46:00	2021-06-06 00:10:00	06-Jun-21	00:10:00	La
2	067B43F67F5DF530	classic_bike	2021-06-11 23:30:00	11-Jun-21	23:30:00	2021-06-12 00:04:00	12-Jun-21	00:04:00	
3	E90B0906B0E6AE92	electric_bike	2021-06-22 23:38:00	22-Jun-21	23:38:00	2021-06-23 01:52:00	23-Jun-21	01:52:00	M
4	6A49443B53ED1E7D	classic_bike	2021-06-12 23:12:00	12-Jun-21	23:12:00	2021-06-13 01:12:00	13-Jun-21	01:12:00	D

5 rows × 25 columns

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming you have already loaded your DataFrame 'df'

# Step 1: Ensure 'started_at' is in datetime format
df['started_at'] = pd.to_datetime(df['started_at'])

# Step 2: Extract day of the week
df['day_of_week'] = df['started_at'].dt.day_name()

# Step 3: Order the days of the week for the x-axis
day_order = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
df['day_of_week'] = pd.Categorical(df['day_of_week'], categories=day_order, ordered=True)

# Step 4: Count the number of rides for each day of the week for all riders
rides_per_day_all = df.groupby(['day_of_week', 'member_casual']).agg(
    ride_count=('ride_id', 'count')
).reset_index()

# Step 5: Plotting the number of rides by day of the week, using 'member_casual' as hue
plt.figure(figsize=(10, 6))
sns.barplot(x='day_of_week', y='ride_count', hue='member_casual', data=rides_per_day_all, palette='coolwarm')

# Add titles and labels
```

```
plt.title('Number of Rides by Day of the Week (Members vs Casual)', fontsize=16)
plt.xlabel('Day of the Week', fontsize=14)
plt.ylabel('Number of Rides', fontsize=14)

# Rotate x-axis labels for better readability
plt.xticks(rotation=45)

# Show the legend and the plot
plt.legend(title='User Type')
plt.tight_layout()  # Adjust layout to prevent clipping of labels
plt.show()
```

```
<ipython-input-51-b5150c5a4183>:18: FutureWarning: The default of observed=False is deprecated and will be changed to Tr
  rides_per_day_all = df.groupby(['day_of_week', 'member_casual']).agg(
```

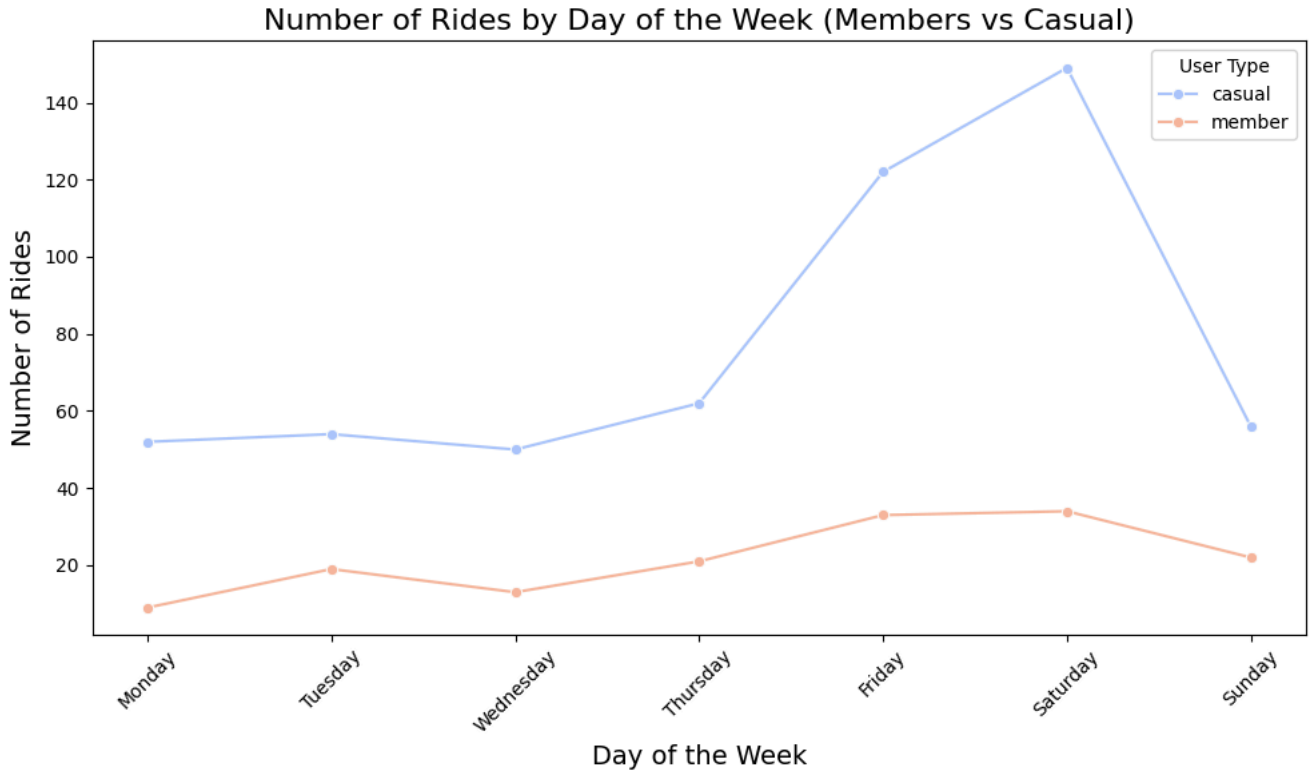


Start coding or generate with AI.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming you have already loaded your DataFrame 'df'

# Step 1: Ensure 'started_at' is in datetime format
df['started_at'] = pd.to_datetime(df['started_at'])

# Step 2: Extract day of the week
df['day_of_week'] = df['started_at'].dt.day_name()

# Step 3: Order the days of the week for the x-axis
day_order = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
df['day_of_week'] = pd.Categorical(df['day_of_week'], categories=day_order, ordered=True)

# Step 4: Count the number of rides for each day of the week for all riders (members and casual)
rides_per_day_all = df.groupby(['day_of_week', 'member_casual']).agg(
    ride_count=('ride_id', 'count')
).reset_index()

# Step 5: Plotting the number of rides by day of the week as a line chart, using 'member_casual' as hue
plt.figure(figsize=(10, 6))
sns.lineplot(x='day_of_week', y='ride_count', hue='member_casual', data=rides_per_day_all, marker='o', palette='coolwarm')

# Add titles and labels
plt.title('Number of Rides by Day of the Week (Members vs Casual)', fontsize=16)
plt.xlabel('Day of the Week', fontsize=14)
plt.ylabel('Number of Rides', fontsize=14)

# Rotate x-axis labels for better readability
plt.xticks(rotation=45)
```

```
# Show the legend and the plot
plt.legend(title='User Type')
plt.tight_layout()  # Adjust layout to prevent clipping of labels
plt.show()
```

```
<ipython-input-52-6ae838d7e034>:18: FutureWarning: The default of observed=False is deprecated and will be changed to Tr
  rides_per_day_all = df.groupby(['day_of_week', 'member_casual']).agg(
```



Start coding or generate with AI.

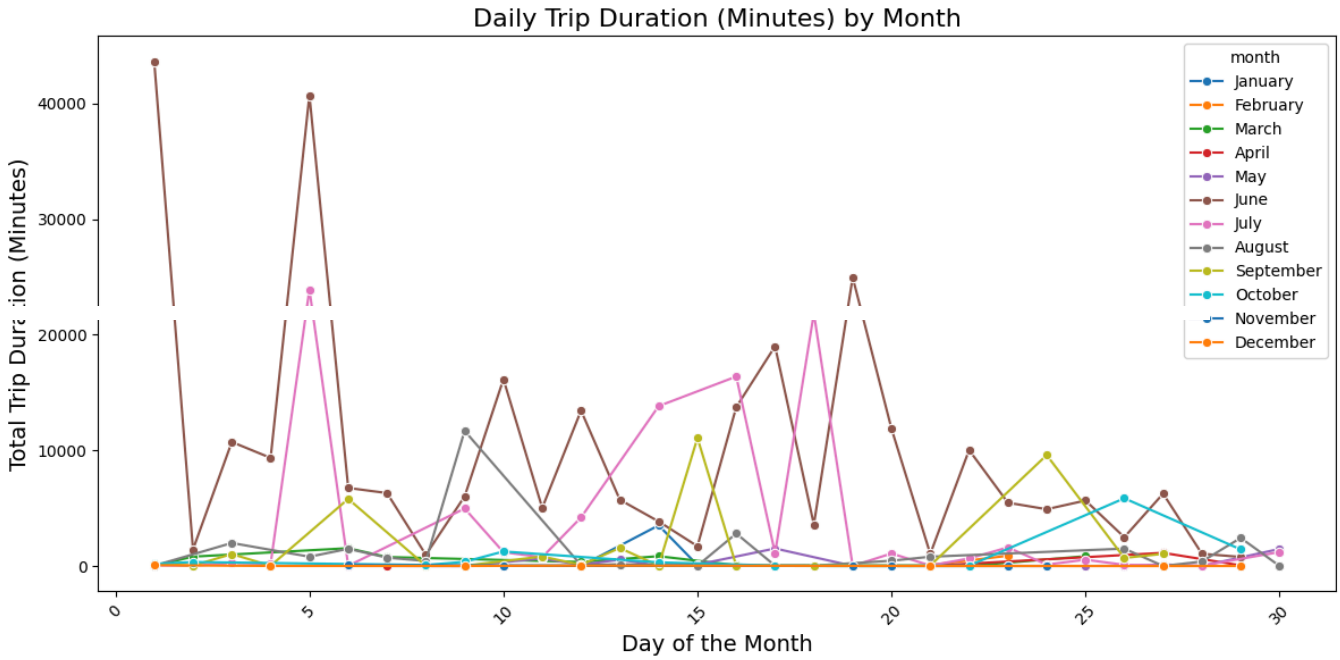
```
df.head()
```

	ride_id	rideable_type	started_at	new_start_date	new_started_time	ended_at	new_end_at	new_ended_time	st
0	F6496DF223062E4D	electric_bike	2021-06-13 22:35:00	13-Jun-21	22:35:00	2021-06-14 00:14:00	14-Jun-21	00:14:00	D
1	2B218870CDC78BB2	classic_bike	2021-06-05 23:46:00	05-Jun-21	23:46:00	2021-06-06 00:10:00	06-Jun-21	00:10:00	La
2	067B43F67F5DF530	classic_bike	2021-06-11 23:30:00	11-Jun-21	23:30:00	2021-06-12 00:04:00	12-Jun-21	00:04:00	
3	E90B0906B0E6AE92	electric_bike	2021-06-22 23:38:00	22-Jun-21	23:38:00	2021-06-23 01:52:00	23-Jun-21	01:52:00	M
4	6A49443B53ED1E7D	classic_bike	2021-06-12 23:12:00	12-Jun-21	23:12:00	2021-06-13 01:12:00	13-Jun-21	01:12:00	D

5 rows × 25 columns

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Sample data (replace with your actual data loading process)
# df = pd.read_csv('your_bike_data.csv')

# Ensure 'started_at' is in datetime format
df['started_at'] = pd.to_datetime(df['started_at'])

# Create new columns for the day and month
df['day'] = df['started_at'].dt.day
df['month'] = df['started_at'].dt.month_name()  # Month name for better readability
```

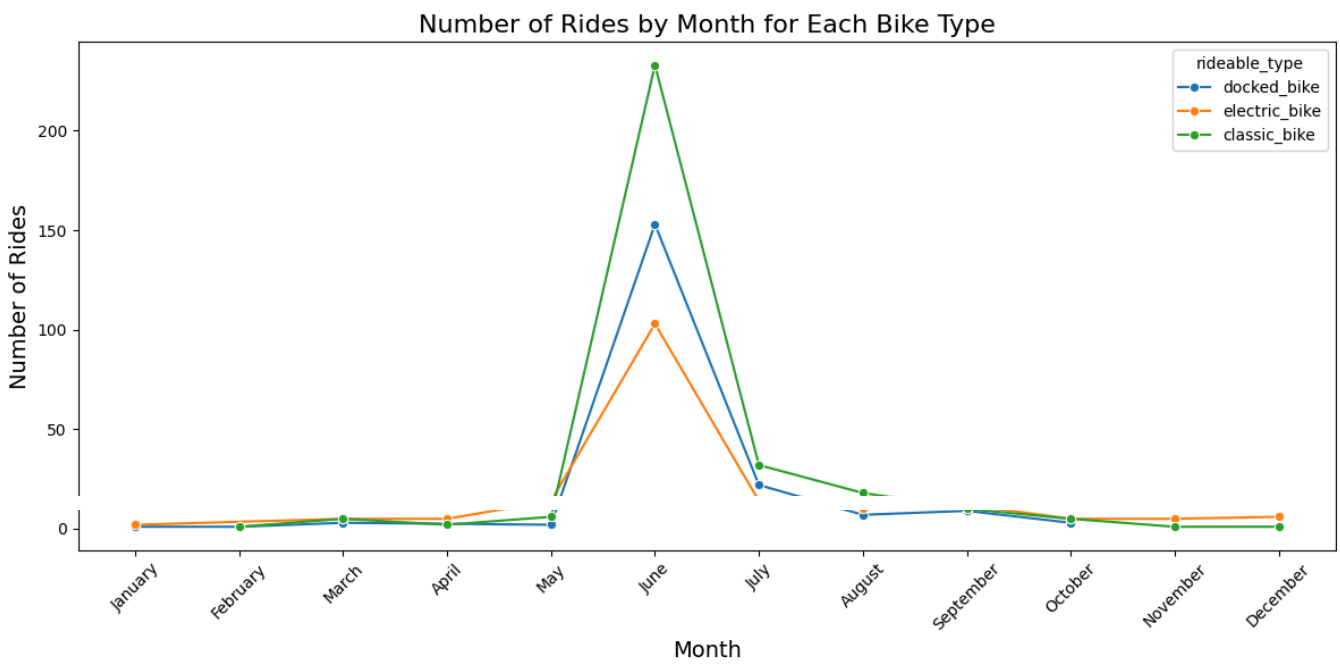
```
# Group by 'day' and 'month' and calculate total daily trip duration
daily_trip_duration = df.groupby(['month', 'day']).agg(
    total_duration=('trip_duration_min', 'sum')
).reset_index()

# Sort the data by month (order can be adjusted as per your dataset)
month_order = ['January', 'February', 'March', 'April', 'May', 'June', 'July', 'August', 'September', 'October', 'November',
daily_trip_duration['month'] = pd.Categorical(daily_trip_duration['month'], categories=month_order, ordered=True)
daily_trip_duration = daily_trip_duration.sort_values(['month', 'day'])

# Plotting the line graph
plt.figure(figsize=(12, 6))
sns.lineplot(x='day', y='total_duration', hue='month', data=daily_trip_duration, palette='tab10', marker='o')

# Add titles and labels
plt.title('Daily Trip Duration (Minutes) by Month', fontsize=16)
plt.xlabel('Day of the Month', fontsize=14)
plt.ylabel('Total Trip Duration (Minutes)', fontsize=14)

# Rotate the x-axis labels if necessary
plt.xticks(rotation=45)

# Show the plot
plt.tight_layout()
plt.show()
```



Start coding or generate with AI.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Sample data loading (replace with your actual DataFrame)
# df = pd.read_csv('your_bike_data.csv')

# Step 1: Ensure 'started_at' is in datetime format
df['started_at'] = pd.to_datetime(df['started_at'])

# Step 2: Extract month from 'started_at'
df['month'] = df['started_at'].dt.month_name()  # Extract month names for readability

# Step 3: Group by 'month' and 'rideable_type' to calculate the number of rides
rides_per_month_bike = df.groupby(['month', 'rideable_type']).size().reset_index(name='ride_count')

# Step 4: Order the months for the x-axis
month_order = ['January', 'February', 'March', 'April', 'May', 'June', 'July', 'August', 'September', 'October', 'November',
rides_per_month_bike['month'] = pd.Categorical(rides_per_month_bike['month'], categories=month_order, ordered=True)
rides_per_month_bike = rides_per_month_bike.sort_values('month')
```

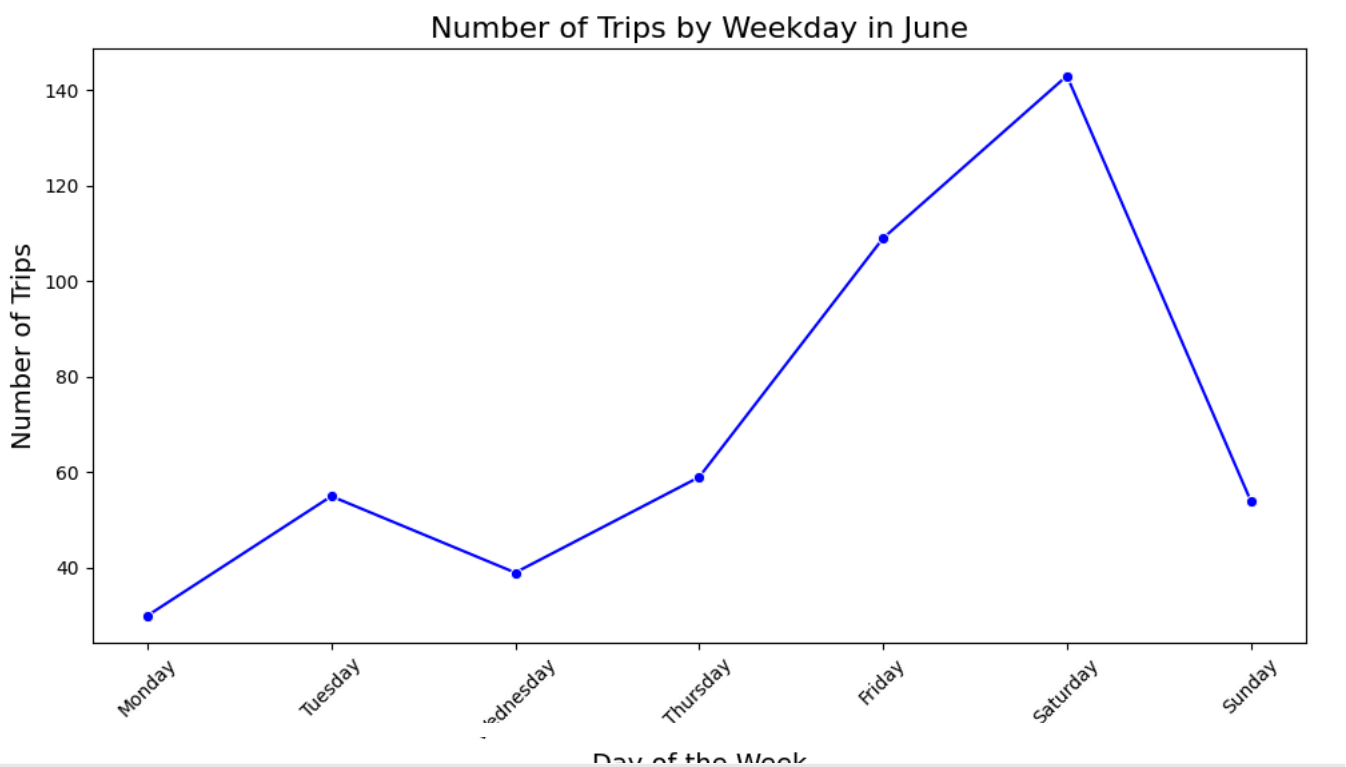
```
# Step 5: Plotting the line chart for the number of rides by month for each bike type
plt.figure(figsize=(12, 6))
sns.lineplot(x='month', y='ride_count', hue='rideable_type', data=rides_per_month_bike, marker='o')

# Add titles and labels
plt.title('Number of Rides by Month for Each Bike Type', fontsize=16)
plt.xlabel('Month', fontsize=14)
plt.ylabel('Number of Rides', fontsize=14)

# Rotate x-axis labels for better readability
plt.xticks(rotation=45)

# Show the plot
plt.tight_layout()
plt.show()
```



Start coding or generate with AI.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Sample data loading (replace with your actual DataFrame)
# df = pd.read_csv('your_bike_data.csv')

# Step 1: Ensure 'started_at' is in datetime format
df['started_at'] = pd.to_datetime(df['started_at'])

# Step 2: Extract month and day of the week from 'started_at'
df['month'] = df['started_at'].dt.month_name()  # Month name for better readability
df['day_of_week'] = df['started_at'].dt.day_name()  # Extract day of the week

# Step 3: Filter data for June
df_june = df[df['month'] == 'June']

# Step 4: Group by 'day_of_week' to calculate the number of trips per day
# Count the number of rides per day of the week for June
rides_per_weekday_june = df_june.groupby('day_of_week').size().reset_index(name='trip_count')

# Step 5: Order the days of the week for the x-axis
day_order = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
rides_per_weekday_june['day_of_week'] = pd.Categorical(rides_per_weekday_june['day_of_week'], categories=day_order, ordered=
rides_per_weekday_june = rides_per_weekday_june.sort_values('day_of_week')

# Step 6: Plotting the line chart for bike counts per weekday in June
plt.figure(figsize=(10, 6))
sns.lineplot(x='day_of_week', y='trip_count', data=rides_per_weekday_june, marker='o', color='b')
```

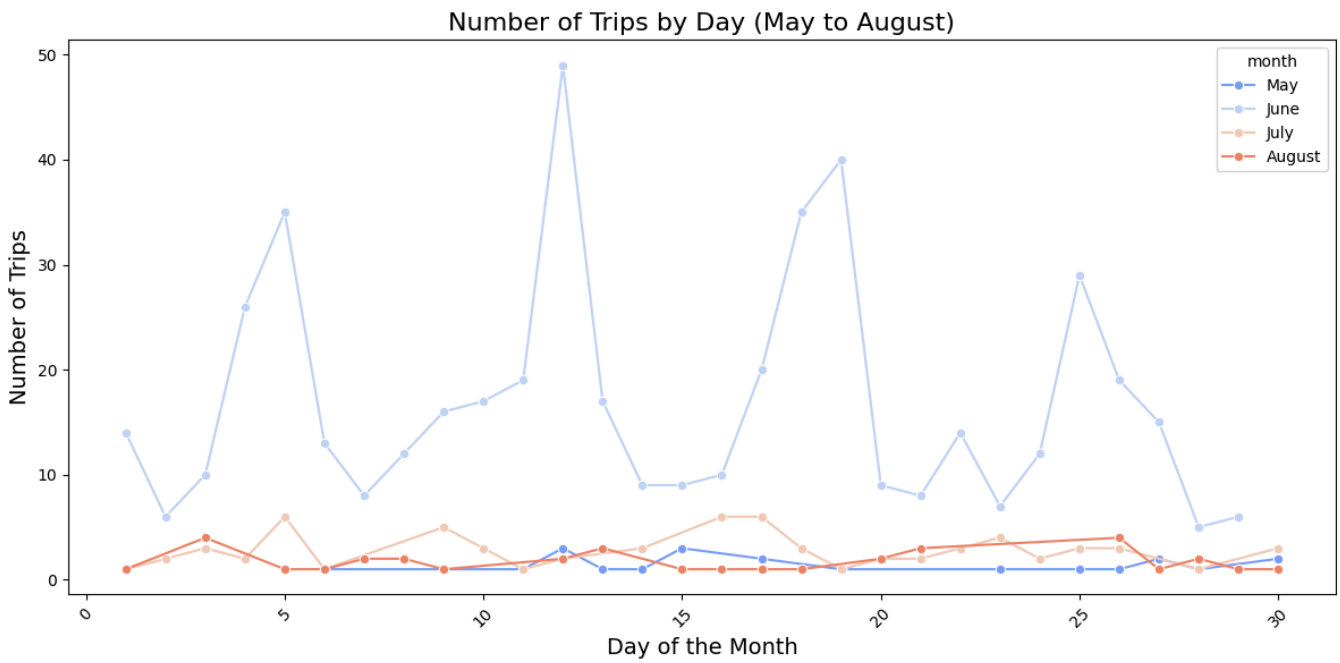
```
# Add titles and labels
plt.title('Number of Trips by Weekday in June', fontsize=16)
plt.xlabel('Day of the Week', fontsize=14)
plt.ylabel('Number of Trips', fontsize=14)

# Rotate x-axis labels for better readability
plt.xticks(rotation=45)

# Show the plot
plt.tight_layout()
plt.show()
```



Start coding or generate with AI.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming your data is loaded into a DataFrame 'df'
# Sample data loading (replace with your actual data loading process)
# df = pd.read_csv('your_bike_data.csv')

# Step 1: Ensure 'started_at' is in datetime format
df['started_at'] = pd.to_datetime(df['started_at'])

# Step 2: Extract the day and month from 'started_at'
df['day'] = df['started_at'].dt.day
df['month'] = df['started_at'].dt.month_name()  # Month name for better readability

# Step 3: Filter for months from May to August
months_of_interest = ['May', 'June', 'July', 'August']
df_filtered = df[df['month'].isin(months_of_interest)]

# Step 4: Group by 'month' and 'day' to calculate the number of trips
daily_trip_count = df_filtered.groupby(['month', 'day']).size().reset_index(name='trip_count')

# Step 5: Sort the data by month and day
month_order = ['May', 'June', 'July', 'August']
daily_trip_count['month'] = pd.Categorical(daily_trip_count['month'], categories=month_order, ordered=True)
daily_trip_count = daily_trip_count.sort_values(['month', 'day'])

# Step 6: Plotting the line chart for trip counts by day of the month for each month
plt.figure(figsize=(12, 6))
sns.lineplot(x='day', y='trip_count', hue='month', data=daily_trip_count, marker='o', palette='coolwarm')

# Add titles and labels
plt.title('Number of Trips by Day (May to August)', fontsize=16)
```

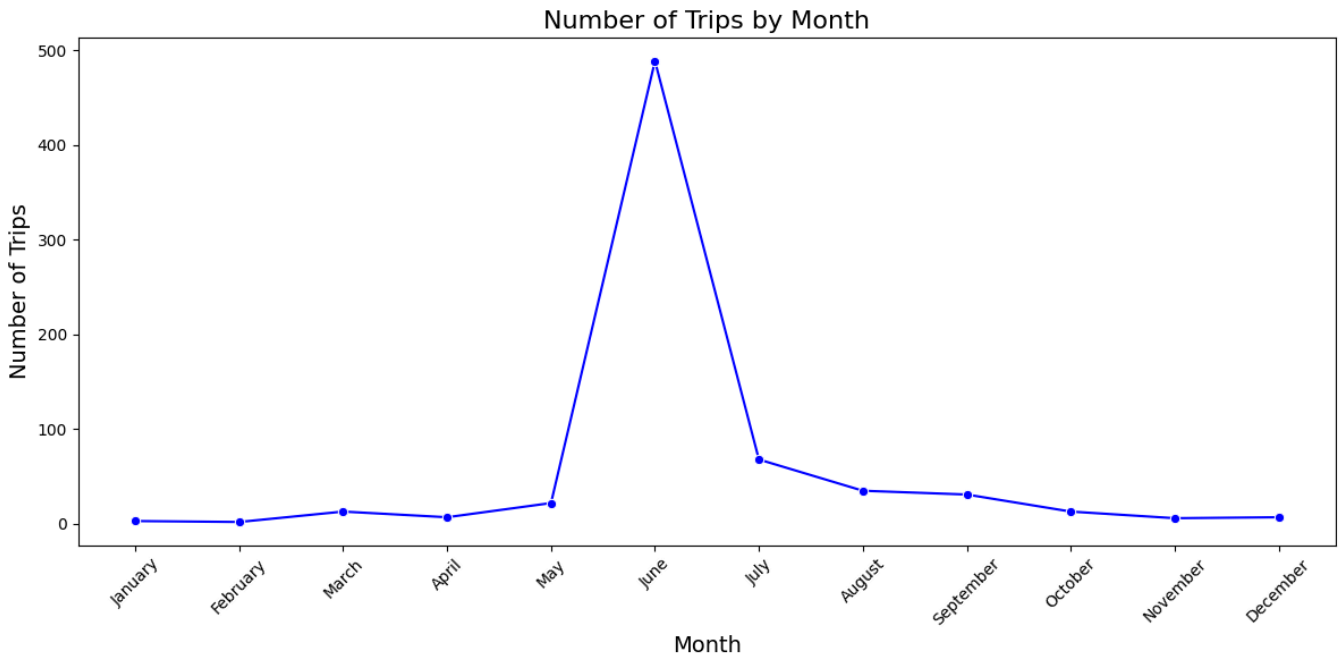
```
plt.xlabel('Day of the Month', fontsize=14)
plt.ylabel('Number of Trips', fontsize=14)

# Rotate the x-axis labels if necessary
plt.xticks(rotation=45)

# Show the plot
plt.tight_layout()
plt.show()
```




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

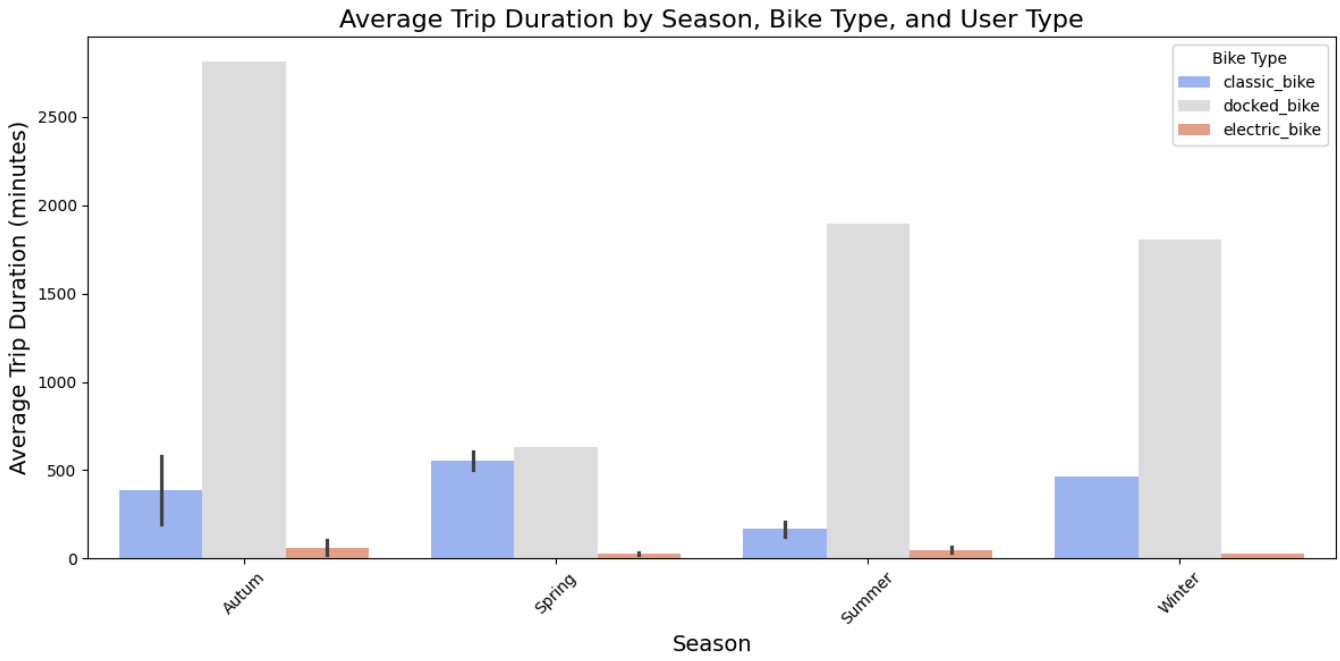
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming your data is loaded into a DataFrame 'df'
# Sample data loading (replace with your actual data loading process)
# df = pd.read_csv('your_bike_data.csv')

# Step 1: Ensure 'started_at' is in datetime format
df['started_at'] = pd.to_datetime(df['started_at'])

# Step 2: Extract day and month from 'started_at'
df['day'] = df['started_at'].dt.day
df['month'] = df['started_at'].dt.month_name()  # Month name for better readability

# Step 3: Count the number of trips by month
monthly_trip_count = df.groupby('month').size().reset_index(name='trip_count')

# Step 4: Sort the months by their order (from January to December)
month_order = ['January', 'February', 'March', 'April', 'May', 'June', 'July', 'August', 'September', 'October', 'November',
monthly_trip_count['month'] = pd.Categorical(monthly_trip_count['month'], categories=month_order, ordered=True)
monthly_trip_count = monthly_trip_count.sort_values('month')

# Step 5: Plotting the line chart for trip counts by month
plt.figure(figsize=(12, 6))
sns.lineplot(x='month', y='trip_count', data=monthly_trip_count, marker='o', color='blue')

# Add titles and labels
plt.title('Number of Trips by Month', fontsize=16)
plt.xlabel('Month', fontsize=14)
plt.ylabel('Number of Trips', fontsize=14)

# Rotate the x-axis labels for better readability if necessary
plt.xticks(rotation=45)

# Show the plot
```

```
plt.tight_layout()
plt.show()
```



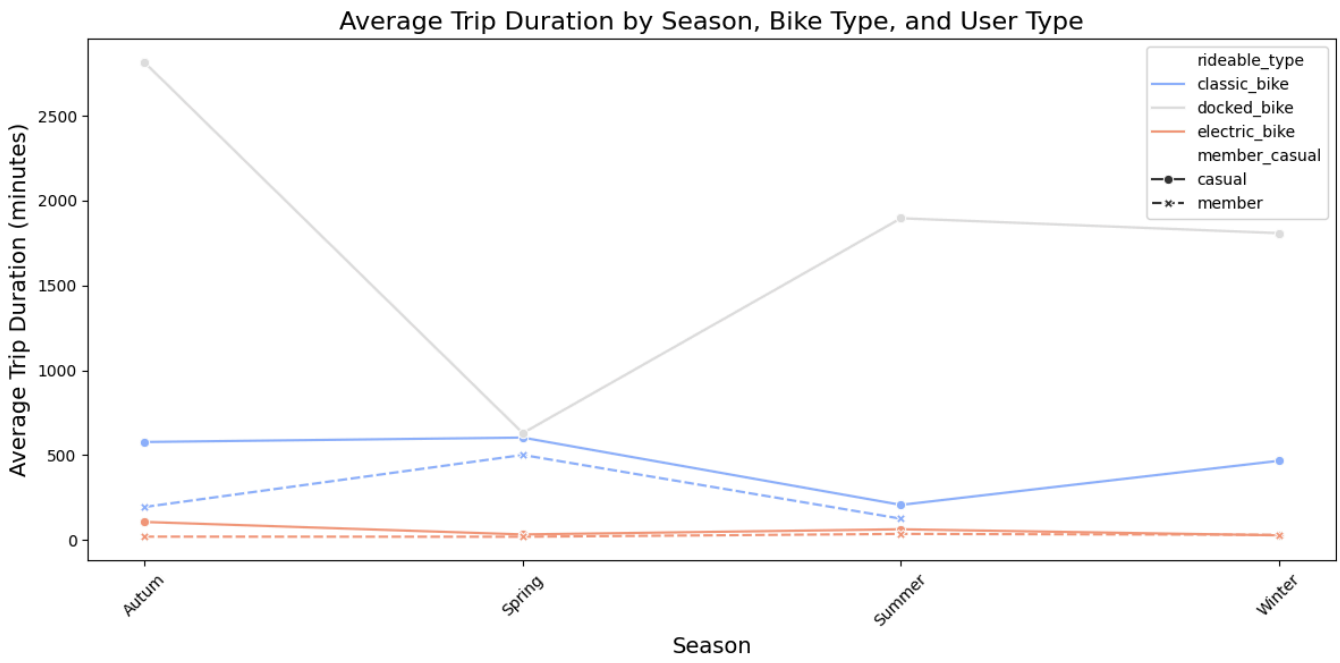

```
Start coding or generate with AI.
```

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Sample data loading (replace with your actual DataFrame)
# df = pd.read_csv('your_bike_data.csv')

# Step 1: Ensure 'started_at' is in datetime format
df['started_at'] = pd.to_datetime(df['started_at'])

# Step 2: Extract the season from the 'season' column (assuming 'season' is already available)
# If you don't have a 'season' column, create it based on the month (as an example)
# For example:
# df['season'] = pd.cut(df['started_at'].dt.month, bins=[0, 2, 5, 8, 11, 12], labels=['Winter', 'Spring', 'Summer', 'Autumn'

# Step 3: Group by 'season', 'rideable_type', and 'member_casual' to calculate the average trip duration
# Assuming you have columns: 'season', 'rideable_type' (bike type), 'member_casual' (user type), and 'trip_duration'
trip_duration_by_season_bike_user = df.groupby(['season', 'rideable_type', 'member_casual']).agg(
    average_duration=('trip_duration', 'mean')
).reset_index()

# Step 4: Plotting the bar chart with 'rideable_type' and 'member_casual' as hue
plt.figure(figsize=(12, 6))
sns.barplot(x='season', y='average_duration', hue='rideable_type',
            data=trip_duration_by_season_bike_user, palette='coolwarm')

# Add titles and labels
plt.title('Average Trip Duration by Season, Bike Type, and User Type', fontsize=16)
plt.xlabel('Season', fontsize=14)
plt.ylabel('Average Trip Duration (minutes)', fontsize=14)

# Rotate x-axis labels if needed
plt.xticks(rotation=45)

# Show the legend and the plot
plt.legend(title='Bike Type')
plt.tight_layout()
plt.show()
```



Start coding or generate with AI.

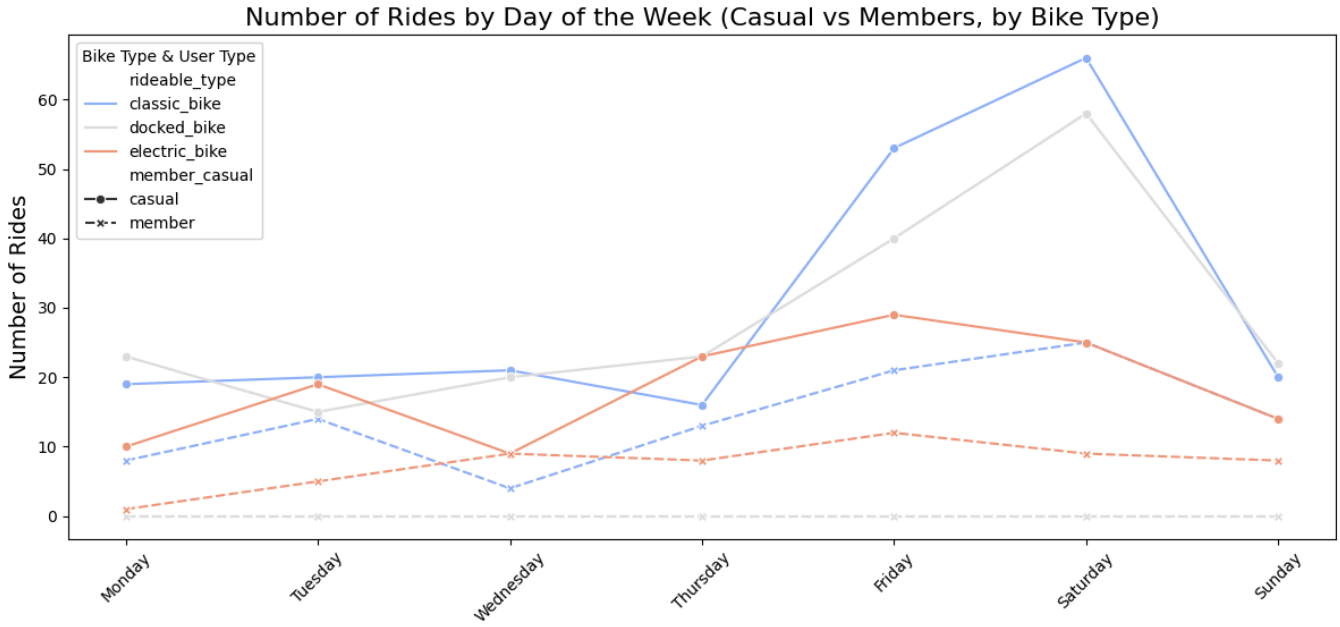
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Sample data loading (replace with your actual DataFrame)
# df = pd.read_csv('your_bike_data.csv')

# Step 1: Ensure 'started_at' is in datetime format
df['started_at'] = pd.to_datetime(df['started_at'])

# Step 2: Extract the season from the 'season' column (assuming 'season' is already available)
# If you don't have a 'season' column, create it based on the month (as an example)
# For example:
# df['season'] = pd.cut(df['started_at'].dt.month, bins=[0, 2, 5, 8, 11, 12], labels=['Winter', 'Spring', 'Summer', 'Autumn'

# Step 3: Group by 'season', 'rideable_type', and 'member_casual' to calculate the average trip duration
# Assuming you have columns: 'season', 'rideable_type' (bike type), 'member_casual' (user type), and 'trip_duration'
trip_duration_by_season_bike_user = df.groupby(['season', 'rideable_type', 'member_casual']).agg(
    average_duration=('trip_duration', 'mean')
).reset_index()

# Step 4: Plotting the line chart with hue for both 'rideable_type' (bike type) and 'member_casual' (user type)
plt.figure(figsize=(12, 6))
sns.lineplot(x='season', y='average_duration', hue='rideable_type', style='member_casual',
             data=trip_duration_by_season_bike_user, markers=True, palette='coolwarm')

# Add titles and labels
plt.title('Average Trip Duration by Season, Bike Type, and User Type', fontsize=16)
plt.xlabel('Season', fontsize=14)
plt.ylabel('Average Trip Duration (minutes)', fontsize=14)

# Rotate x-axis labels if needed
plt.xticks(rotation=45)

# Show the plot
plt.tight_layout()
plt.show()
```



Start coding or generate with AI.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming you have already loaded your DataFrame 'df'

# Step 1: Ensure 'started_at' is in datetime format
df['started_at'] = pd.to_datetime(df['started_at'])

# Step 2: Extract day of the week
df['day_of_week'] = df['started_at'].dt.day_name()

# Step 3: Order the days of the week for the x-axis
day_order = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
df['day_of_week'] = pd.Categorical(df['day_of_week'], categories=day_order, ordered=True)

# Step 4: Count the number of rides for each day of the week, for all users (casual and members) and bike types
rides_per_day_all = df.groupby(['day_of_week', 'member_casual', 'rideable_type']).agg(
    ride_count=('ride_id', 'count')
).reset_index()

# Step 5: Plotting the number of rides by day of the week, using 'member_casual' and 'rideable_type' as hues
plt.figure(figsize=(12, 6))
sns.lineplot(x='day_of_week', y='ride_count', hue='rideable_type', style='member_casual', data=rides_per_day_all, markers=Tr

# Add titles and labels
plt.title('Number of Rides by Day of the Week (Casual vs Members, by Bike Type)', fontsize=16)
plt.xlabel('Day of the Week', fontsize=14)
plt.ylabel('Number of Rides', fontsize=14)

# Rotate x-axis labels for better readability
plt.xticks(rotation=45)

# Show the legend and the plot
plt.legend(title='Bike Type & User Type')
plt.tight_layout()  # Adjust layout to prevent clipping of labels
plt.show()
```

```
<ipython-input-61-4022a9ad8ba1>:18: FutureWarning: The default of observed=False is deprecated and will be changed to Tr
  rides_per_day_all = df.groupby(['day_of_week', 'member_casual', 'rideable_type']).agg(
```




```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming you have already loaded your DataFrame 'df'

# Step 1: Ensure 'started_at' is in datetime format
df['started_at'] = pd.to_datetime(df['started_at'])

# Step 2: Extract the season from the 'season' column (assuming 'season' is already available)
# If you don't have a 'season' column, you can create one based on the month using pd.cut or manually categorize months
# Example: df['season'] = pd.cut(df['started_at'].dt.month, bins=[0, 2, 5, 8, 11, 12], labels=['Winter', 'Spring', 'Summer',

# Step 3: Group by 'season' and calculate the average trip duration
# Assuming 'trip_duration' is in minutes
trip_duration_by_season = df.groupby('season').agg(
    average_duration=('trip_duration', 'mean')
```