final_report_Ber_Marjan

December 14, 2024

1 Citi Bike Analysis: Understanding Ride Patterns in NYC

1.1 ## Marjan Abedini – Ber Bakermans

1.2 Motivation and Background

The bike-sharing program in New York City provides commuters and tourists with an eco-friendly and effective way of getting around the city .Bike-sharing initiatives like Citibike are essential to create better city livability and help in the context of growing urbanization and the need to lower carbon emissions. The need for more data-driven insights to help maximize system performance, user happiness, and operational efficiency is growing together with the demand for these bike-sharing services.

1.2.1 Why This Problem Matters

Companies for bike sharing are growing systems that are impacted by a number of variables, such as geographical trends, rider behavior, and weather. Addressing these factors is important for understanding the different challenges companies like Citi-Bike might be facing. Some of the challenges or improvements are:

1. **Operational Efficiency**: Operators can more efficiently distribute bikes and docks by identifying the busiest stations and periods of high usage. If they do this well it can reduce user discomfort brought on by shortages or crowding.
2. **User Experience**: By recognizing the differences in behavior between casual and member customers, Citi bike can adapt its services, this will help with increasing user retention and bring in new riders.
3. **Sustainability and Accessibility**: By promoting the growth of services to underserved areas, geographic and demographic insights can promote more access for everyone and reduce inequity in transportation.
4. **Weather Resilience**: By examining the effects of weather on riding, it can help to improve demand forecasting. This will ensure that the system can adjust to environmental challenges.

1.2.2 how to translate this to our project

Looking at the possible improvements for a bike sharing platform, there are some ways we can translate it to our project:

- **Station Optimization**: Identifying the busiest stations will help decisions on infrastructure improvements and resource allocation, ensuring availability during hours that are in high demand.

- **Ride Duration Insights**: Exploring the correlation between ride duration and the different bike type will help showing what bikes user need for either short or long rides.
- **Behavioral Analysis**: Insights into member and casual user behaviors can help promotional strategies by understanding the users and their needs better.
- **Temporal Demand Patterns**: understanding peak hours and days will help in figuring out when more bikes are needed, and this will help the systems reliability.
- **Weather-Adaptive Planning**: Weather-based usage patterns can lead to targeted messages such as rider notifications, more and improved safety measures, and adaptive pricing models.
- **Geographic Trends**: Mapping the bike stations provides a better geographic visualization and understanding of the high-demand areas. This can be helpful in decisions regarding to stations (upgrading/ new stations).

1.2.3 Impact besides Citi-Bike

In addition to helping Citi bike create a smoother operation, this analysis can also help bigger initiatives in sustainability and urban planning. Cities can reduce traffic, cut greenhouse gas emissions, and support a healthier lifestyle by improving the efficacy of bike-sharing systems.

2 Summary of Research Questions and Results

1. **Which stations are the most popular for starting and ending rides?**
   We want to find which stations have the most rides in 2024 to understand where people start and finish their trips.
   Answer: "W 21 St & 6 Ave" was the most popular station with 96,428 rides.

2. **What is the distribution of bike types (classic vs. electric) across the top 10 stations?**
   We want to know how often classic bikes and electric bikes are used at the busiest stations to see which type is more popular.
   Answer: Electric bikes are more popular than classic bikes, used by both casual and member riders.

3. **How does ride usage vary by month in 2024?**
   We aim to see how the number of rides changes each month to understand if bike usage is higher at certain times of the year.
   Answer: Ride numbers increased from July to September, with September having the most rides.

4. **What percentage of rides come from members versus casual users?**
   We want to find out how many rides come from people with memberships compared to casual riders.
   Answer: Members make up 75% of rides, while casual users make up 25%.

5. **Which stations are unique to casual users compared to members?**
   We aim to find out which stations are mostly used by casual riders, which might indicate more tourist or leisure areas.
   Answer: Stations like "Pier 61 at Chelsea Piers" are used more by casual riders, likely because they're near tourist spots.

6. **How does ride volume vary by time of day across top stations?**
   We want to understand when during the day the most rides happen to spot peak times at popular stations.
   Answer: Evening and morning hours have the highest ride volumes, likely due to commuting.

7. **What is the correlation between temperature and ride volume at top stations in September?**
   We want to see if warmer weather leads to more bike rides at the most popular stations.
   Answer: There is a moderate positive correlation between warmer weather and more rides.

---

3 Dataset Description

3.0.1 Dataset: Citibike Trips in New York City (2024)

The dataset provides detailed information on trips recorded by the Citibike bike-sharing service in New York City for July, August, and September 2024. It includes individual ride details, such as start and end stations, ride duration, bike type, and user type (member or casual). A different weather data set was merged as well to analyze the impact of environmental conditions on bike usage patterns.

3.0.2 Data Sources

1. **Citibike Trips Dataset**
   - **Source**: Citibike Data Portal
   - **URL**: https://www.citibikenyc.com/system-data
   - **Fields**:
     - `ride_id`: Unique identifier for each ride
     - `start_time`, `end_time`: Start and end timestamps of the ride
     - `start_station_name`, `end_station_name`: Stations where rides started and ended
     - `rideable_type`: Type of bike (`classic_bike`, `electric_bike`)
     - `member_casual`: Indicates if the user is a member or casual rider.
     - `duration`: Ride duration in minutes.

- **Time Period**: There are different dataset but the data we used covers ride records from the months of July, August, and September of 2024.

2. **Weather Data**
   - **Source**: Visual Crossing Weather History
   - **URL**: https://www.visualcrossing.com/weather-history/nyc
   - **Fields**:

  - `date`: Date of the observation
  - `temperature`: Daily average temperature (°F)
  - `precipitation`: Total rainfall (inches)
  - `humidity`: Average daily humidity (%)
- **Time Period**: The data includes hourly weather records for the month of September 2024.

3.0.3 merging the data

The Citibike trips data was combined from several files using shared columns. This was needed because one month's data had millions of trips, which was too much for a single file. Citibike splits its monthly data into smaller files, so they had to be merged to analyze everything together.

The Visual Crossing weather data was merged by adding an extra column based on the hour of the day. Since Citibike data is recorded by the minute and weather data by the hour, it was necessary to add this column to the Citibike data to match and combine both datasets.

---

4 Methodology: Analysis of Citibike Data with Weather Integration

4.0.1 Overview

Our analysis was conducted to answer research questions about Citibike usage patterns in New York City during the summer of 2024, integrating ride data with weather data to explore trends. Below is a step-by-step description of how we started and what methods we used.

4.0.2 1. Data Cleaning and Preprocessing

A. Citibike Data Preprocessing

1. **Load Data**:
   - Import the Citibike trips dataset into a DataFrame.
2. **change format**:
   - Convert `start_time` and `end_time` fields to datetime format.
3. **Calculate Trip Duration**:
   - Compute the duration of each trip in minutes as:
     [duration_minutes = (end_time - start_time).total_seconds() / 60]
4. **Filter Outliers**:
   - Remove trips with a duration less than 1 minute or greater than 300 minutes as these are likely errors or outliers.
5. **Categorize Users**:
   - Ensure the `member_casual` column is consistent and only includes `member` or `casual`.
6. **Data Cleaning**:

- **Duplicate Removal**: Removed any duplicate rows to avoid having the same rides multiple times.

- **Handling Missing Values**:
  - Removed rows with missing `start_station_name` or `end_station_name` as they represent critical information for analysis.

  - Replaced invalid or zero-length `duration` to dropped rows as these are invalid rides.

  - For missing values in other non-critical columns like `rideable_type`, imputed missing values with the mode (the most frequent value).

7. **Feature Engineering**:
   - Extracted additional features from the `start_time` column:
     - `hour`: Extracted the hour part of the start time to analyze ride patterns during the day.

     - `day_of_week`: Extracted the weekday from `start_time` to study rides by day.

     - `month`: Extracted the month from `start_time` to study seasonal trends in bike usage.
8. **Handling Ride Duration**:
   - Converted the ride duration to minutes for consistency and clarity. Any extreme values were filtered out (e.g., rides longer than 24 hours or with unrealistic durations).
9. **Aggregation for Summary Statistics**:
   - Grouped the data by `start_station_name`, `rideable_type`, `member_casual`, `day_of_week`, and other relevant features to compute aggregate statistics such as total ride counts, average ride duration, and average distance between stations.

B. Weather Data Preprocessing

1. **Load Data**:
   - Read five Citibike trip data files for September 2024.
   - Combine them into one large dataset using `pd.concat`.
   - Read the weather data for September 2024.
2. **Data Cleaning**:
   - Drop rows with missing values in critical columns like `start_station_name` and `end_station_name`.
   - Convert coordinate columns (`start_lat`, `start_lng`, `end_lat`, `end_lng`) to numeric format.
   - Remove duplicate rows based on `ride_id`.
     Date time
   - Convert the `started_at` and `ended_at` columns in Citibike data and the `datetime` column in weather data to datetime objects.
3. **Feature Engineering**:
   Extract Hour

- Extract the hour of the day from the `started_at` column in Citibike data.
- Extract the hour of the day from the `datetime` column in weather data. **Simplify Weather Data**
- Drop unnecessary weather columns to retain only relevant features. **Add Weekly Information**
- Add a `week` column to both datasets using ISO calendar weeks.
- Add a `week_of_month` column for both datasets, indicating which week the date falls in within the month.

C. Merging Citibike and Weather Data:

1. **Merge by Hour and Week**
   - The Citibike and weather datasets were merged based on the `hour` and `week` columns using a left join to keep all records from the Citibike dataset.
2. **Merge Daily Averages**
   - Daily weather averages (e.g., average temperature, humidity, wind speed) were computed and merged with the Citibike dataset on the `date` column.

4.0.3 2. Exploratory Data Analysis (EDA)

Overview Performed initial analyses to understand key patterns in the data:
- **Station Usage**: Identified the top 10 starting and ending stations by total rides.
- **Bike Type Preferences**: Compared usage of `classic_bike` and `electric_bike` by user type.
- **User Behavior**: Analyzed ride durations, preferences, and peak times for members and casual users.

4.0.4 3. Analysis by Research Question

Research Question 1: Top Stations in 2024

- **Objective**: Find the 10 most popular starting and ending stations.

- **Steps**:
  1. Grouped rides by `start_station_name` and calculated total rides.

  2. Repeated the process for `end_station_name`.

  3. Sorted the grouped data in descending order and extracted the top 10 for both.

Research Question 2: Ride Duration by Bike Type

- **Objective**: Compare average ride durations for classic and electric bikes.

- **Steps**:

1. Grouped data by `rideable_type`.

2. Computed the mean ride duration for each group.

Research Question 3: Member vs. Casual User Behavior

- **Objective**: Understand differences in usage patterns between members and casual riders.

- **Steps**:
  1. Grouped rides by `member_casual` and aggregated metrics like ride count and average duration.

  2. Compared station preferences and bike type choices for each user type.

Research Question 4: Temporal Analysis by Day and Hour

- **Objective**: Analyze ride frequency by hour, day of the week, and month.

- **Steps**:
  1. Extracted temporal fields (`hour`, `day_of_week`, `month`) from timestamps.

  2. Used pivot tables to calculate ride counts by these temporal dimensions.

Research Question 5: Weather Impact on Rides

- **Objective**: Investigate how weather conditions affect ride frequency.

- **Steps**:
  1. Merged weather data with Citibike data by date.

  2. Calculated correlations between ride counts and weather features (e.g., temperature, precipitation).

Research Question 6: Geographic Trends in Usage

- **Objective**: Explore regional variations in station usage and bike type preferences.

- **Steps**:
  1. Mapped stations to geographic coordinates using Folium.

  2. Aggregated usage metrics by region for comparative analysis.

4.0.5 4. Visualization Techniques

To better understand trends and patterns, various visualization tools and techniques were used:

1. **Folium Maps**: Visualized station popularity and geographic trends.

2. **Bar Plots**:
   - Compared ride durations by bike type and user type.
   - Visualized monthly variations in bike type usage across member and casual users using facet grids.

3. **Scatterplots and Treemaps**: Illustrated rides by day of the week for each month at top stations.

4. **Radar Plots**: Represented hourly ride patterns for each month.

5.

4.1 Heatmaps: Showed ride frequencies by day and weekday for September 2024.

5 Results

1. **Which stations are the most popular for starting and ending rides?**

Through our analysis of ride data for 2024, we identified the most frequently used stations by riders. By visualizing the data, we pinpointed the top stations and compared their ride volumes. All the top 10 stations recorded over 80,000 rides during the 3-month analysis period.

The most popular station in 2024 for both starting and ending rides was "W 21 St & 6 Ave" with 96,428 rides. Other busy stations included "West St & Chambers St" (93,247 rides) and "8 Ave & W 31 St" (89,865 rides). These three stations are located in key areas of Manhattan, where many people live, work, or commute, which explains the high bike activity.

The other top stations were: 4. Lafayette St & E 8 St 5. Broadway & W 58 St 6. Broadway & E 14 St 7. Pier 61 at Chelsea Piers 8. 11 Ave & W 41 St 9. University Pl & E 14 St 10. W 31 St & 7 Ave

When we mapped these stations, we noticed that most of them are concentrated in high-activity areas, including commercial hubs and popular tourist attractions. This further reinforces why these stations experience such high ride volumes.

Key Findings:

"W 21 St & 6 Ave" was the most popular station, with 96,428 rides in 2024, followed by "West St & Chambers St" (93,247 rides) and "8 Ave & W 31 St" (89,865 rides). All top 10 stations recorded over 80,000 rides during the 3-month analysis period.

Implications:

These high-volume stations are concentrated in Manhattan's commercial hubs and tourist spots. This can help city planners and bike share programs focus resources on the most used stations to ensure availability and maintain efficiency.

2. **What is the distribution of bike types (classic vs. electric) across the top 10 stations?**

Our results showed that electric bikes are used significantly more than classic bikes. Electric bikes make up a large portion of all rides, with both casual and member users favoring them. We believe this trend is due to the increasing popularity of electric bikes in recent years, as they offer a more convenient and easier riding experience. On the other hand, classic bikes see more consistent use among members, who tend to rely on them for regular commuting and daily trips.

Key Findings: Electric bikes were used significantly more than classic bikes, across both casual and member riders.

Implications: This trend may reflect a shift toward more accessible and comfortable transportation options in urban areas. It suggests that bike share programs should prioritize maintaining a larger amount of electric bikes to meet user demand.

3. **How does ride usage vary by summer months in 2024?**

Our analysis of monthly ride data shows a steady increase in usage from July to September. In July, we observed moderate ride volumes, which we believe could be due to summer vacations and higher temperatures, leading to fewer people using bikes. September, however, recorded the highest number of riders out of the three months, likely because of cooler weather and people returning to their regular routines.

Key Findings: Ride volumes steadily increased from July to September, with September showing the highest usage.

Implications: The seasonal fluctuations in usage could help bike share programs better prepare for peak demand periods, ensuring more bikes are available in popular areas during the fall months.

4. **What percentage of rides come from members versus casual users?**

By grouping the data by user type, we found that members make up about 75% of all the rides, while casual users account for the remaining 25%. This shows that members rely more on Citi Bike for consistent transportation. It makes sense because, as members, they likely expect to use the service more regularly, especially given their subscription.

Key Findings: Members accounted for 75% of all rides, while casual users made up 25%.

Implications: Citi Bike may want to focus on keeping members by offering incentives or improving their experience, while also targeting casual riders with more appealing options for one-time or infrequent usage.

5. **Which stations are unique to casual users compared to members?**

We found that stations like "Pier 61 at Chelsea Piers" and "Broadway & W 58 St" are very popular with casual users but are not used as much by members. When we mapped these stations, we saw that they are located near tourist attractions and scenic routes. This supports our observation that casual riders mostly use the bikes for leisure, enjoying the sights and the experience rather than for daily commuting.

Key Findings:

Stations like "Pier 61 at Chelsea Piers" and "Broadway & W 58 St" were more popular with casual users.

Implications:

Casual riders are more likely to use bikes for recreational purposes, which could influence the placement of stations and marketing strategies aimed at tourists or leisure riders.

6. **How does ride volume vary by time of day across top stations?**

Evening hours (5–6 PM) consistently show the highest ride volumes at the top stations, and morning hours (7–9 AM) also see significant activity. This can be linked to the commuting habits of members, who make up nearly 75% of the rides. It makes sense that these times are when bikes are used the most, as many members rely on Citi Bike for their daily commutes.

Key Findings: The highest ride volumes occurred during evening hours (5–6 PM) and morning hours (7–9 AM).

Implications: Bike share programs can optimize station placements, availability, and bike redistribution during these peak times to ensure efficient service during commuting hours.

7. **What is the correlation between temperature and ride volume at top stations in September?**

Our analysis found a moderate positive correlation of 0.51 between temperature and ride volume. This means that on warmer days, Citi Bike tends to see more rides. However, since the correlation isn't very strong, it indicates that other factors, besides temperature, also influence the number of rides.

Key Findings:

A moderate positive correlation of 0.51 between temperature and ride volume was found.

Implications: Temperature is a significant, but not the sole factor, in driving bike usage. This suggests that external factors might also play a role in usage trends. Further analysis could explore these factors to refine predictions.

6 Next steps

6.0.1 Next Steps Based on Further Analysis

1. **Demographics of Bike Users**
   We could look at who is using the bikes based on age, gender, and location. This would help us understand which groups use bikes the most and if Citi-bike needs to make any changes, like offering special deals for certain groups.

2. **Effect of Popular Places on Bike Usage**
   We could study how nearby popular places (like parks, tourist spots, or busy areas) affect how many people use bikes. This can help Citi-bike decide where to put more bike stations or have bikes available where people want them the most.

3. **Impact of Holidays and Events**
   We can check how big events (like concerts, festivals, or sports games) and holidays change the number of bike rides. Knowing this would help plan better for these times and possibly add more bikes or stations where they are needed.

4. **Expanding to Other Bike-Sharing Services**
   We could compare how Citi Bike usage matches up with other bike-sharing services in the city or elsewhere. This would help us see if there are ways to work together or offer better bike coverage for riders across different services.
5. **Time of Day and Weather Effects**
   Another step could be to study how different weather conditions (like rain or heat) affect bike usage during different times of the day. This could help Citi bike plan better for distribution and offer deals to encourage bike use even in bad weather.
6. **Improving the Bike Network**
   We can also do more research on how well the current bike stations are placed. By looking at how far people travel between stations, we can find areas that might need more stations or bikes, making the service more convenient for everyone.

7 Analysis and Results of Linear Regression Model

7.0.1 Overview

We decided that after the lecture, we wanted to try using a linear regression as well. We decided to use the `merged_data_sep` for this. We wanted to see if we could predict the duration of the rides based on the positive correlation variables we saw in our correlation matrix. We decided to go with our features that are: `'avg_temp', 'weekday', 'hour', 'start_lat', 'avg_wind_speed', 'end_lat', 'end_lng'`.

We already saw earlier that the correlation of weather with trip amounts was around 0.51, which is not a strong correlation. So we wanted to see if this would work.

7.0.2 Results

As a result, we got: - **MAE** = 496.91 - **MSE** = 1,211,172.90 - **R²** = 0.00267

These results weren't the greatest ones. This means that our model is off by 497 seconds for every trip. It has an average mean squared error of 1,211,172.90, which is really large and indicates significant prediction errors. Our R^2 is 0.27%, which means that the model only explains 0.27% of the variance in the target variable.

7.0.3 Conclusion

So overall, `'avg_temp', 'weekday', 'hour', 'start_lat', 'avg_wind_speed', 'end_lat', 'end_lng'` are not great predictors for trip duration.

---

8 Reflection

8.0.1 what we learned from this assignment

We learned how to analyze bike-sharing data to find patterns, such as which stations are most popular, how bike usage changes by month, and what time of day people use bikes the most. We also learned how to use different data analysis techniques to get useful insights about bike usage based on different factors

8.0.2 what we wish we knew before starting

We wish we had a better plan from the beginning, with clearer research questions and a more focused approach. This would have helped guide our analysis and made the process smoother. Also, we didn't realize how challenging it could be to work with large datasets in jupyterlab, so knowing more about how to manage big data before starting would have been helpful.

8.0.3 what we would do differently

If we could do this again, we would start by defining specific research questions to guide our analysis. We would also choose a smaller dataset that still provides enough useful information, as working with a large dataset caused our system to slow down or crash. Having a more focused approach from the beginning would have made the project more manageable and efficient.

---

9 Citi Bike Data Preprocessing Documentation

We consolidated Citibike trip data from multiple CSV files for the months of July, August, and September 2024 into a single dataset. By defining the file paths for each month's data, reading each file into a DataFrame, and appending them together, we created a comprehensive dataset. Additionally, we included error handling to manage missing or problematic files. Finally, the combined dataset was saved as a new CSV file for further analysis. This approach ensured a streamlined and efficient process for merging the data.

9.1 1. Loading the Dataset

- The dataset `combined(1)_citibike_tripdata.csv` is loaded into a Pandas DataFrame `bike_df`.
- The `low_memory=False` option is used to optimize the loading process for this large dataset.

[9]:
```
import warnings
warnings.filterwarnings('ignore')
```

[10]:
```
import pandas as pd
bike_df = pd.read_csv('combined(1)_citibike_tripdata.csv', low_memory=False)
```

[11]:
```
# Display the first few rows of the combined DataFrame
bike_df.head()
```

[11]:
```
   Unnamed: 0           ride_id  rideable_type               started_at  \
0         0.0  A3818B07E831C033  electric_bike  2024-07-01 21:00:55.640
1         1.0  40B46D3F898A6CA4  electric_bike  2024-07-09 18:28:39.551
2         2.0  33683B345D08C2BC  electric_bike  2024-07-14 17:10:28.899
3         3.0  B213C077CE572EBD  electric_bike  2024-07-01 11:37:06.440
4         4.0  E42FF50D966A009B  electric_bike  2024-07-05 16:55:59.093
```

```
                  ended_at      start_station_name start_station_id  \
0  2024-07-01 21:07:43.553         W 24 St & 7 Ave          6257.03   
1  2024-07-09 18:46:43.854   Park Pl & Buffalo Ave          3999.06   
2  2024-07-14 17:27:59.917         E 48 St & 5 Ave          6626.01   
3  2024-07-01 11:42:02.259  Melrose Ave & E 154 St          7918.12   
4  2024-07-05 16:59:41.250         W 24 St & 7 Ave          6257.03   

             end_station_name end_station_id  start_lat  start_lng    end_lat  \
0            W 20 St & 10 Ave        6306.01  40.744784 -73.995524  40.745686   
1      Irving Ave & Harman St        4856.05  40.671992 -73.925502  40.701080   
2       E 43 St & Madison Ave        6551.11  40.757246 -73.978059  40.753547   
3        Wales Ave & E 147 St        7751.05  40.819665 -73.916111  40.811314   
4  Greenwich Ave & Charles St        5914.08  40.744817 -73.995416  40.735238   

     end_lng member_casual  Month rideable_type_duplicate_column_name_1  
0 -74.005141        member      7                                   NaN  
1 -73.917900        member      7                                   NaN  
2 -73.978966        casual      7                                   NaN  
3 -73.907729        member      7                                   NaN  
4 -74.000271        member      7                                   NaN  
```

[12]:
```
bike_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14324369 entries, 0 to 14324368
Data columns (total 16 columns):
 #   Column                                 Dtype  
---  ------                                 -----  
 0   Unnamed: 0                             float64
 1   ride_id                                object 
 2   rideable_type                          object 
 3   started_at                             object 
 4   ended_at                               object 
 5   start_station_name                     object 
 6   start_station_id                       object 
 7   end_station_name                       object 
 8   end_station_id                         object 
 9   start_lat                              float64
 10  start_lng                              float64
 11  end_lat                                float64
 12  end_lng                                float64
 13  member_casual                          object 
 14  Month                                  int64  
 15  rideable_type_duplicate_column_name_1  object 
dtypes: float64(5), int64(1), object(10)
memory usage: 1.7+ GB
```

```
[13]: bike_df['Month'].unique()
```

```
[13]: array([7, 8, 9], dtype=int64)
```

```
[14]: bike_df = bike_df.drop(columns=["Unnamed:␣
       ↪0","rideable_type_duplicate_column_name_1"])
```

9.2 2. Data Type Conversion Steps

1. **Converting the `Month` Column to Integer**:
    - Ensures valid integer values by handling non-numeric entries gracefully with `pd.to_numeric(errors='coerce')`.
    - Uses the nullable integer type `Int64` to maintain compatibility with missing values (`NaN`).
2. **Converting Categorical Columns to `category`**:
    - Optimizes memory usage for columns like `member_casual` and `rideable_type` by storing repeated values as integer codes.
    - Improves the performance of operations such as grouping, filtering, and aggregation.
    - Provides semantic clarity, highlighting that these columns represent a finite set of distinct groups.

```
[16]: # Convert to integer, coercing errors to handle non-numeric values gracefully
      bike_df["Month"] = pd.to_numeric(bike_df["Month"], errors='coerce').
       ↪astype('Int64')
      # Convert the relevant columns to categorical types
      bike_df['member_casual'] = bike_df['member_casual'].astype('category')
      bike_df['rideable_type'] = bike_df['rideable_type'].astype('category')
```

```
[17]: bike_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14324369 entries, 0 to 14324368
Data columns (total 14 columns):
 #   Column              Dtype
---  ------              -----
 0   ride_id             object
 1   rideable_type       category
 2   started_at          object
 3   ended_at            object
 4   start_station_name  object
 5   start_station_id    object
 6   end_station_name    object
 7   end_station_id      object
 8   start_lat           float64
 9   start_lng           float64
 10  end_lat             float64
 11  end_lng             float64
 12  member_casual       category
 13  Month               Int64
```

```
dtypes: Int64(1), category(2), float64(4), object(7)
memory usage: 1.3+ GB
```

[18]:
```
bike_df.describe()
```

[18]:
```
          start_lat     start_lng       end_lat       end_lng        Month
count  1.432437e+07  1.432437e+07  1.432099e+07  1.432099e+07   14324369.0
mean   4.073760e+01 -7.397109e+01  4.073721e+01 -7.397061e+01     8.019198
std    4.108766e-02  2.939902e-02  1.132144e-01  1.937733e-01      0.82356
min    4.054541e+01 -7.404000e+01  0.000000e+00 -7.629000e+01          7.0
25%    4.071263e+01 -7.399203e+01  4.071260e+01 -7.399209e+01          7.0
50%    4.073568e+01 -7.397897e+01  4.073555e+01 -7.397897e+01          8.0
75%    4.076096e+01 -7.395412e+01  4.076088e+01 -7.395412e+01          9.0
max    4.091000e+01 -7.378000e+01  4.135000e+01  0.000000e+00          9.0
```

9.3 3. Handling Missing Values

[20]:
```
bike_df.isnull().sum()
```

[20]:
```
ride_id                   0
rideable_type             0
started_at                0
ended_at                  0
start_station_name    10062
start_station_id      10062
end_station_name      38812
end_station_id        41367
start_lat                 0
start_lng                 0
end_lat                3380
end_lng                3380
member_casual             0
Month                     0
dtype: int64
```

9.3.1 Filling Missing Categorical Values

- Columns such as `start_station_name`, `start_station_id`, `end_station_name`, and `end_station_id` often contain missing data.
- These are filled with the placeholder value `"Unknown"`. This ensures:
  - Consistent data types for the columns.
  - Missing values do not disrupt analyses or aggregations.

9.3.2 Filling Missing Numeric Values

- Columns like `end_lat` and `end_lng` contain numeric data, and missing values are replaced with their respective column means. This approach:
  - Prevents data loss by maintaining all rows.

– Avoids introducing extreme outliers compared to filling with a fixed value.

[22]:
```
# Filling categorical columns with 'Unknown'
bike_df['start_station_name'] = bike_df['start_station_name'].fillna('Unknown')
bike_df['start_station_id'] = bike_df['start_station_id'].fillna('Unknown')
bike_df['end_station_name'] = bike_df['end_station_name'].fillna('Unknown')
bike_df['end_station_id'] = bike_df['end_station_id'].fillna('Unknown')

# Filling numeric columns with mean
bike_df['end_lat'] = bike_df['end_lat'].fillna(bike_df['end_lat'].mean())
bike_df['end_lng'] = bike_df['end_lng'].fillna(bike_df['end_lng'].mean())
```

9.3.3 Dictionary-Based Filling

- Using a dictionary of column-value pairs (`fill_values`) allows for efficient and scalable handling of missing data in both categorical and numeric columns.
- The `fillna()` method with the `inplace=True` argument directly modifies the DataFrame.

[24]:
```
# Define the fill values for each column
fill_values = {
    'start_station_name': 'Unknown',
    'start_station_id': 'Unknown',
    'end_station_name': 'Unknown',
    'end_station_id': 'Unknown',
    'end_lat': bike_df['end_lat'].mean(),
    'end_lng': bike_df['end_lng'].mean()
}

# Apply fillna using the dictionary
bike_df.fillna(value=fill_values, inplace=True)
bike_df.isnull().sum()
```

[24]:
```
ride_id               0
rideable_type         0
started_at            0
ended_at              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
Month                 0
dtype: int64
```

9.4 4. Converting Date-Time Columns

The `started_at` and `ended_at` columns are originally stored as strings, which limits their usability for time-based calculations. By converting them to the `datetime` type using `pd.to_datetime`, the following benefits are achieved: - Enables operations like time differences, filtering by date, or grouping by month. - Improves data integrity by ensuring valid date-time formatting.

[26]:
```
# Convert 'started_at' and 'ended_at' to datetime
bike_df['started_at'] = pd.to_datetime(bike_df['started_at'])
bike_df['ended_at'] = pd.to_datetime(bike_df['ended_at'])
```

[27]:
```
# Check for duplicates
duplicates = bike_df.duplicated().sum()
print(f"\nNumber of duplicate rows: {duplicates}")
if duplicates > 0:
    bike_df.drop_duplicates(inplace=True)
```

```

Number of duplicate rows: 0
```

[28]:
```
bike_df.dtypes
```

[28]:
```
ride_id                       object
rideable_type               category
started_at            datetime64[ns]
ended_at              datetime64[ns]
start_station_name            object
start_station_id              object
end_station_name              object
end_station_id                object
start_lat                    float64
start_lng                    float64
end_lat                      float64
end_lng                      float64
member_casual               category
Month                          Int64
dtype: object
```

9.4.1 Identify the Most Popular Stations

- Determine which bike stations are the most frequently used.
- Analyze their geographic distribution to understand user concentration in different areas of the city.

9.4.2 Analyze Usage by Day of the Week

- Examine the variation in rides taken on different days.
- Identify peak usage days to inform operational decisions.

9.4.3 Evaluate Time of Day Patterns

- Investigate how the frequency of rides changes throughout the day.
- Identify peak hours for bike usage to optimize availability and resource allocation.

9.5 ### Analyze Citibike ride data and weather data

10 Aggregating and Analyzing Station Trips for 2024

10.1 Analyzing the Most Popular Stations

1. Aggregate Trips Starting and Ending at Each Station -Objective: Count the number of trips that start and end at each station.
   -Steps:

   -Use groupby('start_station_name') to group the data by starting station and calculate the count of trips for each station using .size().
   -Similarly, group by end_station_name to calculate the count of trips ending at each station.
   -Convert the results into DataFrames with columns start_station_name and Starts_2024 for starting trips, and end_station_name and Ends_2024 for ending trips.

2. Merge Start and End Counts -Objective: Combine the start and end counts into a single DataFrame.
   -Steps:

   -Use pd.merge() to perform an outer join on start_station_name and end_station_name.
   -This ensures that all stations are included, even if they only appear as starting or ending stations.
   -Missing values (NaN) in either column are replaced with 0 using .fillna(0).

3. Compute Total Trips for Each Station -Objective: Calculate the total trips for each station by adding the start and end counts.
   -Steps:

   -Create a new column Total_2024 that sums up Starts_2024 and Ends_2024.

4. Sort and Select the Top 10 Stations -Objective: Identify the top 10 stations based on total trips.
   -Steps:

   -Select only the start_station_name and Total_2024 columns for clarity.
   -Use .sort_values() to sort the stations in descending order of Total_2024.
   -Retrieve the top 10 stations using .head(10).

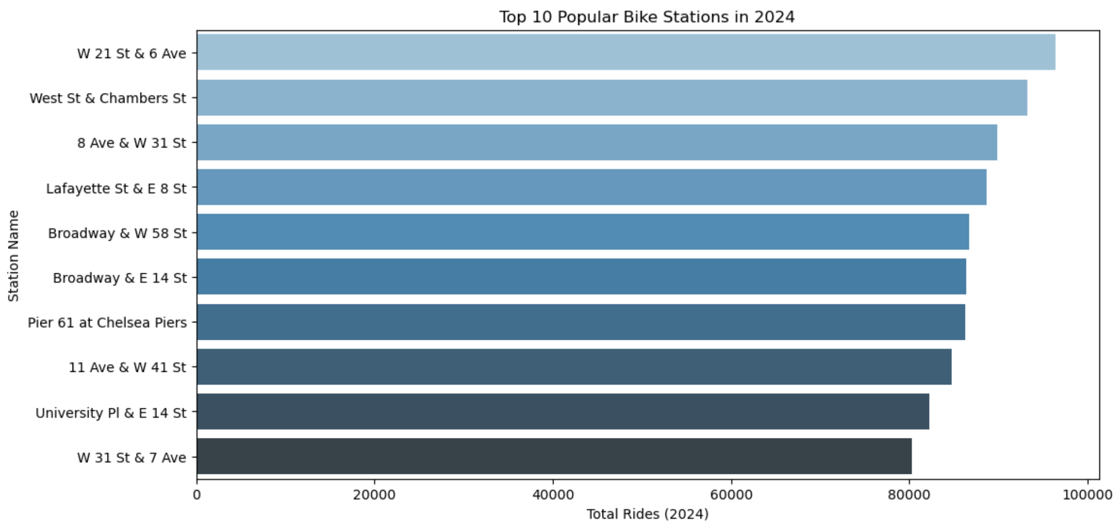
[32]:
```python
# Aggregate trips starting and ending at each station for 2024
start_counts_2024 = bike_df.groupby('start_station_name').size().
  ↪reset_index(name='Starts_2024')
end_counts_2024 = bike_df.groupby('end_station_name').size().
  ↪reset_index(name='Ends_2024')

# Merge start and end counts for 2024
popular_stations_2024 = pd.merge(
    start_counts_2024,
    end_counts_2024,
    left_on='start_station_name',
    right_on='end_station_name',
    how='outer'
).fillna(0)

# Add a Total column to compute the total trips for each station
popular_stations_2024['Total_2024'] = popular_stations_2024['Starts_2024'] +␣
 ↪popular_stations_2024['Ends_2024']

# Sort by total trips and get the top 10 stations
top_stations_2024 = (
    popular_stations_2024[['start_station_name', 'Total_2024']]
    .sort_values(by='Total_2024', ascending=False)
    .head(10)
)

# Display the top 10 stations for 2024
print("\nTop 10 Stations in 2024:")
print(top_stations_2024)
```

```
Top 10 Stations in 2024:
            start_station_name  Total_2024
2043               W 21 St & 6 Ave     96428.0
2180         West St & Chambers St     93247.0
458                8 Ave & W 31 St     89865.0
1434         Lafayette St & E 8 St     88678.0
689             Broadway & W 58 St     86682.0
651             Broadway & E 14 St     86412.0
1706      Pier 61 at Chelsea Piers     86311.0
40                11 Ave & W 41 St     84706.0
1960       University Pl & E 14 St     82257.0
2063               W 31 St & 7 Ave     80313.0
```

10.2 Visualization of Top 10 Popular Bike Stations for 2024

[34]:
```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

# Plotting Top 10 Popular Stations for 2024
plt.figure(figsize=(12, 6))

# Plot for 2024 with hue assigned to 'start_station_name'
```

```
sns.barplot(data=top_stations_2024, x='Total_2024', y='start_station_name',␣
↪hue='start_station_name', palette='Blues_d', legend=False)

# Title and labels for 2024 plot
plt.title("Top 10 Popular Bike Stations in 2024")
plt.xlabel("Total Rides (2024)")
plt.ylabel("Station Name")

# Show plot
plt.show()
```



10.3 Interactive Map of NYC's Top 10 Bike Stations by Total Rides in 2024

-The interactive map will show NYC with blue markers at the locations of the top 10 stations.
-Clicking a marker will display the station name and total number of rides in 2024.
-The map allows zooming and panning for detailed exploration of station locations.

[37]:
```
import folium

# filter the top stations and extract relevant data
top_stations_2024 = top_stations_2024.merge(
    bike_df[['start_station_name', 'start_lat', 'start_lng']],
    on='start_station_name',
    how='left'
).drop_duplicates(subset='start_station_name')

# Create a folium map centered around New York (approximate center of NYC)
nyc_map = folium.Map(location=[40.7128, -74.0060], zoom_start=12)
```

```
# Add markers for each of the top stations
for _, row in top_stations_2024.iterrows():
    folium.Marker(
        location=[row['start_lat'], row['start_lng']],
        popup=row['start_station_name'] + '\nTotal Rides: ' +␣
  ↪str(int(row['Total_2024'])),
        icon=folium.Icon(color='blue', icon='info-sign')
    ).add_to(nyc_map)

# Show the map
nyc_map
```

[37]:
```
<folium.folium.Map at 0x1d65b3fff80>
```

10.4 Analysis of Rideable Types and User Types for Top Stations in 2024

1. Combining Top Stations from 2024
2. Extracting Names of Top Stations
3. Filtering Data for Top Stations -Purpose: Select only the rides that started at one of the top stations from the bike_df DataFrame.
   -Condition: The start_station_name column must match one of the station names in top_station_names.
4. Grouping by Rideable Type and Member Type -Purpose: Group rides by:

   -rideable_type (type of bike used, e.g., electric or standard).
   -member_casual (user type: member or casual rider).

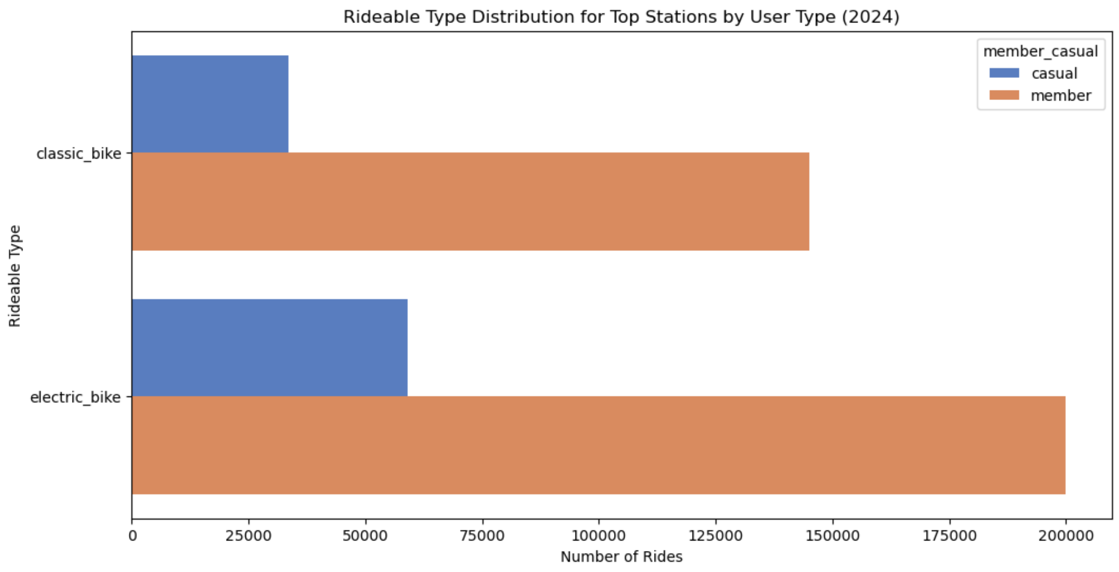
-Steps:

```
-Use .groupby() with ['rideable_type', 'member_casual'] to create groups.
-Use .size() to count the number of rides in each group.
-Use .reset_index(name='Counts') to convert the group counts into a DataFrame with a column nam
```

[39]:
```
# Combine top stations from 2024
top_stations = top_stations_2024

# Filter the data for the top stations
top_station_names = top_stations['start_station_name'].tolist()

# Filter bike_df for the top stations
filtered_data = bike_df[bike_df['start_station_name'].isin(top_station_names)]

# Count rides by rideable_type and member_casual
usage_summary = filtered_data.groupby(['rideable_type', 'member_casual'],␣
  ↪observed=False).size().reset_index(name='Counts')
```

```python
# Display the summary
print(usage_summary)

# Plotting the counts for the top stations
plt.figure(figsize=(12, 6))

# Plot for top stations
sns.barplot(data=usage_summary, x='Counts', y='rideable_type',␣
↪hue='member_casual', palette='muted')

# Title and labels
plt.title("Rideable Type Distribution for Top Stations by User Type (2024)")
plt.xlabel("Number of Rides")
plt.ylabel("Rideable Type")

# Show plot
plt.show()
```

```
   rideable_type member_casual  Counts
0   classic_bike        casual   33464
1   classic_bike        member  145089
2  electric_bike        casual   59135
3  electric_bike        member  199927
```



-Usage Summary Table:

`-A DataFrame showing the count of rides for each combination of rideable_type and member_casua`

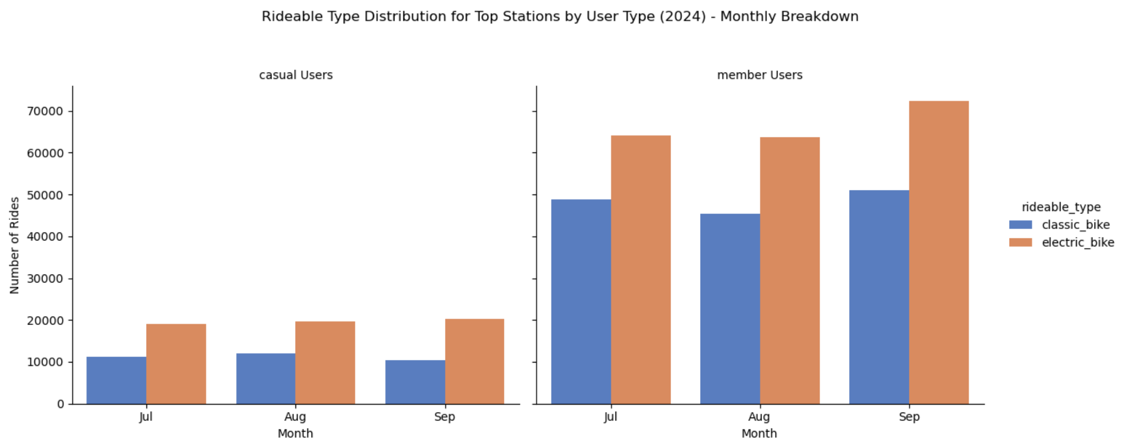
-Barplot:

```
-The barplot visualizes the distribution of rides by:
    Bike type (rideable_type) on the Y-axis.
    User type (member_casual) differentiated by hue.
    Total ride counts (Counts) on the X-axis.
-The plot helps analyze how different user types utilize bike types at the top stations.
```

10.5 Monthly Rideable Type Trends by User Type (2024)

The code aggregates ride counts by month, rideable type, and user type into a summary DataFrame. A Seaborn catplot creates a facet grid of bar plots, with each column representing a user type (member or casual). Bars are colored by rideable type, and the X-axis shows months (Jul, Aug, Sep). Titles, axis labels, and custom tick labels enhance clarity. The layout is optimized with tight_layout, providing a clear visualization of rideable type preferences across user groups and months in 2024.

```
[43]: import seaborn as sns
import matplotlib.pyplot as plt

usage_summary1 = filtered_data.groupby(['Month', 'rideable_type',␣
 ↪'member_casual']).size().reset_index(name='Counts')

# Plot: Facet grid with bar plots for rideable type across months by user type
g = sns.catplot(
    data=usage_summary1,
    x='Month',
    y='Counts',
    hue='rideable_type',
    col='member_casual',
    kind='bar',
    palette='muted',
    height=5,
    aspect=1.2
)

# Titles and labels
g.fig.suptitle("Rideable Type Distribution for Top Stations by User Type (2024)␣
 ↪- Monthly Breakdown", y=1.05)
g.set_axis_labels("Month", "Number of Rides")
g.set_titles("{col_name} Users")
g.set_xticklabels(["Jul", "Aug", "Sep"])


# Adjust layout and display
g.tight_layout()
plt.show()
```



10.6 Rides by Day of the Week for Top Stations (2024)

The code filters bike_df to include only trips starting at the top 10 stations. A new column, day_of_week, is extracted from the started_at timestamp to capture the weekday name. Rides are then grouped by Month, day_of_week, rideable_type, and member_casual, counting the occurrences. The results are stored in a pivot-like DataFrame with size().unstack(fill_value=0) to handle missing combinations gracefully. To ensure proper weekday order, the data is reindexed using the sequence from Monday to Sunday. This analysis provides detailed insights into weekday ride trends for the top stations across months in 2024.

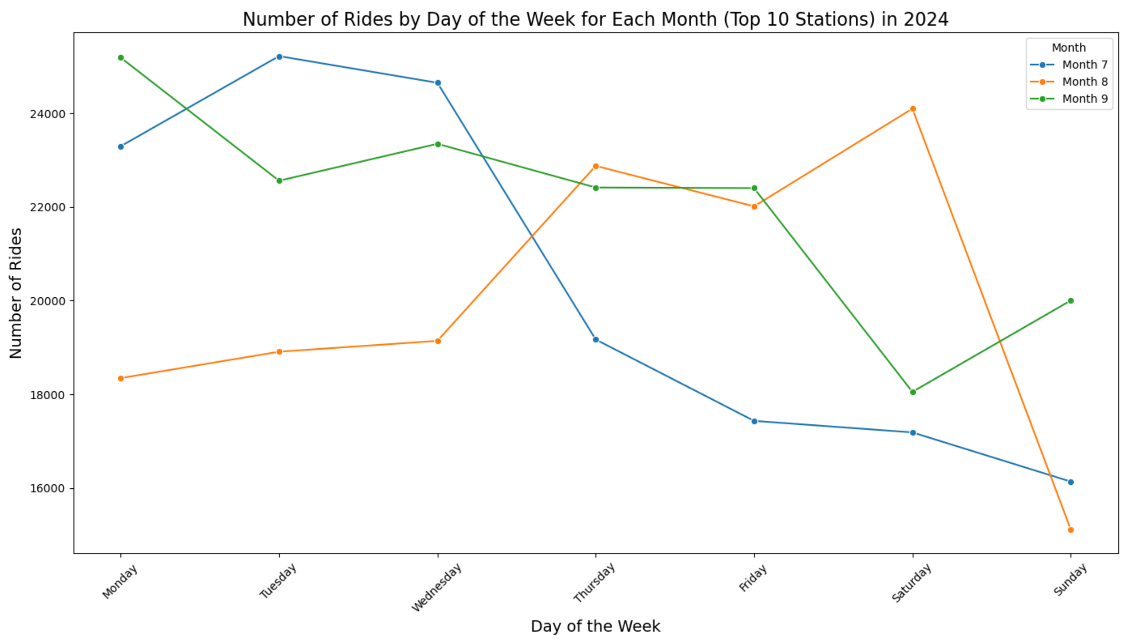
[46]:
```
# Filter the data for the top 10 stations
top_station_names = top_stations['start_station_name'].tolist()
filtered_top_stations = bike_df[bike_df['start_station_name'].
  ↪isin(top_station_names)]

# Add a column for the day of the week from the 'started_at' column
filtered_top_stations['day_of_week'] = filtered_top_stations['started_at'].dt.
  ↪day_name()

# Group by month, day of the week, rideable type, and member type, then count␣
  ↪the rides
# Explicitly set observed=False to avoid the warning
rides_by_day_month_combined_top_stations = filtered_top_stations.groupby(
    ['Month', 'day_of_week', 'rideable_type', 'member_casual'],
    observed=False
).size().unstack(fill_value=0)

# Reorder the days of the week to maintain proper order
rides_by_day_month_combined_top_stations =␣
  ↪rides_by_day_month_combined_top_stations.reindex(
    ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday',␣
 ↪'Sunday'], level='day_of_week'
)

# Display the combined results for each month for the top stations
print("\nRides by Day of the Week for Each Month in 2024 (Top Stations):")
print(rides_by_day_month_combined_top_stations)
```

```
Rides by Day of the Week for Each Month in 2024 (Top Stations):
member_casual                         casual  member
Month day_of_week rideable_type
7     Monday      classic_bike      1522    8165
                  electric_bike     2583   11032
      Tuesday     classic_bike      1608    9197
                  electric_bike     2768   11654
      Wednesday   classic_bike      1477    8970
                  electric_bike     2706   11504
      Thursday    classic_bike      1458    6718
                  electric_bike     2444    8551
      Friday      classic_bike      1306    5728
                  electric_bike     2528    7870
      Saturday    classic_bike      1980    5047
                  electric_bike     3287    6872
      Sunday      classic_bike      1833    4937
                  electric_bike     2803    6562
8     Monday      classic_bike      1098    6350
                  electric_bike     2223    8675
      Tuesday     classic_bike      1139    6544
                  electric_bike     2282    8945
      Wednesday   classic_bike      1214    6554
                  electric_bike     2137    9237
      Thursday    classic_bike      1371    7570
                  electric_bike     2774   11166
      Friday      classic_bike      1502    6984
                  electric_bike     3123   10408
      Saturday    classic_bike      4043    6760
                  electric_bike     4467    8834
      Sunday      classic_bike      1549    4546
                  electric_bike     2660    6358
9     Monday      classic_bike      1613    8783
                  electric_bike     3026   11772
      Tuesday     classic_bike      1251    7942
                  electric_bike     2468   10903
      Wednesday   classic_bike      1228    8196
                  electric_bike     2537   11392
      Thursday    classic_bike      1177    7781
```

```
                electric_bike    2500   10961
Friday          classic_bike     1427    7217
                electric_bike    3073   10689
Saturday        classic_bike     1647    5055
                electric_bike    3326    8024
Sunday          classic_bike     2021    6045
                electric_bike    3420    8518
```

10.7 Visualization of Ride Distribution by Day of the Week for Top 10 Stations Across Months (2024)

[48]:
```python
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Add a column for the day of the week from the 'started_at' column
filtered_top_stations['day_of_week'] = filtered_top_stations['started_at'].dt.
 ↪day_name()

# Group by day of the week and month, and count the rides for the top stations
rides_by_day_top_stations = filtered_top_stations.groupby(['day_of_week',␣
 ↪'Month']).size().reset_index(name='Counts')

# Reindex for the day of the week to maintain order
rides_by_day_top_stations['day_of_week'] = pd.
 ↪Categorical(rides_by_day_top_stations['day_of_week'],
                                                                    ␣
 ↪categories=['Monday', 'Tuesday', 'Wednesday',
                                                                                ␣
 ↪'Thursday', 'Friday', 'Saturday', 'Sunday'],
                                                                    ordered=True)

# Visualization: Plot rides by day of the week for each month for the top␣
 ↪stations
plt.figure(figsize=(14, 8))

# Loop through each month to plot
for month in rides_by_day_top_stations['Month'].unique():
    month_data = rides_by_day_top_stations[rides_by_day_top_stations['Month']␣
 ↪== month]
    sns.lineplot(data=month_data, x='day_of_week', y='Counts', label=f'Month␣
 ↪{month}', marker='o')

# Adding title and labels
plt.title('Number of Rides by Day of the Week for Each Month (Top 10 Stations)␣
 ↪in 2024', fontsize=16)
plt.xlabel('Day of the Week', fontsize=14)
plt.ylabel('Number of Rides', fontsize=14)

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

# Show the legend
plt.legend(title='Month')

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

# Print the summary of rides by day for the top stations
print("\nRides by Day of the Week and Month for Top Stations:")
print(rides_by_day_top_stations)
```



```
Rides by Day of the Week and Month for Top Stations:
   day_of_week  Month  Counts
0       Friday      7   17432
1       Friday      8   22017
2       Friday      9   22406
3       Monday      7   23302
4       Monday      8   18346
5       Monday      9   25194
6     Saturday      7   17186
7     Saturday      8   24104
```

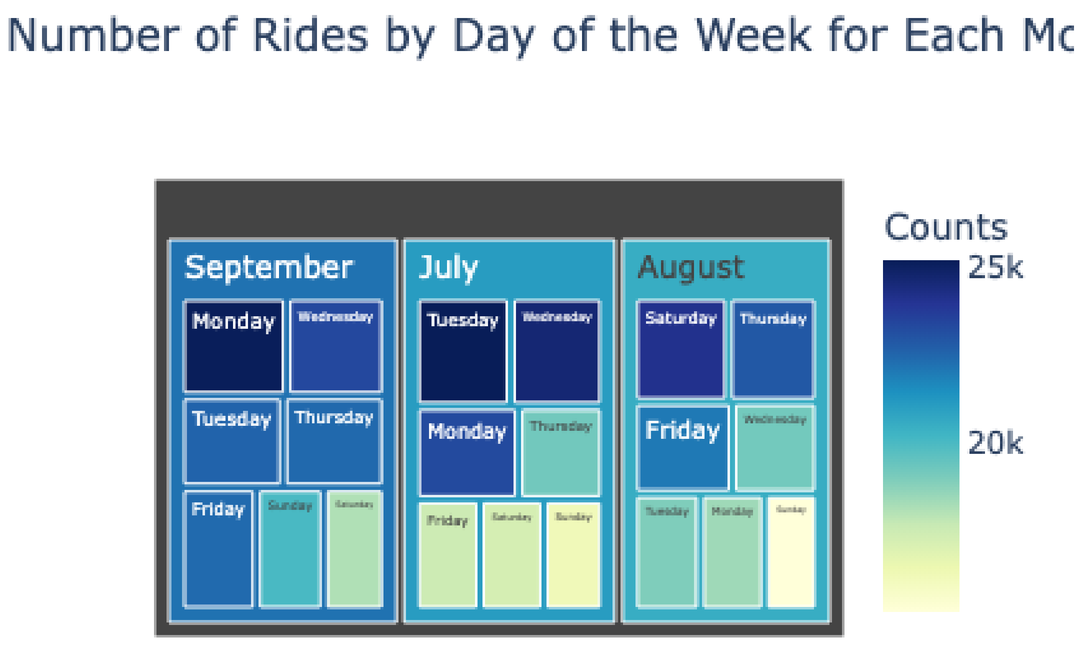
```
8      Saturday      9   18052
9        Sunday      7   16135
10       Sunday      8   15113
11       Sunday      9   20004
12     Thursday      7   19171
13     Thursday      8   22881
14     Thursday      9   22419
15      Tuesday      7   25227
16      Tuesday      8   18910
17      Tuesday      9   22564
18    Wednesday      7   24657
19    Wednesday      8   19142
20    Wednesday      9   23353
```

-A line plot displays the number of rides for each day of the week, grouped by month (July, August, and September).
-Each line represents one month, with markers indicating the number of rides for each day of the week.
-The output also includes the summary data, showing the exact counts of rides for each combination of day of the week and month.

This code generates a treemap visualization of the number of rides by day of the week for each month, focusing on the top 10 stations in 2024.

[51]:
```
import pandas as pd
import plotly.express as px

# Add a column for the day of the week from the 'started_at' column
filtered_top_stations['day_of_week'] = filtered_top_stations['started_at'].dt.
 ↪day_name()

# Group by day of the week and month, and count the rides for the top stations
rides_by_day_top_stations = filtered_top_stations.groupby(['day_of_week',␣
 ↪'Month'], observed=False).size().reset_index(name='Counts')

# Reindex for the day of the week to maintain order
rides_by_day_top_stations['day_of_week'] = pd.
 ↪Categorical(rides_by_day_top_stations['day_of_week'],
                                                                          ␣
 ↪categories=['Monday', 'Tuesday', 'Wednesday',
                                                                                         ␣
 ↪'Thursday', 'Friday', 'Saturday', 'Sunday'],
                                                                           ordered=True)

# Replace numeric month values with month names
month_name_map = {7: 'July', 8: 'August', 9: 'September'}
rides_by_day_top_stations['Month'] = rides_by_day_top_stations['Month'].
 ↪map(month_name_map)
```

```
# Ensure the 'Month' column is treated as a categorical variable with a␣
 ↪specific order
rides_by_day_top_stations['Month'] = pd.
 ↪Categorical(rides_by_day_top_stations['Month'], categories=['July',␣
 ↪'August', 'September'], ordered=True)

# Create the treemap using plotly.express
fig = px.treemap(rides_by_day_top_stations,
                 path=['Month', 'day_of_week'],  # Define the hierarchical␣
 ↪structure
                 values='Counts',  # Size of the rectangles corresponds to the␣
 ↪counts
                 color='Counts',  # Color by the number of rides
                 color_continuous_scale='YlGnBu',  # Choose a color scale
                 title="Number of Rides by Day of the Week for Each Month (Top␣
 ↪10 Stations) in 2024")


# Show the treemap
fig.show()
```



10.8 Analyzing Frequency of Trips by Time of Day

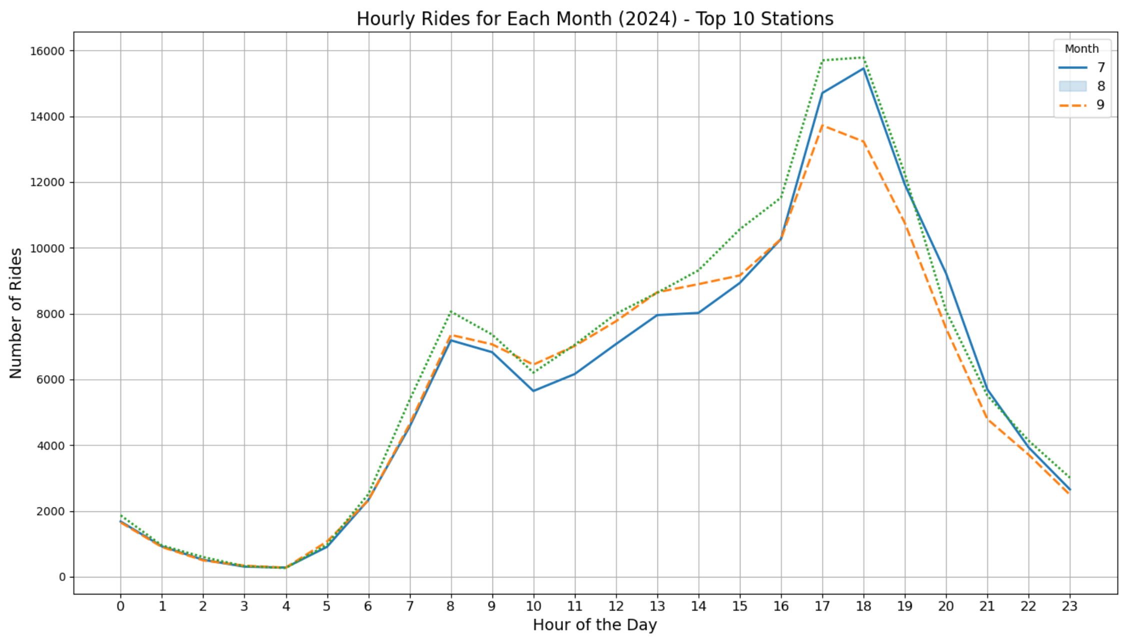
```
[53]: # Extract hour from 'started_at' column
filtered_top_stations['hour'] = filtered_top_stations['started_at'].dt.hour

# Group by hour and month to calculate ride counts for the top stations
rides_by_hour_month_top_stations = filtered_top_stations.groupby(['Month',␣
 ↪'hour']).size().reset_index(name='Counts')

# Pivot the table so that each month is a separate column for better comparison
rides_by_hour_month_pivot_top_stations = rides_by_hour_month_top_stations.
 ↪pivot(index='hour', columns='Month', values='Counts').fillna(0)

# Display the combined table for top stations
print("\nCombined Rides by Hour and Month (Top 10 Stations):")
print(rides_by_hour_month_pivot_top_stations)
```

```
Combined Rides by Hour and Month (Top 10 Stations):
Month      7      8      9
hour
0       1677   1650   1865
1        921    901    949
2        513    495    596
3        304    334    318
4        277    274    263
5        907   1069    973
6       2329   2318   2507
7       4557   4632   5370
8       7186   7351   8059
9       6825   7064   7361
10      5645   6447   6204
11      6160   7014   7049
12      7071   7765   7992
13      7953   8647   8632
14      8019   8894   9315
15      8929   9157  10561
16     10268  10266  11524
17     14705  13720  15698
18     15452  13229  15787
19     11922  10751  12246
20      9216   7546   8076
21      5692   4790   5505
22      3929   3708   4134
```

```
23        2653    2491    3008
```

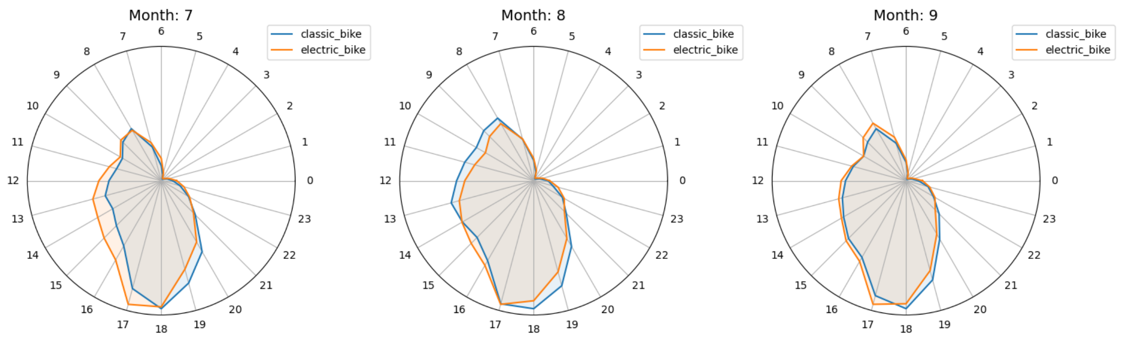
This code generates a clear and informative line plot showing hourly ride trends for each month, helping to visualize patterns in bike usage across different hours of the day for the top 10 stations in 2024.

[55]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Set a color palette for better distinction between months
palette = sns.color_palette("tab10", len(rides_by_hour_month_pivot_top_stations.
  ↪columns))

# Plot the hourly rides for each month (filtered for top 10 stations)
plt.figure(figsize=(14, 8))
sns.lineplot(data=rides_by_hour_month_pivot_top_stations, linewidth=2,␣
  ↪palette=palette)

# Adding title and labels
plt.title("Hourly Rides for Each Month (2024) - Top 10 Stations", fontsize=16)
plt.xlabel("Hour of the Day", fontsize=14)
plt.ylabel("Number of Rides", fontsize=14)

# Customize the x-axis with hours of the day
plt.xticks(range(0, 24), fontsize=12)

# Show the legend for months with a larger font
plt.legend(title='Month', labels=rides_by_hour_month_pivot_top_stations.
  ↪columns, loc='upper right', fontsize=12)

# Add gridlines for better readability
plt.grid(True)

# Display the plot with tight layout
plt.tight_layout()
plt.show()
```



[56]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Prepare data (aggregate by hour, rideable_type, and month)
rides_avg_monthly = filtered_top_stations.groupby(['Month', 'hour',␣
↪'rideable_type']).size().reset_index(name='Number of Rides')
rides_avg_monthly = rides_avg_monthly.pivot_table(index='hour',␣
↪columns=['Month', 'rideable_type'], values='Number of Rides', fill_value=0)

# Normalize data for radar chart
rides_avg_monthly_norm = rides_avg_monthly.div(rides_avg_monthly.max(axis=0),␣
↪axis=1)

# Radar chart setup
categories = rides_avg_monthly_norm.index.tolist()  # Hours of the day
num_vars = len(categories)
angles = np.linspace(0, 2 * np.pi, num_vars, endpoint=False).tolist()  # Angles␣
↪for radar chart
angles += angles[:1]  # Close the radar chart

# Create subplots for each month
months = rides_avg_monthly.columns.get_level_values('Month').unique()
num_months = len(months)
```

```python
fig, axes = plt.subplots(1, num_months, figsize=(5 * num_months, 6),␣
 ↪subplot_kw=dict(polar=True))

# If only one month, axes won't be iterable
if num_months == 1:
    axes = [axes]

# Plot radar charts for each month
for ax, month in zip(axes, months):
    month_data = rides_avg_monthly_norm.xs(month, level='Month', axis=1)
    rideable_types = month_data.columns.tolist()

    for rideable_type in rideable_types:
        values = month_data[rideable_type].tolist()
        values += values[:1]  # Close the line
        ax.plot(angles, values, label=rideable_type)
        ax.fill(angles, values, alpha=0.1)

    # Add labels and title
    ax.set_yticks([])
    ax.set_xticks(angles[:-1])
    ax.set_xticklabels(categories)
    ax.set_title(f"Month: {month}", size=14)
    ax.legend(loc='upper right', bbox_to_anchor=(1.3, 1.1))

# Adjust layout
plt.tight_layout()
plt.show()
```



11 Citibike Data and Weather Analysis for September 2024

In this analysis, we examined the Citibike ride data and weather data for September 2024 to identify patterns in bike usage and how weather conditions might affect ridership. The following steps were taken:

1. **Loading and Cleaning Data**
   - Loaded multiple CSV files containing bike trip data (`202409-citibike-tripdata_1.csv` to `202409-citibike-tripdata_5.csv`) and weather data for New York City in September 2024.
   - Cleaned the bike data by converting `started_at` and `ended_at` columns to `datetime` objects.
   - Dropped rows with missing critical columns (`start_station_name`, `end_station_name`).
   - Removed duplicates based on the `ride_id` and filtered out trips with negative durations.
2. **Feature Engineering**
   - Calculated the trip duration in seconds and filtered out trips with negative durations.
   - Extracted the hour of the day from the `started_at` column for bike data and from the `datetime` column for weather data.
   - Added a `week` column to both datasets to group data by the week of the year.
   - Created a `week_of_month` column to indicate the week number within the month.
   - Extracted the date from the datetime columns for both datasets.
3. **Data Merging**
   - Merged the bike trip data with daily weather averages (temperature, humidity, wind speed) on matching dates.
   - Aggregated the weather data to compute daily averages of temperature, humidity, and wind speed.
4. **Station Analysis**
   - Analyzed the most popular Citibike stations by counting rides for each `start_station_name`.
   - Identified the top 10 stations based on ride counts.
5. **Aggregation and Pivoting**
   - Aggregated the merged data by day and weekday, calculating the total number of rides and the average temperature per day.
   - Pivoted the aggregated data into calendar-like structures, one for ride counts and one for average temperatures, to allow for easy visualization.
6. **Heatmap Visualizations**
   - **Ride Count Calendar Heatmap**: A heatmap visualizing the total number of rides for each day of the month and weekday.
   - **Temperature Calendar Heatmap**: A heatmap showing the average temperature for each day of the month and weekday.

These heatmaps provide clear insights into bike usage patterns and how temperature influences ridership throughout September 2024.

[58]:
```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
```

```python
# Load the September 2024 bike data files
bike_data_sep_1 = pd.read_csv('202409-citibike-tripdata_1.csv',␣
 ↪low_memory=False)
bike_data_sep_2 = pd.read_csv('202409-citibike-tripdata_2.csv',␣
 ↪low_memory=False)
bike_data_sep_3 = pd.read_csv('202409-citibike-tripdata_3.csv',␣
 ↪low_memory=False)
bike_data_sep_4 = pd.read_csv('202409-citibike-tripdata_4.csv',␣
 ↪low_memory=False)
bike_data_sep_5 = pd.read_csv('202409-citibike-tripdata_5.csv',␣
 ↪low_memory=False)

# Concatenate all the datasets together to create one large dataset
bike_data_sep = pd.concat([bike_data_sep_1, bike_data_sep_2, bike_data_sep_3,␣
 ↪bike_data_sep_4, bike_data_sep_5], ignore_index=True)

# Load the September 2024 weather data
weather_data_sep = pd.read_csv('new york city 2024-09-01 to 2024-09-30 (2).
 ↪csv', low_memory=False)

# Convert the 'started_at' and 'ended_at' columns to datetime objects in the␣
 ↪bike data
bike_data_sep['started_at'] = pd.to_datetime(bike_data_sep['started_at'])
bike_data_sep['ended_at'] = pd.to_datetime(bike_data_sep['ended_at'])

# Drop rows with missing critical columns
bike_data_sep.dropna(subset=['start_station_name', 'end_station_name'],␣
 ↪inplace=True)

# Clean column names
bike_data_sep.columns = bike_data_sep.columns.str.strip()

# Convert coordinates to numeric format
bike_data_sep[['start_lat', 'start_lng', 'end_lat', 'end_lng']] =␣
 ↪bike_data_sep[['start_lat', 'start_lng', 'end_lat', 'end_lng']].apply(pd.
 ↪to_numeric, errors='coerce')

# Remove duplicate rows based on 'ride_id'
bike_data_sep.drop_duplicates(subset='ride_id', inplace=True)

# Calculate the trip duration in seconds
bike_data_sep['duration'] = (bike_data_sep['ended_at'] -␣
 ↪bike_data_sep['started_at']).dt.total_seconds()

# Filter out trips with negative duration
bike_data_sep = bike_data_sep[bike_data_sep['duration'] >= 0]
```

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

# Convert the 'datetime' column in the weather data to datetime type
weather_data_sep['datetime'] = pd.to_datetime(weather_data_sep['datetime'])

# Extract the hour from the 'datetime' column
weather_data_sep['hour'] = weather_data_sep['datetime'].dt.hour

# Drop unnecessary columns from the weather data
columns_to_delete = ['preciptype', 'snow', 'sealevelpressure',␣
 ↪'solarradiation', 'solarenergy', 'severerisk']
weather_data_sep.drop(columns=columns_to_delete, inplace=True)

# Add a 'week' column to both datasets
bike_data_sep['week'] = bike_data_sep['started_at'].dt.isocalendar().week
weather_data_sep['week'] = weather_data_sep['datetime'].dt.isocalendar().week

# Merge the data based on 'week' and 'hour'
merged_data_sep = pd.merge(bike_data_sep, weather_data_sep, on=['hour',␣
 ↪'week'], how='left')

# Add 'week_of_month' column to both datasets
def get_week_of_month(date):
    return (date.day - 1) // 7 + 1

bike_data_sep['week_of_month'] = bike_data_sep['started_at'].
 ↪apply(get_week_of_month)
weather_data_sep['week_of_month'] = weather_data_sep['datetime'].
 ↪apply(get_week_of_month)

# Extract the date from the datetime columns
bike_data_sep['date'] = bike_data_sep['started_at'].dt.date
weather_data_sep['date'] = weather_data_sep['datetime'].dt.date

# Aggregate the weather data to get daily averages
daily_weather_avg = weather_data_sep.groupby('date').agg(
    avg_temp=('temp', 'mean'),
    avg_humidity=('humidity', 'mean'),
    avg_wind_speed=('windspeed', 'mean')
).reset_index()

# Merge the bike data with daily weather averages
merged_data_sep = pd.merge(bike_data_sep, daily_weather_avg, on='date',␣
 ↪how='left')
```

```python
# Group by the start station name and count the number of rides
station_counts = merged_data_sep['start_station_name'].value_counts().
 ↪reset_index()
station_counts.columns = ['start_station_name', 'ride_count']

# Sort in descending order and select the top 10 stations
top_10_stations = station_counts.head(10)

# Filter the merged data to include only the top 10 stations
top_10_station_names = top_10_stations['start_station_name'].tolist()
top_10_station_data = merged_data_sep[merged_data_sep['start_station_name'].
 ↪isin(top_10_station_names)]

# Extract day and weekday
merged_data_sep['day'] = merged_data_sep['started_at'].dt.day
merged_data_sep['weekday'] = merged_data_sep['started_at'].dt.weekday

# Aggregate data: Total rides and average temperature per day
daily_data = merged_data_sep.groupby(['day', 'weekday']).agg(
    ride_count=('ride_id', 'count'),
    avg_temp=('avg_temp', 'mean')
).reset_index()

# Pivot data for calendar view
ride_count_calendar = daily_data.pivot(index='weekday', columns='day',␣
 ↪values='ride_count')
temperature_calendar = daily_data.pivot(index='weekday', columns='day',␣
 ↪values='avg_temp')

# Weekday labels
weekday_labels = ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun']

# Plot Ride Count Calendar Heatmap
plt.figure(figsize=(15, 8))
sns.heatmap(
    ride_count_calendar,
    annot=True, fmt='.0f', cmap='Blues', linewidths=0.5,
    cbar_kws={'label': 'Number of Rides'}
)
plt.title('Ride Count Calendar Heatmap (September 2024)')
plt.xlabel('Day of Month')
plt.ylabel('Weekday')
plt.yticks(ticks=np.arange(7) + 0.5, labels=weekday_labels, rotation=0)
plt.show()

# Plot Temperature Calendar Heatmap
plt.figure(figsize=(15, 8))
```

```
sns.heatmap(
    temperature_calendar,
    annot=True, fmt='.1f', cmap='coolwarm', linewidths=0.5,
    cbar_kws={'label': 'Average Temperature (°F)'}
)
plt.title('Temperature Calendar Heatmap (September 2024)')
plt.xlabel('Day of Month')
plt.ylabel('Weekday')
plt.yticks(ticks=np.arange(7) + 0.5, labels=weekday_labels, rotation=0)
plt.show()
```

```
---------------------------------------------------------------------------
MemoryError                               Traceback (most recent call last)
Cell In[58], line 59
     56 weather_data_sep['week'] = weather_data_sep['datetime'].dt.isocalendar().
  ↪week
     58 # Merge the data based on 'week' and 'hour'
---> 59 merged_data_sep = pd.merge(bike_data_sep, weather_data_sep, on=['hour',␣
  ↪'week'], how='left')
     61 # Add 'week_of_month' column to both datasets
     62 def get_week_of_month(date):

File ~\anaconda3\Lib\site-packages\pandas\core\reshape\merge.py:184, in␣
  ↪merge(left, right, how, on, left_on, right_on, left_index, right_index, sort, ␣
  ↪suffixes, copy, indicator, validate)
    169 else:
    170     op = _MergeOperation(
    171         left_df,
    172         right_df,
   (…)
    182         validate=validate,
    183     )
--> 184     return op.get_result(copy=copy)

File ~\anaconda3\Lib\site-packages\pandas\core\reshape\merge.py:888, in␣
  ↪_MergeOperation.get_result(self, copy)
    884     self.left, self.right = self._indicator_pre_merge(self.left, self.
  ↪right)
    886 join_index, left_indexer, right_indexer = self._get_join_info()
--> 888 result = self._reindex_and_concat(
    889     join_index, left_indexer, right_indexer, copy=copy
    890 )
    891 result = result.__finalize__(self, method=self._merge_type)
    893 if self.indicator:

File ~\anaconda3\Lib\site-packages\pandas\core\reshape\merge.py:879, in␣
  ↪_MergeOperation._reindex_and_concat(self, join_index, left_indexer,␣
  ↪right_indexer, copy)
```

```
    877 left.columns = llabels
    878 right.columns = rlabels
--> 879 result = concat([left, right], axis=1, copy=copy)
    880 return result

File ~\anaconda3\Lib\site-packages\pandas\core\reshape\concat.py:395, in␣
 ↪concat(objs, axis, join, ignore_index, keys, levels, names, verify_integrity,␣
 ↪sort, copy)
    380     copy = False
    382 op = _Concatenator(
    383     objs,
    384     axis=axis,
   (…)
    392     sort=sort,
    393 )
--> 395 return op.get_result()

File ~\anaconda3\Lib\site-packages\pandas\core\reshape\concat.py:684, in␣
 ↪_Concatenator.get_result(self)
    680             indexers[ax] = obj_labels.get_indexer(new_labels)
    682     mgrs_indexers.append((obj._mgr, indexers))
--> 684 new_data = concatenate_managers(
    685     mgrs_indexers, self.new_axes, concat_axis=self.bm_axis, copy=self.
 ↪copy
    686 )
    687 if not self.copy and not using_copy_on_write():
    688     new_data._consolidate_inplace()

File ~\anaconda3\Lib\site-packages\pandas\core\internals\concat.py:131, in␣
 ↪concatenate_managers(mgrs_indexers, axes, concat_axis, copy)
    124 # Assertions disabled for performance
    125 # for tup in mgrs_indexers:
    126 #    # caller is responsible for ensuring this
    127 #    indexers = tup[1]
    128 #    assert concat_axis not in indexers
    130 if concat_axis == 0:
--> 131     mgrs = _maybe_reindex_columns_na_proxy(axes, mgrs_indexers,␣
 ↪needs_copy)
    132     return mgrs[0].concat_horizontal(mgrs, axes)
    134 if len(mgrs_indexers) > 0 and mgrs_indexers[0][0].nblocks > 0:

File ~\anaconda3\Lib\site-packages\pandas\core\internals\concat.py:230, in␣
 ↪_maybe_reindex_columns_na_proxy(axes, mgrs_indexers, needs_copy)
    220         mgr = mgr.reindex_indexer(
    221             axes[i],
    222             indexers[i],
   (…)
    227             use_na_proxy=True,  # only relevant for i==0
    228         )
    229     if needs_copy and not indexers:
--> 230         mgr = mgr.copy()
    232     new_mgrs.append(mgr)
    233 return new_mgrs

File ~\anaconda3\Lib\site-packages\pandas\core\internals\managers.py:593, in␣
 ↪BaseBlockManager.copy(self, deep)
    590     else:
    591         new_axes = list(self.axes)
--> 593 res = self.apply("copy", deep=deep)
    594 res.axes = new_axes
    596 if self.ndim > 1:
    597     # Avoid needing to re-compute these

File ~\anaconda3\Lib\site-packages\pandas\core\internals\managers.py:363, in␣
 ↪BaseBlockManager.apply(self, f, align_keys, **kwargs)
    361         applied = b.apply(f, **kwargs)
    362     else:
--> 363         applied = getattr(b, f)(**kwargs)
    364     result_blocks = extend_blocks(applied, result_blocks)
    366 out = type(self).from_blocks(result_blocks, self.axes)

File ~\anaconda3\Lib\site-packages\pandas\core\internals\blocks.py:796, in Bloc
 ↪copy(self, deep)
    794 refs: BlockValuesRefs | None
    795 if deep:
--> 796     values = values.copy()
    797     refs = None
    798 else:

MemoryError: Unable to allocate 2.22 GiB for an array with shape (9, 33097503)␣
 ↪and data type float64
```

11.0.1 Ride Count Calendar Heatmap

This heatmap shows the number of rides for each day and weekday in September 2024.

11.0.2 Temperature Calendar Heatmap

This heatmap displays the average temperature for each day and weekday in September 2024.

11.1 Predicting Duration with Linear Regression

This script uses linear regression to predict our target variable `duration`. We based this prediciton on the features: 'avg_temp','weekday','hour','start_lat','avg_wind_speed','end_lat','end_lng' .

1. A correlation matrix is computed to identify numerical columns most correlated with

`duration` for feature selection.

2. Missing values are removed from the selected columns, and the data is split into training and testing sets.
3. The Linear Regression model is trained, and predictions are made on the test set.
4. Model performance is evaluated using metrics like Mean Absolute Error (MAE), Mean Squared Error (MSE), and R², showing the model's accuracy and predictive power.

[61]:
```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

#first we are starting with creating a correlation matrix based on our target␣
 ↪variable duration.
# check for numerical columns in the dataset to set up a correlation
numerical_columns = merged_data_sep.select_dtypes(include=['float64', 'int64',␣
 ↪'int32', 'UInt32']).columns

# calculate the correlation matrix for the numerical columns
correlation_matrix = merged_data_sep[numerical_columns].corr()

# foccused it based on the duration column
duration_correlation = correlation_matrix['duration'].
 ↪sort_values(ascending=False)

# look at the correlations with 'duration'
print(duration_correlation)


# select features and target variable
#features are based on the correlation we saw earlier.
features =␣
 ↪['avg_temp','weekday','hour','start_lat','avg_wind_speed','end_lat','end_lng']
target = 'duration'

#drop rows with missing values in selected columns as some of these columns␣
 ↪aren't cleaned before
merged_data_sep = merged_data_sep.dropna(subset=features + [target])

X = merged_data_sep[features]
y = merged_data_sep[target]

# split the data in a test and trainnig set
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
  ↪random_state=42)

# train the linear model regression
model = LinearRegression()
model.fit(X_train, y_train)

# make the predictions
y_pred = model.predict(X_test)

# look at the scores of the model
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# print results
print(f"MAE = {mae}")
print(f"MSE= {mse}")
print(f"R² = {r2}")
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[61], line 8
      4 from sklearn.metrics import mean_absolute_error, mean_squared_error,␣
  ↪r2_score
      6 #first we are starting with creating a correlation matrix based on our␣
  ↪target variable duration.
      7 # check for numerical columns in the dataset to set up a correlation
----> 8 numerical_columns = merged_data_sep.select_dtypes(include=['float64',␣
  ↪'int64', 'int32', 'UInt32']).columns
     10 # calculate the correlation matrix for the numerical columns
     11 correlation_matrix = merged_data_sep[numerical_columns].corr()

NameError: name 'merged_data_sep' is not defined
```

```
[ ]:
```