Final_Report

May 10, 2024

0.1 Title & Author:-

Saikumarreddy Pochireddygari : spochire@syr.edu

Vaishnavi Meka : vameka@syr.edu

0.2 Motivation & Background:-

Divvy is a Bike Sharing Company operating in Chicago. They have two Segment of customers premium and non premium members(Casual Riders).

Through this Bike Sharing program by Divvy, People can rent out bikes and roam freely in Chicago.

Everything seemed good, But Divvy's current marketing approach isn't effectively converting casual riders into annual members, hindering long-term financial stability.

So the problem statement was to optimize DIVVY's revenue model and drive long-term sustainability by maximizing annual memberships for the year 2024.

If we can find some sort of difference between two segment of users we can devise effective targeting marketing strategies to one segment converting them to paid premium segment which would increase the revenue.

0.3 Analysis Strategy:-

My approach is to analyze the patterns of "Member" and "Casual" riders for the year 2022 & check if similar trends occur in 2023, then trends may repeat again in 2024, by doing this kind of approach we can target specific casual riders and offer them targeted incentives through membership

0.4 Data Sources:-

We collected the data from the below two sources. 1. Data: https://divvybikes.com/system-data 2. Weather Data: https://www.visualcrossing.com/weather/weather-data-services

0.5 Data Characteristics:-

Data consist of 32 features and 5 million records, Temperature consist of 720 records for two years and three features

0.6 Data Cleaning and Transformations:-

[3]:
```
trip_data = pd.read_csv('divvy_trip_data_2022_2023.csv')
weather_2022 = pd.read_csv('chicago 2022-01-01 to 2022-12-31.csv')
weather_2023 = pd.read_csv('chicago 2023-01-01 to 2023-12-31.csv')

weather_22 = weather_2022.
  ↪drop(columns=['name','tempmax','tempmin','feelslikemax','feelslikemin','feelslike','dew','h
  ↪'windgust','precip','precipprob','precipcover','preciptype','snow','snowdepth','winddir','s
  ↪axis=1)
weather_23 = weather_2023.
  ↪drop(columns=['name','tempmax','tempmin','feelslikemax','feelslikemin','feelslike','dew','h
  ↪'windgust','precip','precipprob','precipcover','preciptype','snow','snowdepth','winddir','s
  ↪axis=1)

# #temp - F, windspeed- mph, visibility-miles

weather_data_22_23 = pd.concat([weather_22, weather_23], axis=0)
```

```
/var/folders/p1/lzpy06lx2hzfdmdx0x9lcvzc0000gn/T/ipykernel_59467/919988511.py:1:
DtypeWarning: Columns (9) have mixed types. Specify dtype option on import or
set low_memory=False.
  trip_data = pd.read_csv('divvy_trip_data_2022_2023.csv')
```

0.6.1 Data Cleaning

Approach 1: Data Standardization

[10]:
```
trip_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7182862 entries, 0 to 7182861
Data columns (total 13 columns):
 #   Column              Dtype
---  ------              -----
 0   ride_id             object
 1   rideable_type       object
 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           object
 10  end_lat             float64
 11  end_lng             float64
 12  member_casual       object
```

```
dtypes: float64(3), object(10)
memory usage: 712.4+ MB
```

[11]:
```
weather_data_22_23.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 730 entries, 0 to 364
Data columns (total 33 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   name              730 non-null    object
 1   datetime          730 non-null    object
 2   tempmax           730 non-null    float64
 3   tempmin           730 non-null    float64
 4   temp              730 non-null    float64
 5   feelslikemax      730 non-null    float64
 6   feelslikemin      730 non-null    float64
 7   feelslike         730 non-null    float64
 8   dew               730 non-null    float64
 9   humidity          730 non-null    float64
 10  precip            730 non-null    float64
 11  precipprob        730 non-null    int64
 12  precipcover       730 non-null    float64
 13  preciptype        304 non-null    object
 14  snow              730 non-null    float64
 15  snowdepth         730 non-null    float64
 16  windgust          729 non-null    float64
 17  windspeed         730 non-null    float64
 18  winddir           730 non-null    float64
 19  sealevelpressure  730 non-null    float64
 20  cloudcover        730 non-null    float64
 21  visibility        730 non-null    float64
 22  solarradiation    730 non-null    float64
 23  solarenergy       730 non-null    float64
 24  uvindex           730 non-null    int64
 25  severerisk        721 non-null    float64
 26  sunrise           730 non-null    object
 27  sunset            730 non-null    object
 28  moonphase         730 non-null    float64
 29  conditions        730 non-null    object
 30  description       730 non-null    object
 31  icon              730 non-null    object
 32  stations          730 non-null    object
dtypes: float64(22), int64(2), object(9)
memory usage: 193.9+ KB
```

Obs for weather:

1. Can observe that the date time is an object data type, so we can convert it back to date time format like yyyy-mm-dd
2. Name is also an object so we can conver it to str or category, since name is representing a city name, we can change it to category
3. Precip type is also an object but when we look at the dataset, we can notice it as string type, so we can change this to string
4. Sunrise, Sunset is also object but we should convert it to proper date time format
5. Similarly we should convet conditions to Categorical, descritpion to string, icon to categorical dtypes

Observation:-

1. Rideid is a object can probably convert it to str because we cannot convert every unique ride id to category
2. rideable type is a object but upon reviewing we can treat it as category
3. started at and ended it is a object but upon reviewing it is a time stamp so we can convert it to date time dtype
4. starting station name & ending is a object but we can treat station names a categories
5. Can observe that for the start station id and end station id we have mix of objects, integers and strings as dtypes so we can convert everything into one str dtype
6. member_casual is a object upon inspection its a category so we can convert it to categorical column

[12]:
```
## Making above changes

#1 Weather data

weather_data_22_23['datetime'] = pd.to_datetime(weather_data_22_23['datetime'],␣
↪format='%Y-%m-%d')

weather_data_22_23['name'] = weather_data_22_23['name'].astype('category')

weather_data_22_23['preciptype'] = weather_data_22_23['preciptype'].astype(str)

weather_data_22_23['sunrise'] = pd.to_datetime(weather_data_22_23['sunrise'])

weather_data_22_23['sunset'] = pd.to_datetime(weather_data_22_23['sunset'])

weather_data_22_23['conditions'] = weather_data_22_23['conditions'].
↪astype('category')

weather_data_22_23['icon'] = weather_data_22_23['icon'].astype('category')
```

```
weather_data_22_23['description'] = weather_data_22_23['description'].
 ↪astype(str)

weather_data_22_23.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 730 entries, 0 to 364
Data columns (total 33 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   name              730 non-null    category
 1   datetime          730 non-null    datetime64[ns]
 2   tempmax           730 non-null    float64
 3   tempmin           730 non-null    float64
 4   temp              730 non-null    float64
 5   feelslikemax      730 non-null    float64
 6   feelslikemin      730 non-null    float64
 7   feelslike         730 non-null    float64
 8   dew               730 non-null    float64
 9   humidity          730 non-null    float64
 10  precip            730 non-null    float64
 11  precipprob        730 non-null    int64
 12  precipcover       730 non-null    float64
 13  preciptype        730 non-null    object
 14  snow              730 non-null    float64
 15  snowdepth         730 non-null    float64
 16  windgust          729 non-null    float64
 17  windspeed         730 non-null    float64
 18  winddir           730 non-null    float64
 19  sealevelpressure  730 non-null    float64
 20  cloudcover        730 non-null    float64
 21  visibility        730 non-null    float64
 22  solarradiation    730 non-null    float64
 23  solarenergy       730 non-null    float64
 24  uvindex           730 non-null    int64
 25  severerisk        721 non-null    float64
 26  sunrise           730 non-null    datetime64[ns]
 27  sunset            730 non-null    datetime64[ns]
 28  moonphase         730 non-null    float64
 29  conditions        730 non-null    category
 30  description       730 non-null    object
 31  icon              730 non-null    category
 32  stations          730 non-null    object
dtypes: category(3), datetime64[ns](3), float64(22), int64(2), object(3)
memory usage: 179.6+ KB
```

[13]:
```
#2 Trip Data

trip_data['ride_id'] = trip_data['ride_id'].astype(str)
trip_data['rideable_type'] = trip_data['rideable_type'].astype('category')
trip_data['started_at'] = pd.to_datetime(trip_data['started_at'])
trip_data['ended_at'] = pd.to_datetime(trip_data['ended_at'])
trip_data['start_station_name'] = trip_data['start_station_name'].
 ↪astype('category')
trip_data['end_station_name'] = trip_data['end_station_name'].astype('category')


trip_data['start_station_id'] = trip_data['start_station_id'].astype(str)
trip_data['end_station_id'] = trip_data['end_station_id'].astype(str)

trip_data['member_casual'] = trip_data['member_casual'].astype('category')

trip_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7182862 entries, 0 to 7182861
Data columns (total 13 columns):
 #   Column              Dtype
---  ------              -----
 0   ride_id             object
 1   rideable_type       category
 2   started_at          datetime64[ns]
 3   ended_at            datetime64[ns]
 4   start_station_name  category
 5   start_station_id    object
 6   end_station_name    category
 7   end_station_id      object
 8   start_lat           float64
 9   start_lng           object
 10  end_lat             float64
 11  end_lng             float64
 12  member_casual       category
dtypes: category(4), datetime64[ns](2), float64(3), object(4)
memory usage: 534.5+ MB
```

[30]:
```

```

[ ]:
```

```

Approach 2. Dealing with Null Values

[14]:
```
## Since we are having big data we are removing the null values
```

```
weather_data_22_23.dropna(inplace=True)
trip_data.dropna(inplace=True)
```

[ ]:

Approach 3: Removing duplicates

[15]:
```
# Removing duplcates for weather data
weather_data_22_23.drop_duplicates(inplace=True)
```

[16]:
```
# Removing duplcates for ride data
trip_data.drop_duplicates(inplace=True)
```

[ ]:

[ ]:

0.6.2 Dividing data into 2022 and 2023 year as We can see the pattern for year 2022 and 2023 year and suggest our findings for 2024 FY year.¶

[134]:
```
rd_ = trip_data[~(trip_data.rideable_type == 'docked_bike')].copy()

ride_data_2022 = rd_[rd_['end_year'] == 2022]
ride_data_2023 = rd_[rd_['end_year'] == 2023]
```

[ ]:

[41]:
```
## Dividing Data in 2022 year with respect to member,casual

member_2022 = ride_data_2022[ride_data_2022.member_casual == 'member']
casual_2022 = ride_data_2022[ride_data_2022.member_casual == 'casual']
member_2023 = ride_data_2023[ride_data_2023.member_casual == 'member']
casual_2023 = ride_data_2023[ride_data_2023.member_casual == 'casual']
```

[ ]:

0.7 Summary of research questions and results:-

0.7.1 1. DA: How does trip duration looks like members and casuals in 2022 & 2023 year?

[54]:
```
member_2022['trip_duration'] = member_2022['trip_duration'] / 60
casual_2022['trip_duration'] = casual_2022['trip_duration'] / 60
member_2023['trip_duration'] = member_2023['trip_duration'] / 60
casual_2023['trip_duration'] = casual_2023['trip_duration'] / 60
```

[55]:
```
member_2022['trip_duration'].mean(),  casual_2022['trip_duration'].mean() ,␣
↪member_2023['trip_duration'].mean(), casual_2023['trip_duration'].mean()
```

[55]:
```
(11.677182126295166, 19.61467696546003, 12.130703080452207, 21.311094948626195)
```

Findings: Members maintained consistent average ride times across 2022 and 2023. Casual riders rode 78% and 73% longer than members in 2022 and 2023, respectively.

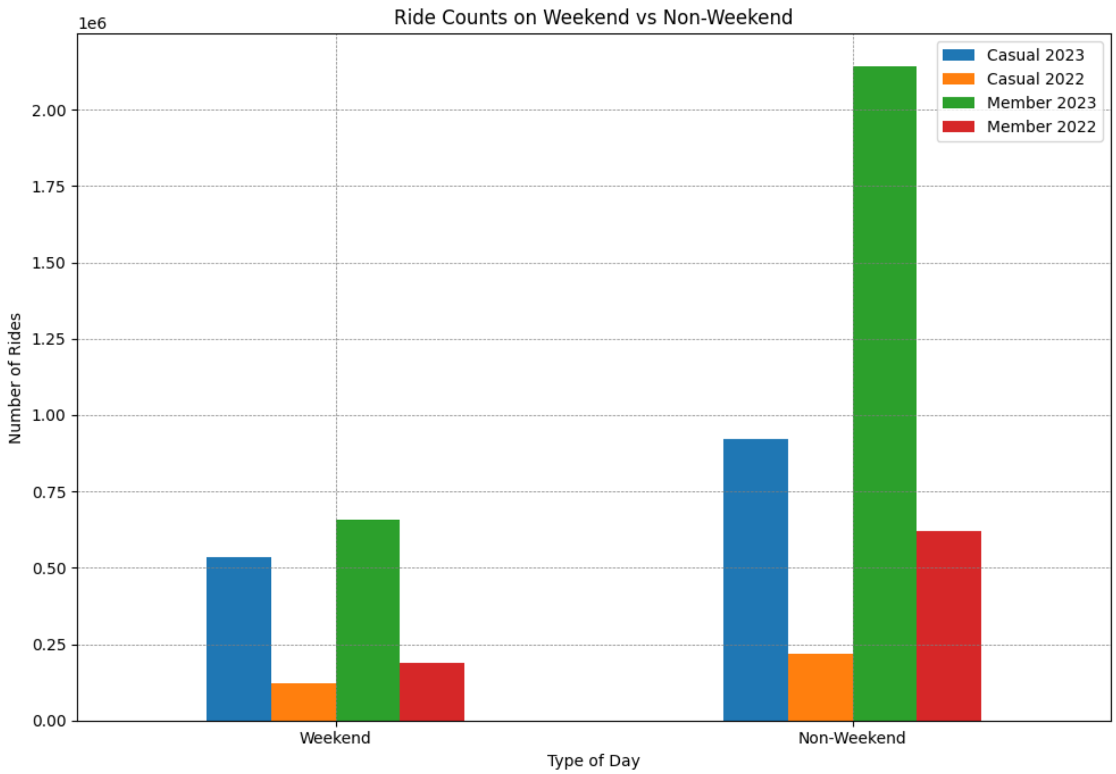
Inference: The consistent usage by members suggests routine trips, while the significantly longer ride times for casual riders indicate more sporadic or leisurely use.

[ ]:
```

```

0.7.2 2. DA How does weekend and non-weekend engagement looks like?

[124]:
```
def calculate_weekend_counts(df):
    weekend_rides = df[df['start_weekday'] >= 5].shape[0]
    non_weekend_rides = df[df['start_weekday'] < 5].shape[0]
    return weekend_rides, non_weekend_rides

# Calculate weekend and non-weekend counts for each DataFrame
casual_2023_weekend, casual_2023_non_weekend =␣
↪calculate_weekend_counts(casual_2023)
casual_2022_weekend, casual_2022_non_weekend =␣
↪calculate_weekend_counts(casual_2022)
member_2023_weekend, member_2023_non_weekend =␣
↪calculate_weekend_counts(member_2023)
member_2022_weekend, member_2022_non_weekend =␣
↪calculate_weekend_counts(member_2022)

# Data for plotting
data = {
    'Casual 2023': [casual_2023_weekend, casual_2023_non_weekend],
    'Casual 2022': [casual_2022_weekend, casual_2022_non_weekend],
    'Member 2023': [member_2023_weekend, member_2023_non_weekend],
    'Member 2022': [member_2022_weekend, member_2022_non_weekend]
}

# Convert data into a DataFrame for easier plotting
counts_df = pd.DataFrame(data, index=['Weekend', 'Non-Weekend'])

# Plot
counts_df.plot(kind='bar', figsize=(10, 7))

# Customization
plt.title('Ride Counts on Weekend vs Non-Weekend')
```

```
plt.ylabel('Number of Rides')
plt.xlabel('Type of Day')
plt.xticks(rotation=0)
plt.tight_layout()
plt.grid(True, which='both', linestyle='--', linewidth=0.5, color='grey')
plt.show()
```



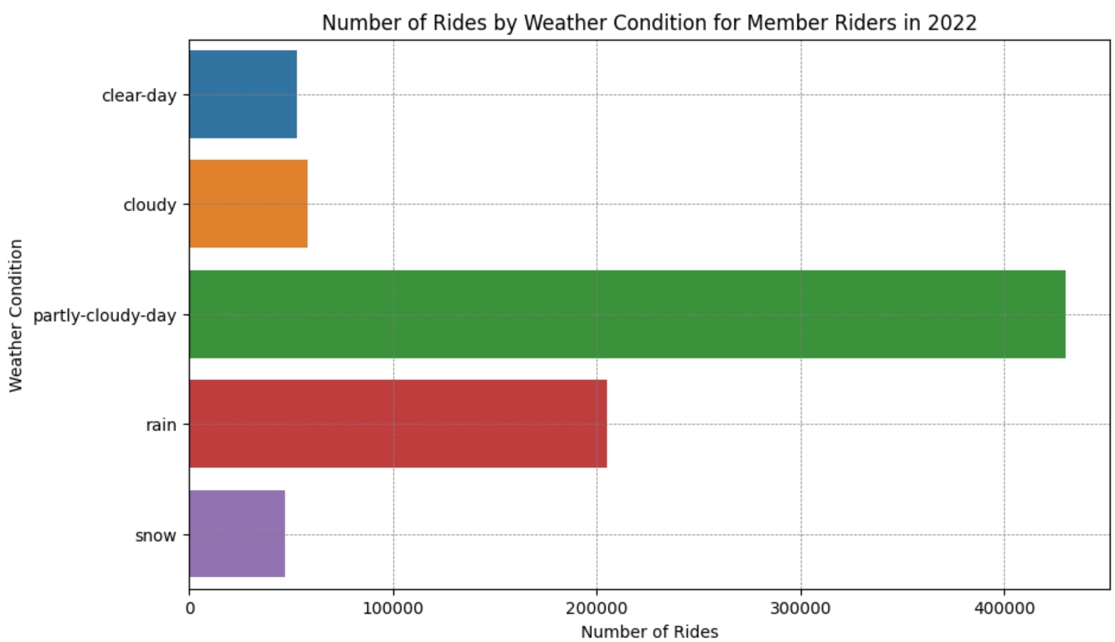
Findings: During weekends in both 2022 and 2023, ride engagement was equal between casuals and members. On non-weekend days, casual riders had 53% fewer rides compared to members in both years.

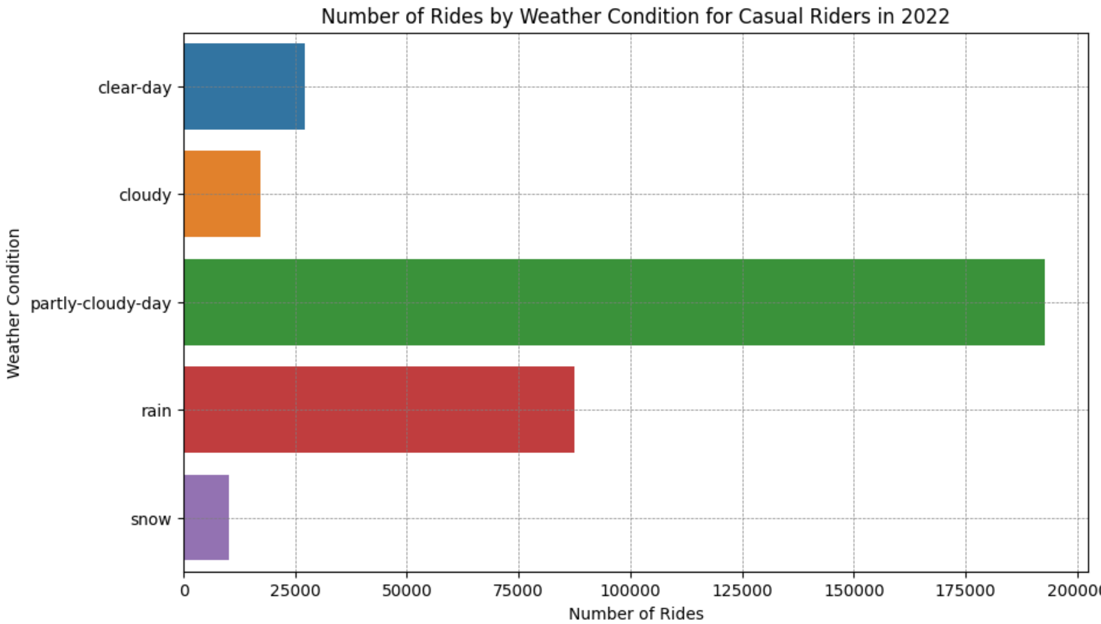
Inference: Equal weekend engagement suggests similar leisure activity patterns for both user types, whereas the significant drop in casual rider activity on non-weekends indicates that members likely use the service more for routine or commuter purposes.

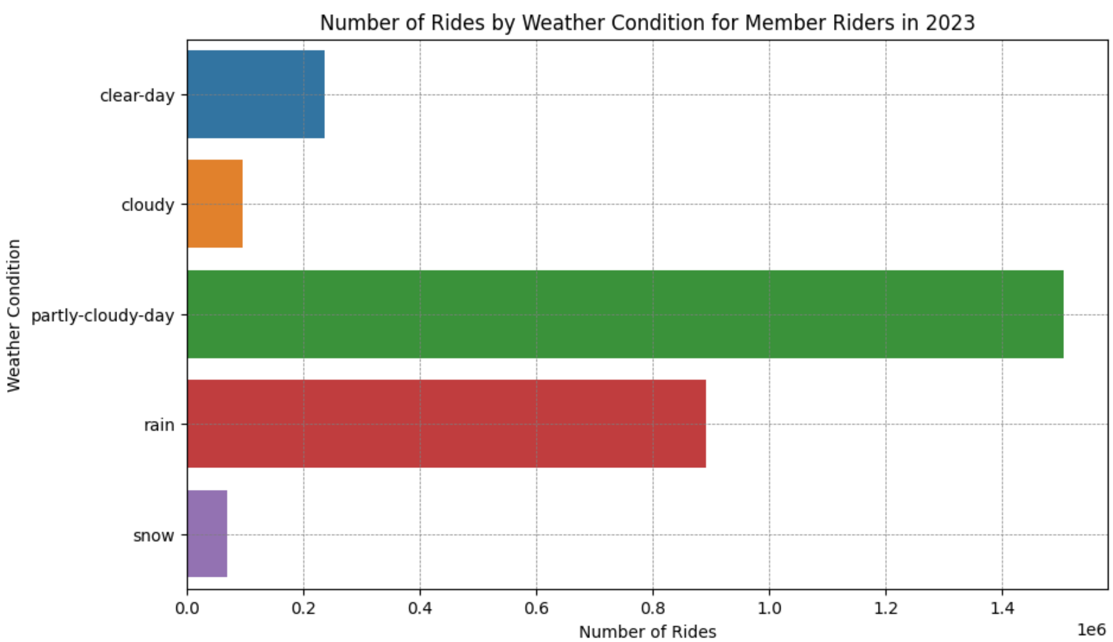
```
[ ]:
```

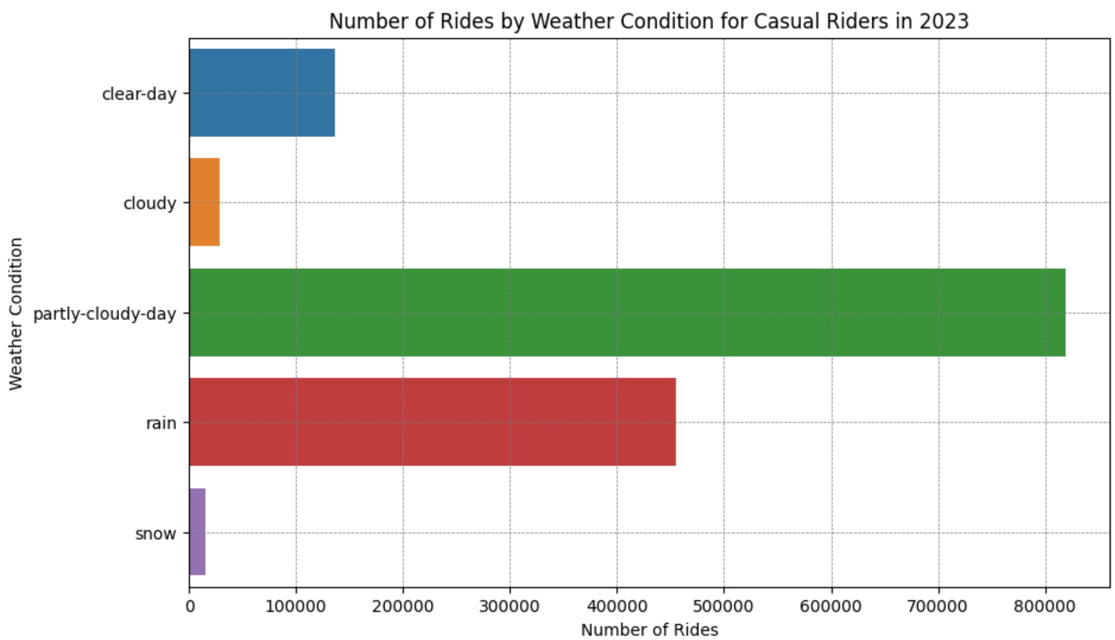
0.7.3 3. DA How does weather effect daily engagement of riders for 2022 and 2023 year?

[122]:
```
import matplotlib.pyplot as plt
import seaborn as sns

def weather_impact_analysis(df, user_type, year):
    weather_counts = df.groupby('icon')['ride_id'].count().
 ↪sort_values(ascending=False)
    plt.figure(figsize=(10, 6))
    sns.barplot(x=weather_counts.values, y=weather_counts.index)
    plt.title(f'Number of Rides by Weather Condition for {user_type} Riders in␣
 ↪{year}')
    plt.xlabel('Number of Rides')
    plt.ylabel('Weather Condition')
    plt.grid(True, which='both', linestyle='--', linewidth=0.5, color='grey')
    plt.show()

weather_impact_analysis(member_2022, 'Member', '2022')
weather_impact_analysis(casual_2022, 'Casual', '2022')
weather_impact_analysis(member_2023, 'Member', '2023')
weather_impact_analysis(casual_2023, 'Casual', '2023')
```




Number of Rides by Weather Condition for Casual Riders in 2022
Weather Condition
clear-day
cloudy
partly-cloudy-day
rain
snow
0
25000
50000
75000
100000
125000
150000
175000
200000
Number of Rides


Number of Rides by Weather Condition for Member Riders in 2023
Weather Condition
clear-day
cloudy
partly-cloudy-day
rain
snow
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
Number of Rides
1e6



Findings:

1. In 2022, on cloudy and snowy days, casual members took fewer rides compared to regular members.
2. The same trend was observed in 2023, with casual members engaging less on adverse weather days.

Inference: This consistent trend across both years indicates that regular members are less deterred by poor weather conditions, likely due to reliance on the service for essential commutes, whereas casual members' usage is more weather-sensitive, possibly due to discretionary or leisure-oriented travel.

[ ]:

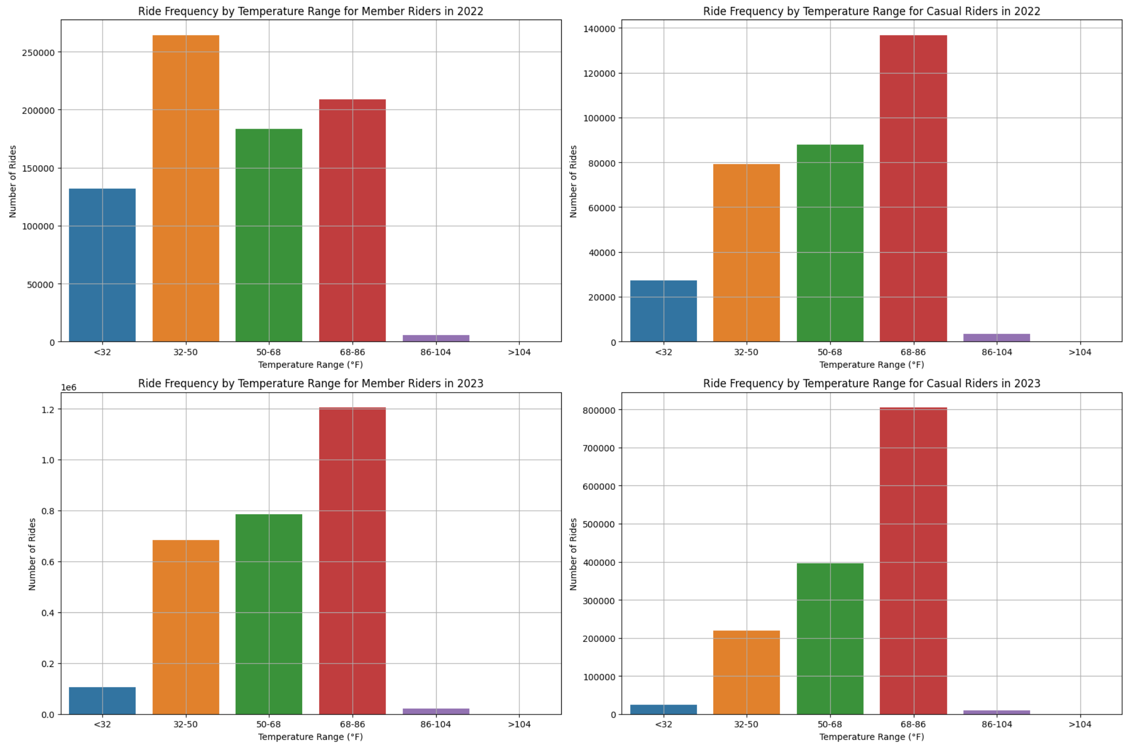
```

```

0.7.4 4. Analyzing theTemperature effect on members and casual riders for years 2022 and 2023

[131]:

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

def temperature_impact_analysis_combined(data_dict):
    # Create figure and axes for the subplots
    fig, axs = plt.subplots(2, 2, figsize=(18, 12)) # 2 rows, 2 columns
    axs = axs.flatten()  # Flatten the array to easily iterate over it
```

```
    # Loop through the dictionary of dataframes
    for idx, ((user_type, year), df) in enumerate(data_dict.items()):
        # Assuming 'temp' is in Fahrenheit, adjust the bin edges if 'temp' is␣
 ↪in another unit
        df['temp_range'] = pd.cut(df['temp'], bins=[-10, 32, 50, 68, 86, 104,␣
 ↪122],
                                   labels=['<32', '32-50', '50-68', '68-86',␣
 ↪'86-104', '>104'])
        temp_counts = df['temp_range'].value_counts().sort_index()

        # Plotting on the specified subplot axis
        sns.barplot(x=temp_counts.index, y=temp_counts.values, ax=axs[idx])
        axs[idx].set_title(f'Ride Frequency by Temperature Range for␣
 ↪{user_type} Riders in {year}')
        axs[idx].set_xlabel('Temperature Range (°F)')
        axs[idx].set_ylabel('Number of Rides')
        axs[idx].grid(True)  # Enabling grid

    plt.tight_layout()
    plt.show()

# Example usage with a dictionary of dataframes
data_dict = {
    ('Member', '2022'): member_2022,
    ('Casual', '2022'): casual_2022,
    ('Member', '2023'): member_2023,
    ('Casual', '2023'): casual_2023
}
temperature_impact_analysis_combined(data_dict)
```



Findings: 1. In both 2022 and 2023, when temperatures fell below 32°F, casual members took 300% fewer rides compared to members.

Inference:

This pattern suggests that regular members are likely committed to using the service regardless of colder conditions, possibly for essential commuting, while casual members are more sensitive to unfavorable temperatures, reducing their usage significantly in cold weather.

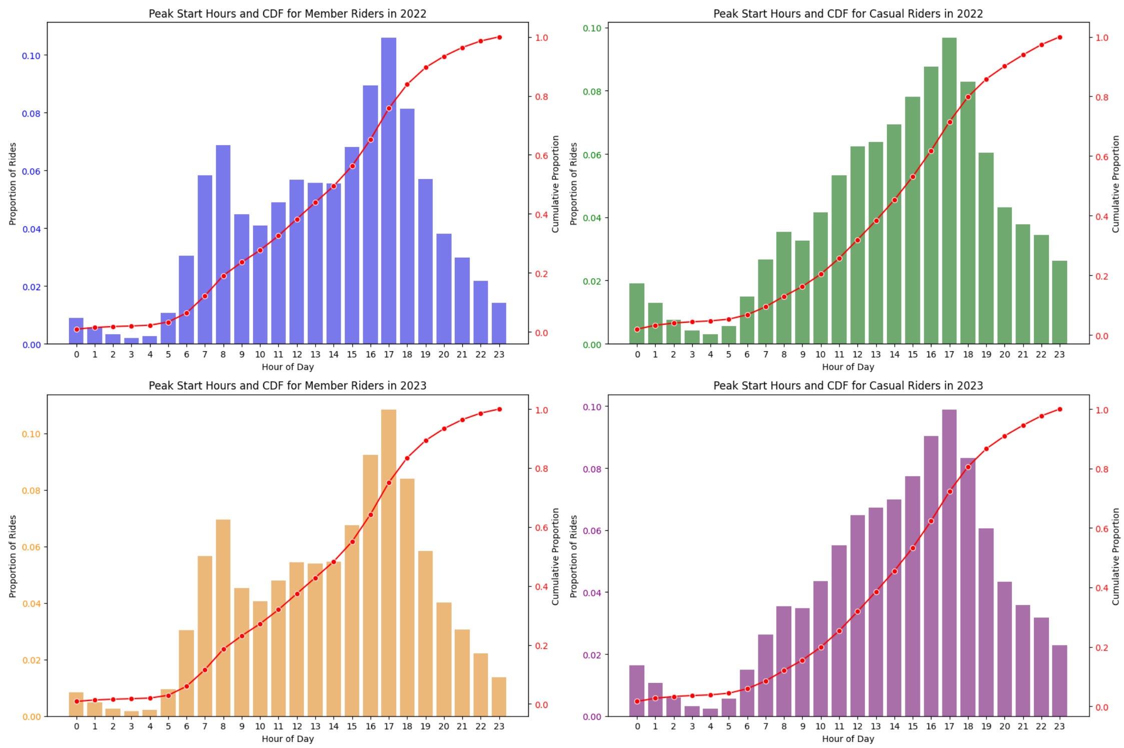
[ ]:

[ ]:

0.7.5 5: How is the Peak hours activity when we compared to member and casual riders for 2022 and 2023 year ? Starting stations

[132]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import warnings

# Ignore warnings
warnings.filterwarnings('ignore')
```

```python
def plot_peak_hours_combined(df_dict):
    fig, axes = plt.subplots(2, 2, figsize=(18, 12))  # 2 rows, 2 columns
    colors = ['blue', 'green', 'darkorange', 'purple']  # Colors for different plots
    ax_idx = 0  # Index to manage which subplot to draw on

    for (user_type, year), df in df_dict.items():
        peak_hours = df['start_hour'].value_counts(normalize=True).sort_index()
        cdf = peak_hours.cumsum()

        # Select the right subplot
        ax1 = axes[ax_idx // 2, ax_idx % 2]
        sns.barplot(x=peak_hours.index, y=peak_hours.values, ax=ax1, alpha=0.6, color=colors[ax_idx])
        ax1.set_title(f'Peak Start Hours and CDF for {user_type} Riders in {year}')
        ax1.set_xlabel('Hour of Day')
        ax1.set_ylabel('Proportion of Rides')
        ax1.tick_params(axis='y', labelcolor=colors[ax_idx])

        # Create a twin axis for the CDF
        ax2 = ax1.twinx()
        sns.lineplot(x=cdf.index, y=cdf.values, ax=ax2, color='red', marker='o', sort=False)
        ax2.set_ylabel('Cumulative Proportion')
        ax2.tick_params(axis='y', labelcolor='red')

        ax_idx += 1

    plt.tight_layout()
    plt.show()

# Example usage with hypothetical dataframes
df_dict = {
    ('Member', '2022'): member_2022,
    ('Casual', '2022'): casual_2022,
    ('Member', '2023'): member_2023,
    ('Casual', '2023'): casual_2023
}
plot_peak_hours_combined(df_dict)
```



Findings:

1. Across 2022 and 2023, 65% of members consistently started their rides between 6 AM and 8 PM.

Inference: This trend indicates that the majority of members likely use the service for daily routines or commuting purposes.

[ ]:

[ ]:

0.7.6 Results:

Over all we identified 5 factors that could differentiate differences between casual riders (non-premium riders) & premium riders, These differences can be used by marketing teams to devise marketing strategies to convert the casual riders into premium riders which could improve revenue.

[ ]:

0.7.7 Reflection:

From this project we learned about team work, collaboration and effective presentation skills. Also, we learned about data graphing techniques, how to formulate research question, how to conduct analysis based on a objective.

Overall, we beleive that the scripting with data analysis class has taught has good tools and techniques which effectively came into use in this project and helped us solve a problem statement

[ ]:
```

```