Cyclistic Bike Share Client Usage Report

August 17, 2024

1 Cyclistic Bike Share Data Analysis Report

2 Table of Contents

2.1 1. Introduction

Welcome to the Cyclistic bike-share analysis case study! In this report, we explore how Cyclistic, a bike-share company in Chicago, can maximize the number of annual memberships by understanding the differences in usage between casual riders and annual members. This analysis aims to provide actionable insights to design an effective marketing strategy to convert casual riders into annual members.

2.2 2. Data Overview

Data Source: The data-set used for this analysis comprises 12 monthly data .csv files from July 2023 to June 2024.

Python: For data cleaning and analysis

Tableau: For visualization and dashboard creation

2.3 3. Data Preparation

2.3.1 3.1. Data Cleaning before merging

In this section, the 202406.csv file was cleaned by standardizing the date format and performing the necessary transformations to prepare the data for analysis. These steps were required to ensure consistency with the other files.

[ ]:
```
import pandas as pd
```

```
# Read the original 202406.csv file
file_path = '202406.csv'
df = pd.read_csv(file_path)

# Convert the started_at and ended_at columns to a consistent format␣
 ↪(yyyy-mm-dd HH:MM:SS)
df['started_at'] = pd.to_datetime(df['started_at'], format='%m/%d/%Y %H:%M').dt.
 ↪strftime('%Y-%m-%d %H:%M:%S')
df['ended_at'] = pd.to_datetime(df['ended_at'], format='%m/%d/%Y %H:%M').dt.
 ↪strftime('%Y-%m-%d %H:%M:%S')

# Save the cleaned file
cleaned_file_path = 'cleaned202406.csv'
df.to_csv(cleaned_file_path, index=False)

print(f'Cleaned file saved as {cleaned_file_path}')
```

```
Cleaned file saved as cleaned202406.csv
```

2.4 3.2 Data Merging

All monthly datasets were combined into a single data frame. It was confirmed that the June file was loaded correctly. Additionally, the data structure was explored to identify any errors or inconsistencies.

[ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os
import warnings
from datetime import datetime

warnings.filterwarnings("ignore", category=FutureWarning)


# Load the data from different files
file_paths = [
    '202307.csv', '202308.csv', '202309.csv', '202310.csv', '202311.csv',␣
 ↪'202312.csv',
    '202401.csv', '202402.csv', '202403.csv', '202404.csv', '202405.csv',␣
 ↪'cleaned202406.csv'
]

# Merging data into one DataFrame
cyclistic_bike_share = pd.concat([pd.read_csv(file) for file in file_paths],␣
 ↪ignore_index=True)
```

```
# Confirm that the June file is correctly loaded
print("June data loaded:")
print(cyclistic_bike_share[cyclistic_bike_share['started_at'].str.
 ↪contains('2024-06')])

#Explore the structure of the data
print(cyclistic_bike_share.head())
print(cyclistic_bike_share.info())
```

```
June data loaded:
                  ride_id  rideable_type           started_at  \
5023660  CDE6023BE6B11D2F  electric_bike  2024-06-11 17:20:00
5023661  462B48CD292B6A18  electric_bike  2024-06-11 17:19:00
5023662  9CFB6A858D23ABF7  electric_bike  2024-06-11 17:25:00
5023663  6365EFEB64231153  electric_bike  2024-06-11 11:53:00
5023664  BA0323C33134CBA8  electric_bike  2024-06-11 00:11:00
…                     …              …                   …
5734376  1D1EBE57758FB1EE  electric_bike  2024-06-11 08:25:00
5734377  2F63E9CD01D79515  electric_bike  2024-06-24 11:40:00
5734378  97D225818F9C7AC3  electric_bike  2024-06-30 10:43:00
5734379  C8D2A48B901F7399  electric_bike  2024-06-11 18:20:00
5734380  C372E7A1A7BA19D4  electric_bike  2024-06-15 15:48:00

                   ended_at                start_station_name  \
5023660  2024-06-11 17:21:00                               NaN
5023661  2024-06-11 17:19:00                               NaN
5023662  2024-06-11 17:30:00                               NaN
5023663  2024-06-11 12:08:00                               NaN
5023664  2024-06-11 00:11:00                               NaN
…                        …                                 …
5734376  2024-06-11 08:33:00    Ravenswood Ave & Lawrence Ave
5734377  2024-06-24 11:42:00          Damen Ave & Leland Ave
5734378  2024-06-30 10:45:00          Damen Ave & Leland Ave
5734379  2024-06-11 18:29:00  Pine Grove Ave & Irving Park Rd
5734380  2024-06-15 15:52:00    Ravenswood Ave & Lawrence Ave

        start_station_id             end_station_name end_station_id  \
5023660              NaN                          NaN            NaN
5023661              NaN                          NaN            NaN
5023662              NaN                          NaN            NaN
5023663              NaN                          NaN            NaN
5023664              NaN                          NaN            NaN
…                    …                            …              …
5734376     TA1309000066  Campbell Ave & Montrose Ave          15623
5734377     TA1307000158                          NaN            NaN
5734378     TA1307000158                          NaN            NaN
5734379     TA1308000022                          NaN            NaN
```

```
5734380     TA1309000066                           NaN           NaN

         start_lat  start_lng    end_lat    end_lng member_casual
5023660  41.890000 -87.650000  41.890000 -87.650000        casual
5023661  41.890000 -87.650000  41.890000 -87.650000        casual
5023662  41.930000 -87.650000  41.940000 -87.650000        casual
5023663  41.880000 -87.640000  41.880000 -87.640000        casual
5023664  41.940000 -87.640000  41.940000 -87.640000        casual
…             …         …         …         …            …
5734376  41.968466 -87.674225  41.961524 -87.691177        member
5734377  41.967121 -87.679127  41.970000 -87.670000        member
5734378  41.967154 -87.679091  41.970000 -87.680000        member
5734379  41.954404 -87.647983  41.930000 -87.640000        member
5734380  41.968486 -87.674196  41.970000 -87.660000        member

[710510 rows x 13 columns]
            ride_id  rideable_type           started_at             ended_at  \
0  9340B064F0AEE130  electric_bike  2023-07-23 20:06:14  2023-07-23 20:22:44
1  D1460EE3CE0D8AF8   classic_bike  2023-07-23 17:05:07  2023-07-23 17:18:37
2  DF41BE31B895A25E   classic_bike  2023-07-23 10:14:53  2023-07-23 10:24:29
3  9624A293749EF703  electric_bike  2023-07-21 08:27:44  2023-07-21 08:32:40
4  2F68A6A4CDB4C99A   classic_bike  2023-07-08 15:46:42  2023-07-08 15:58:08

         start_station_name start_station_id  \
0     Kedzie Ave & 110th St            20204
1   Western Ave & Walton St     KA1504000103
2   Western Ave & Walton St     KA1504000103
3  Racine Ave & Randolph St            13155
4     Clark St & Leland Ave     TA1309000014

                     end_station_name end_station_id  start_lat  start_lng  \
0  Public Rack - Racine Ave & 109th Pl            877  41.692406 -87.700905
1            Milwaukee Ave & Grand Ave          13033  41.898418 -87.686596
2               Damen Ave & Pierce Ave   TA1305000041  41.898418 -87.686596
3              Clinton St & Madison St   TA1305000032  41.884112 -87.656943
4                     Montrose Harbor   TA1308000012  41.967088 -87.667291

     end_lat    end_lng member_casual
0  41.694835 -87.653041        member
1  41.891578 -87.648384        member
2  41.909396 -87.677692        member
3  41.882752 -87.641190        member
4  41.963982 -87.638181        member
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5734381 entries, 0 to 5734380
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           float64
 10  end_lat             float64
 11  end_lng             float64
 12  member_casual       object
dtypes: float64(4), object(9)
memory usage: 568.7+ MB
None
```

2.4.1 4. Data Cleaning

The data was further cleaned by converting date-time columns, handling errors, and performing various transformations to ensure data consistency.

2.4.2 4.1. Identified Issues

After the structure of the data was confirmed, several issues were detected:

Date and Time Format: The started_at and ended_at columns, which were in date-time format, were converted to datetime objects. Rows where datetime conversion failed were dropped.

Column Naming: The column names started_at and ended_at were found to be insufficiently descriptive. They were renamed to improve clarity.

Defining Days and Months: Columns for days of the week and months were added to facilitate better temporal analysis.

Trip Duration Calculation: A new column was created to calculate the duration of each trip. Negative values were removed to ensure accuracy, and rows with negative trip durations were dropped.

Missing Station Information: Missing station names and IDs were addressed. If either the name or ID was provided, the missing counterpart was tracked. However, if both were missing, the data was filtered out due to unreliable coordinates that did not match any station in several cases.

Handling Incomplete Data: Incomplete data were removed to avoid errors in analysis.

Removing Duplicates: Duplicate records were removed to ensure that each entry was unique.

[ ]:
```
# Convert date-time columns to datetime objects, handling errors
cyclistic_bike_share['started_at'] = pd.
 ↪to_datetime(cyclistic_bike_share['started_at'], errors='coerce')
cyclistic_bike_share['ended_at'] = pd.
 ↪to_datetime(cyclistic_bike_share['ended_at'], errors='coerce')
```

```
# Check for NaT values in datetime columns
print("NaT values in 'started_at':", cyclistic_bike_share['started_at'].isna().
 ↪sum())
print("NaT values in 'ended_at':", cyclistic_bike_share['ended_at'].isna().
 ↪sum())

# Drop rows where datetime conversion failed
cyclistic_bike_share = cyclistic_bike_share.dropna(subset=['started_at',␣
 ↪'ended_at'])

# Rename columns
cyclistic_bike_share = cyclistic_bike_share.rename(columns={
    'started_at': 'initial_time',
    'ended_at': 'final_time',
    'member_casual': 'client'
})

# Extract day of the week and month
cyclistic_bike_share['day_of_week'] = cyclistic_bike_share['initial_time'].dt.
 ↪day_name()
cyclistic_bike_share['month'] = cyclistic_bike_share['initial_time'].dt.
 ↪month_name()

# Create a new column for trip duration
cyclistic_bike_share['trip_time_minutes'] = (cyclistic_bike_share['final_time']␣
 ↪- cyclistic_bike_share['initial_time']).dt.total_seconds() / 60

# Filter out negative trip times
cyclistic_bike_share = cyclistic_bike_share.query('trip_time_minutes >= 0')

# Filter rows with missing station names or IDs
cyclistic_bike_share = cyclistic_bike_share.query('start_station_name != "" or␣
 ↪start_station_id != "" and end_station_name != "" or end_station_id != ""')

# Drop rows with missing values
cyclistic_bike_share = cyclistic_bike_share.dropna()

# Drop duplicates
cyclistic_bike_share = cyclistic_bike_share.drop_duplicates()

# Clean column names
cyclistic_bike_share.columns = [col.lower().replace(' ', '_') for col in␣
 ↪cyclistic_bike_share.columns]

# Order days of the week and months
```

```
#-------------------------------------------------------------------------------

# Define the order for days of the week
days_order = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday',␣
 ↪'Saturday', 'Sunday']
months_order = ['January', 'February', 'March', 'April', 'May', 'June', 'July',␣
 ↪'August', 'September', 'October', 'November', 'December']

# Convert to categorical with specified order
cyclistic_bike_share['day_of_week'] = pd.
 ↪Categorical(cyclistic_bike_share['day_of_week'], categories=days_order,␣
 ↪ordered=True)
cyclistic_bike_share['month'] = pd.Categorical(cyclistic_bike_share['month'],␣
 ↪categories=months_order, ordered=True)

# Confirm the cleaned and ordered data
print(cyclistic_bike_share.head())
print(cyclistic_bike_share.info())
```

```
NaT values in 'started_at': 0
NaT values in 'ended_at': 0
            ride_id  rideable_type        initial_time          final_time  \
0  9340B064F0AEE130  electric_bike 2023-07-23 20:06:14 2023-07-23 20:22:44
1  D1460EE3CE0D8AF8   classic_bike 2023-07-23 17:05:07 2023-07-23 17:18:37
2  DF41BE31B895A25E   classic_bike 2023-07-23 10:14:53 2023-07-23 10:24:29
3  9624A293749EF703  electric_bike 2023-07-21 08:27:44 2023-07-21 08:32:40
4  2F68A6A4CDB4C99A   classic_bike 2023-07-08 15:46:42 2023-07-08 15:58:08

         start_station_name start_station_id  \
0     Kedzie Ave & 110th St            20204
1   Western Ave & Walton St     KA1504000103
2   Western Ave & Walton St     KA1504000103
3  Racine Ave & Randolph St            13155
4     Clark St & Leland Ave     TA1309000014

                     end_station_name end_station_id  start_lat  start_lng  \
0  Public Rack - Racine Ave & 109th Pl            877  41.692406 -87.700905
1            Milwaukee Ave & Grand Ave          13033  41.898418 -87.686596
2               Damen Ave & Pierce Ave   TA1305000041  41.898418 -87.686596
3              Clinton St & Madison St   TA1305000032  41.884112 -87.656943
4                      Montrose Harbor   TA1308000012  41.967088 -87.667291

     end_lat    end_lng  client day_of_week month  trip_time_minutes
0  41.694835 -87.653041  member      Sunday  July          16.500000
1  41.891578 -87.648384  member      Sunday  July          13.500000
2  41.909396 -87.677692  member      Sunday  July           9.600000
3  41.882752 -87.641190  member      Friday  July           4.933333
```

```
4  41.963982 -87.638181  member    Saturday  July          11.433333
<class 'pandas.core.frame.DataFrame'>
Index: 4274279 entries, 0 to 5734376
Data columns (total 16 columns):
 #   Column              Dtype
---  ------              -----
 0   ride_id             object
 1   rideable_type       object
 2   initial_time        datetime64[ns]
 3   final_time          datetime64[ns]
 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  client              object
 13  day_of_week         category
 14  month               category
 15  trip_time_minutes   float64
dtypes: category(2), datetime64[ns](2), float64(5), object(7)
memory usage: 497.3+ MB
None
```

2.5 5. Descriptive Analysis

A descriptive analysis was conducted on trip durations, including calculations of mean, median, maximum, and minimum times for both member and casual users. Ridership patterns were analyzed by day of the week and month, with visualizations depicting the number of rides and average trip durations. Weekly and monthly summary data were processed and saved for further analysis.

[ ]:
```
# Descriptive analysis on trip_time_minutes (all data in minutes)
print(cyclistic_bike_share['trip_time_minutes'].describe())

# Compare members and casual users
mean_duration = cyclistic_bike_share.groupby('client')['trip_time_minutes'].
 ↪mean()
median_duration = cyclistic_bike_share.groupby('client')['trip_time_minutes'].
 ↪median()
max_duration = cyclistic_bike_share.groupby('client')['trip_time_minutes'].max()
min_duration = cyclistic_bike_share.groupby('client')['trip_time_minutes'].min()

print("Mean Duration:\n", mean_duration)
print("Median Duration:\n", median_duration)
print("Max Duration:\n", max_duration)
print("Min Duration:\n", min_duration)
```

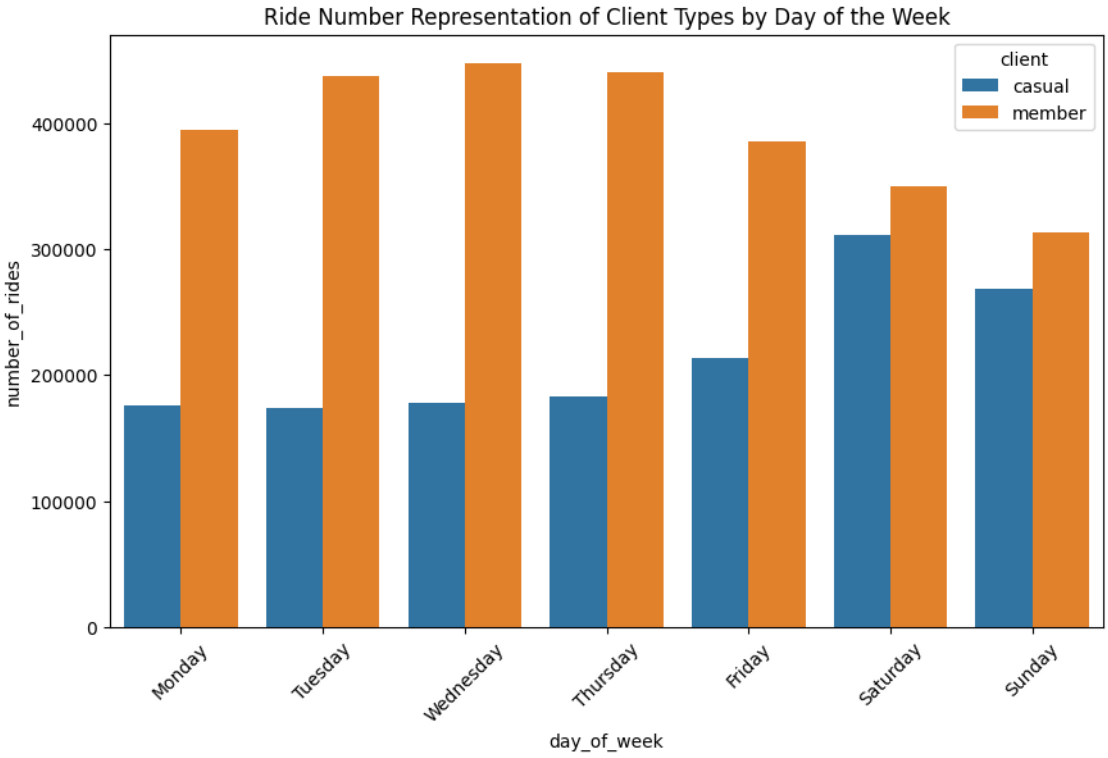
```
count    4.274279e+06
mean     1.651058e+01
std      3.651675e+01
min      0.000000e+00
25%      5.816667e+00
50%      1.001667e+01
75%      1.800000e+01
max      6.891217e+03
Name: trip_time_minutes, dtype: float64
Mean Duration:
 client
casual    23.902368
member    12.498952
Name: trip_time_minutes, dtype: float64
Median Duration:
 client
casual    13.216667
member     8.850000
Name: trip_time_minutes, dtype: float64
Max Duration:
 client
casual    6891.216667
member    1497.650000
Name: trip_time_minutes, dtype: float64
Min Duration:
 client
casual    0.0
member    0.0
Name: trip_time_minutes, dtype: float64
```

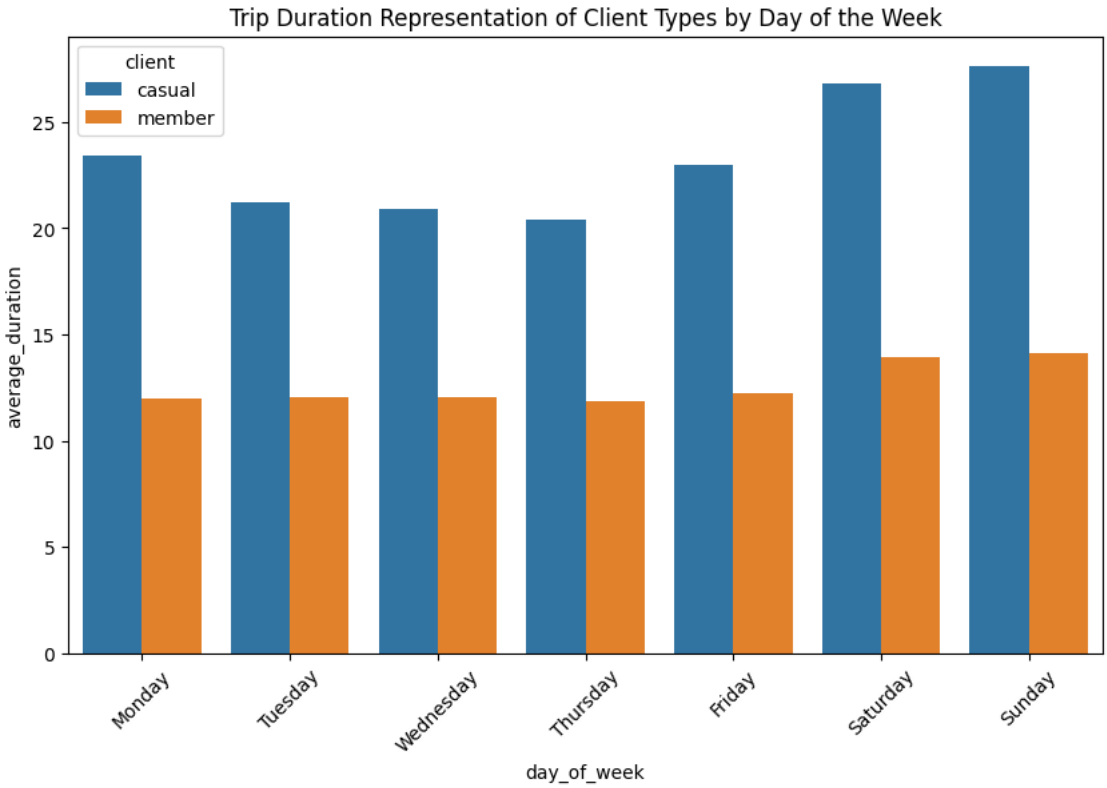
[ ]:
```
# Analyze ridership data by type and weekday
weekday_ridership = (cyclistic_bike_share
                     .groupby(['client', 'day_of_week'])
                     .agg(number_of_rides=('trip_time_minutes', 'count'),
                          average_duration=('trip_time_minutes', 'mean'))
                     .reset_index()
                     .sort_values(by=['client', 'day_of_week']))

# Visualize the number of rides by rider type and day of the week
plt.figure(figsize=(10, 6))
sns.barplot(data=weekday_ridership, x='day_of_week', y='number_of_rides',␣
 ↪hue='client')
plt.title('Ride Number Representation of Client Types by Day of the Week')
plt.xticks(rotation=45)
plt.show()

# Visualize the average duration by rider type and day of the week
```

```
plt.figure(figsize=(10, 6))
sns.barplot(data=weekday_ridership, x='day_of_week', y='average_duration',␣
 ↪hue='client')
plt.title('Trip Duration Representation of Client Types by Day of the Week')
plt.xticks(rotation=45)
plt.show()
```



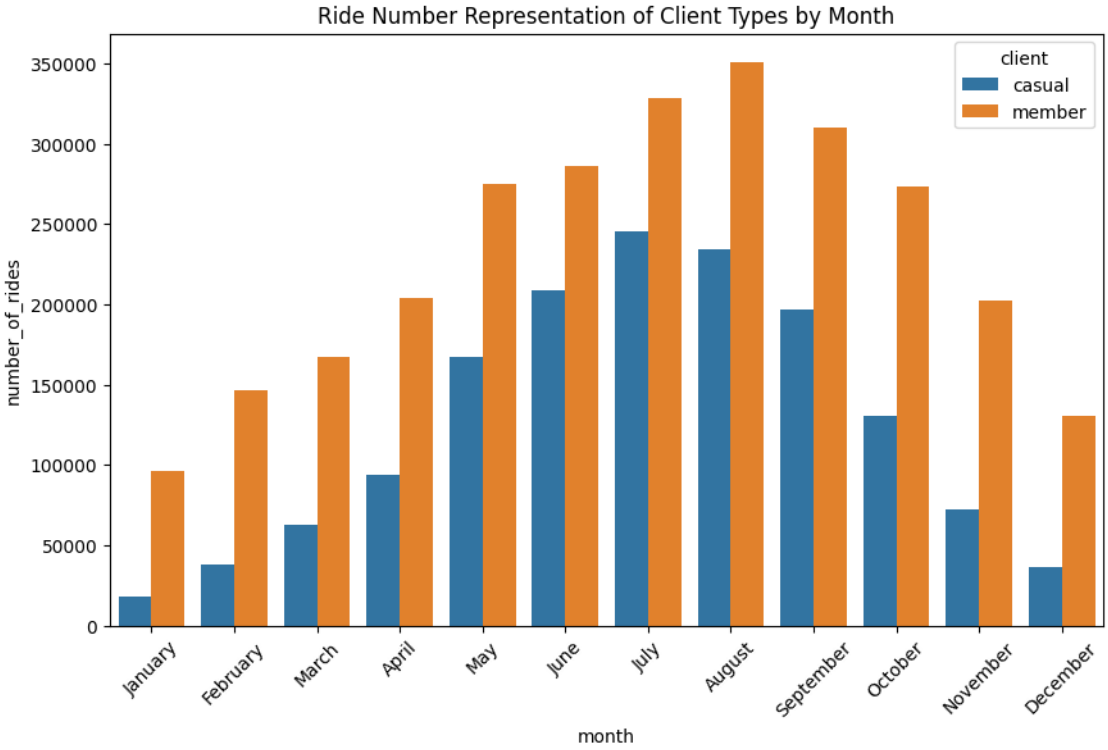


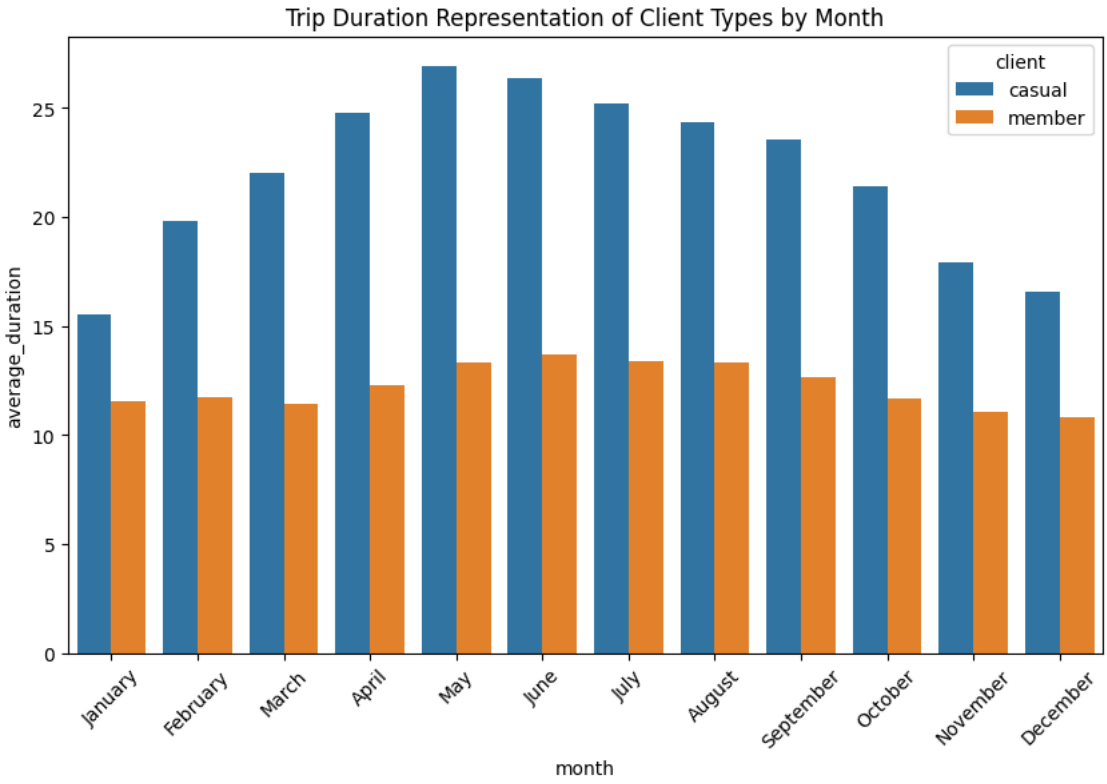
[ ]:
```
# Analyze ridership data by type and month
monthly_ridership = (cyclistic_bike_share
                     .groupby(['client', 'month'])
                     .agg(number_of_rides=('trip_time_minutes', 'count'),
                          average_duration=('trip_time_minutes', 'mean'))
                     .reset_index()
                     .sort_values(by=['client', 'month']))

# Visualize the number of rides by rider type and month
plt.figure(figsize=(10, 6))
sns.barplot(data=monthly_ridership, x='month', y='number_of_rides',␣
 ↪hue='client')
plt.title('Ride Number Representation of Client Types by Month')
plt.xticks(rotation=45)
plt.show()

# Visualize the average duration by rider type and month
plt.figure(figsize=(10, 6))
sns.barplot(data=monthly_ridership, x='month', y='average_duration',␣
 ↪hue='client')
plt.title('Trip Duration Representation of Client Types by Month')
```

```
plt.xticks(rotation=45)
plt.show()
```





[ ]:
```
# Process the weekly summary data
week_summary = (cyclistic_bike_share
                .groupby(['day_of_week', 'client'])
                .agg(total_cases=('trip_time_minutes', 'count'))
                .reset_index()
                .groupby('day_of_week')
                .apply(lambda df: df.assign(total_cases_day=df['total_cases'].
 ↪sum()))
                .assign(percentage=lambda df: (df['total_cases'] /␣
 ↪df['total_cases_day']) * 100)
                .reset_index(drop=True))

# Process the monthly summary data
monthly_summary = (cyclistic_bike_share
                   .groupby(['month', 'client'])
                   .agg(total_cases=('trip_time_minutes', 'count'))
                   .reset_index()
                   .groupby('month')
                   .apply(lambda df: df.
 ↪assign(total_cases_month=df['total_cases'].sum()))
```

```
                    .assign(percentage=lambda df: (df['total_cases'] /␣
 ↪df['total_cases_month']) * 100)
                    .reset_index(drop=True))

# Save the processed summary data
week_summary.to_csv('weekly_summary.csv', index=False)
monthly_summary.to_csv('monthly_summary.csv', index=False)

print('Weekly and monthly summary data saved as weekly_summary.csv and␣
 ↪monthly_summary.csv')
```

2.6 6. Recommendations

Based on the analysis, we recommend the following actions:

Targeted Promotions: Increase marketing efforts on weekends and the most popular days for casual riders to encourage them to sign up for annual memberships. **Time-Based Incentives:** Offer incentives for rides during off-peak hours to attract more members. **Monthly Campaigns:** Launch monthly campaigns highlighting the benefits of annual memberships, especially during months with lower membership growth.

2.7 7. Conclusion

This analysis provides valuable insights into the usage patterns of Cyclistic's bike-share system. By understanding these patterns, Cyclistic can design targeted marketing strategies to increase annual memberships and enhance overall user satisfaction.

2.8 8. Data Usage Note

Data Licence Link (here)

Data Source: The data used in this dashboard is sourced from the Divvy bike-sharing service, operated by Lyft Bikes and Scooters, LLC in partnership with the City of Chicago. **Data-sets** here

Purpose: This report was created for educational and portfolio demonstration purposes only. It is not intended for commercial use.

Data Usage: The data has been integrated into this analysis to showcase data visualization and analytical skills. The data itself is not sold or distributed as a standalone product.

Affiliation: This work is independent and not affiliated with, endorsed by, or sponsored by Bike-share or the City of Chicago.

Trademark Notice: No logos or trademarks of Divvy or Bikeshare are used in this dashboard. All trademarks and logos are the property of their respective owners.

Data Access: The data was accessed through authorized channels, including the provided API and data download options.