Bicycle Pass Analysis

Team

Demetrios Daniel

Yougender Chauhan

California State University,

Los Angeles

CIS 5270

Business Intelligence

Spring 2023

A: Introduction

Bicycle sharing systems have become increasingly popular in urban areas around the world. In recent years, Chicago-based company Cyclistic has seen a rise in the number of people using their bikes. Cyclistic have a significant number of casual riders who use their bikes on a short-term basis, such as for daily commuting, running errands, pleasure, or business since some people use it to commute to work. However, the company has also been successful in attracting annual members, who purchase a yearly subscription that provides them with unlimited access to the bikes.

Cyclistic's financial department has conducted an analysis that determined annual passes for riders to be more profitable than daily passes. The company's management team is interested in exploring this more profitable avenue and aims to capitalize on key indicators that can push a customer towards buying an annual subscription instead of a daily one. It is of high importance to identify differences in service usage between casual and annual members. The goal is to identify patterns that could potentially turn casual riders into annual riders by purchasing annual tickets.

In this analysis, we will be using Python to examine Cyclistic's bicycle usage data. The data includes information on all bike rides taken in the first quarter of 2021 by both casual and annual riders. The data was collected from a very detailed repository with data dating all the way back in 2004. For the purposes of this project, however, we will concentrate on a specific time frame only which will be Q2-2019 - Q1-2020. The reason for this decision being in the nature of our analysis. Since we are aiming to understand the current differences in bike service usage, we determined the most recent data to be more beneficial in analyzing instead of going further back.

We could make use of older data if the objective of our project would be different in a way that looking back through the years could benefit our aim.

We will begin by importing and exploring the data to gain an understanding of the variables and their relationships. Afterwards, we will conduct our analysis to identify patterns in the data, as well as output visualizations to make it easier for the executives to understand when presented with the information. Finally, this analysis will be used to identify factors that influence whether a rider is likely to become an annual member.

Using Python to analyze the data will enable us to conduct a robust and efficient analysis. The combination of our findings along with the visualizations can provide us, and anyone who reads this paper, a comprehensive understanding of the data and enable us to build a model that can identify key factors that influence rider behaviour. Ultimately, this analysis will provide Cyclistic with the information it needs to improve its services and attract more annual members.

The results of this analysis will be used to inform Cyclistic's marketing and help their strategy development team form a course of action moving forward. By understanding these factors, the company can create targeted marketing campaigns that encourage casual riders to purchase annual passes. This, in turn, will increase the company's revenue and help it continue to provide high-quality bike-sharing services to the people of Chicago.

B: Dataset Urls and Dataset Description

Dataset Link: https://divvy-tripdata.s3.amazonaws.com/index.html

As mentioned above in our Introduction section, the specific timeframe we will be looking into would be a collection of the quarterly datasets Q2-2019 - Q1-2020 for the reasons explained

earlier. Had the objective of this analysis been different, then older data would most definitely be used.

Dataset Description:

Field Name	Description	Example Value
Trip_id	A unique identification number given to every trip	21742455
start_time	The date and time of the start of the trip	1/1/2019 12:04:37 AM
end_time	The date and time at the end of every trip	1/1/2019 12:20:21 AM
bike_id	A unique identification number given to every bike	2167
trip_duration	The duration of the trip in seconds	1,727
from_station_id	A unique identification number given to the station the bike was picked up from	211
from_station_name	The name of the station the bike was picked up from	Millennium Park
to_station_id	A unique identification number given to the station the bike was taken to	644
to_station_name	The name of the station the bike was taken to	Ellis Ave & 55th Street
user_type	The type of service used based on subscription	Customer
gender	Gender of the customer using the service	Male
birth_year	The customer's year of birth	1992
is_member	Type of customer membership	Casual

ride_duration (in minutes)	The duration of the trip in Minutes	2
weekday	The day of the week in range from 0-6	5
month	The month in numeric format	4
hour	The hour of the	0

C: Data Cleaning

At first we will proceed with importing our libraries for processing, analyzing, and visualizing our data, as well as importing our datasets shortly after.

In [5]:
```
# Importing libraries
import pandas as pd
import datetime as dt
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.io as pio

# Making exporting interactive graphs possible
pio.renderers.default='notebook'
# Setting aesthetic parameters for plots
plt.rcParams['figure.figsize'] = (16, 8)
pio.templates.default = 'presentation'
sns.set_style('whitegrid')
# Turning off scientific notation in displaying numbers
pd.set_option('display.float_format', '{:.5f}'.format)
# Turning off warnings for chained assignments
pd.options.mode.chained_assignment = None
```

In [6]:
```
# Importing data and declaring variables corresponding to different quarters
q2_2019 = pd.read_csv('/Users/Demetris Daniel/Desktop/5270/pythonproject/Divvy_Trips_2019_Q2.csv')
q3_2019 = pd.read_csv('/Users/Demetris Daniel/Desktop/5270/pythonproject/Divvy_Trips_2019_Q3.csv')
q4_2019 = pd.read_csv('/Users/Demetris Daniel/Desktop/5270/pythonproject/Divvy_Trips_2019_Q4.csv')
q1_2020 = pd.read_csv('/Users/Demetris Daniel/Desktop/5270/pythonproject/Divvy_Trips_2020_Q1.csv')
```

After importing our datasets, let's look at the column names and examine the imported data frames more closely.

```
In [7]: # List comprehension for printing the column names of each dataframe
        [data.columns for data in [q2_2019, q3_2019, q4_2019, q1_2020]]
```

```
Out[7]: [Index(['01 - Rental Details Rental ID', '01 - Rental Details Local Start Time',
                '01 - Rental Details Local End Time', '01 - Rental Details Bike ID',
                '01 - Rental Details Duration In Seconds Uncapped',
                '03 - Rental Start Station ID', '03 - Rental Start Station Name',
                '02 - Rental End Station ID', '02 - Rental End Station Name',
                'User Type', 'Member Gender',
                '05 - Member Details Member Birthday Year'],
               dtype='object'),
         Index(['trip_id', 'start_time', 'end_time', 'bikeid', 'tripduration',
                'from_station_id', 'from_station_name', 'to_station_id',
                'to_station_name', 'usertype', 'gender', 'birthyear'],
               dtype='object'),
         Index(['trip_id', 'start_time', 'end_time', 'bikeid', 'tripduration',
                'from_station_id', 'from_station_name', 'to_station_id',
                'to_station_name', 'usertype', 'gender', 'birthyear'],
               dtype='object'),
         Index(['ride_id', 'rideable_type', 'started_at', 'ended_at',
                'start_station_name', 'start_station_id', 'end_station_name',
                'end_station_id', 'start_lat', 'start_lng', 'end_lat', 'end_lng',
                'member_casual'],
               dtype='object')]
```

The good news is the abundance of data. The bad news is the lack of standardization, as well as the irregularity in column numbers among the datasets. It seems some datasets contain columns that do not exist in others. To analyze the best and most consistent way possible, let's only include columns and work with features that are common property of all datasets currently imported. This means we would have to leave out information pertaining to columns such as geography, gender, and birth year since they do not appear on all datasets. Let's take a look at the q2_2019 dataset and rename it in accordance to the q_2020 dataset since it is the most recent and compliant.

```
In [262]: # Looking at a sample of the q2_2019 dataset
          q2_2019.sample(2)
```

Out[262]:

	01 - Rental Details Rental ID	01 - Rental Details Local Start Time	01 - Rental Details Local End Time	01 - Rental Details Bike ID	01 - Rental Details Duration In Seconds Uncapped	03 - Rental Start Station ID	03 - Rental Start Station Name	02 - Rental End Station ID	02 - Rental End Station Name	User Type	Member Gender	05 - Member Details Member Birthday Year
469839	22732439	2019-05-20 08:00:12	2019-05-20 08:09:33	5261	561.0	191	Canal St & Monroe St (*)	53	Wells St & Huron St	Subscriber	Male	1981.00000
146104	22350005	2019-04-17 18:33:41	2019-04-17 18:38:29	6079	288.0	321	Wabash Ave & 9th St	394	Clark St & 9th St (AMLI)	Subscriber	Male	1968.00000

In [263]:
```
# Renaming the q2_2019 columns in accordance with the 2020 standard
q2_2019 = \
q2_2019.rename(columns={
    '01 - Rental Details Rental ID': 'ride_id',
    '01 - Rental Details Local Start Time': 'started_at',
    '01 - Rental Details Local End Time': 'ended_at',
    '03 - Rental Start Station ID': 'start_station_id',
    '03 - Rental Start Station Name': 'start_station_name',
    '02 - Rental End Station ID': 'end_station_id',
    '02 - Rental End Station Name': 'end_station_name',
    'User Type': 'member_casual'
})
```

In [264]:
```
# Rewriting the dataframe so as to only include columns that also
# exist in the q1_2020 dataframe
q2_2019 = \
q2_2019.loc[:, q2_2019.columns.isin(q1_2020.columns)]
# Validating the result
q2_2019.sample(5)
```

Out[264]:

	ride_id	started_at	ended_at	start_station_id	start_station_name	end_station_id	end_station_name	member_casual
232411	22450970	2019-04-25 19:48:04	2019-04-25 20:06:39	88	Racine Ave & Randolph St	255	Indiana Ave & Roosevelt Rd	Subscriber
696404	22997703	2019-06-05 12:14:52	2019-06-05 13:59:57	334	Lake Shore Dr & Belmont Ave	85	Michigan Ave & Oak St	Customer
448936	22708068	2019-05-18 08:21:24	2019-05-18 08:30:08	331	Halsted St & Clybourn Ave (*)	127	Lincoln Ave & Fullerton Ave	Subscriber
959302	23304171	2019-06-22 15:01:40	2019-06-22 15:18:13	244	Ravenswood Ave & Irving Park Rd	632	Clark St & Newport St	Customer
801790	23119885	2019-06-11 17:20:50	2019-06-11 17:40:27	638	Clinton St & Jackson Blvd (*)	176	Clark St & Elm St	Subscriber

Let’s do the same with our remaining datasets as well.

In [265]:
```
# Renaming the q3_2019 columns in accordance with the 2020 standard
q3_2019 = \
q3_2019.rename(columns={
    'trip_id': 'ride_id',
    'start_time': 'started_at',
    'end_time': 'ended_at',
    'from_station_id': 'start_station_id',
    'from_station_name': 'start_station_name',
    'to_station_id': 'end_station_id',
    'to_station_name': 'end_station_name',
    'usertype': 'member_casual'
})
```

In [266]:
```
# Rewriting the dataframe so as to only include columns that also
# exist in the q1_2020 dataframe
q3_2019 = \
q3_2019.loc[:, q3_2019.columns.isin(q1_2020.columns)]
# Validating the result
q3_2019.sample(5)
```

Out[266]:

	ride_id	started_at	ended_at	start_station_id	start_station_name	end_station_id	end_station_name	member_casual
121505	23623108	2019-07-08 13:25:34	2019-07-08 13:33:44	326	Clark St & Leland Ave	294	Broadway & Berwyn Ave	Subscriber
392793	23917234	2019-07-23 17:11:24	2019-07-23 17:19:33	43	Michigan Ave & Washington St	286	Franklin St & Adams St (Temp)	Subscriber
482723	24011817	2019-07-27 23:40:49	2019-07-28 00:33:35	621	Aberdeen St & Randolph St	621	Aberdeen St & Randolph St	Customer
671533	24211192	2019-08-06 15:01:55	2019-08-06 15:05:43	80	Aberdeen St & Monroe St	346	Ada St & Washington Blvd	Subscriber
1614252	25195730	2019-09-29 00:20:42	2019-09-29 00:34:47	637	Wood St & Chicago Ave (*)	58	Marshfield Ave & Cortland St	Subscriber

In [267]:
```
# Renaming the q4_2019 columns in accordance with the 2020 standard
q4_2019 = \
q4_2019.rename(columns={
    'trip_id': 'ride_id',
    'start_time': 'started_at',
    'end_time': 'ended_at',
    'from_station_id': 'start_station_id',
    'from_station_name': 'start_station_name',
    'to_station_id': 'end_station_id',
    'to_station_name': 'end_station_name',
    'usertype': 'member_casual'
})
```

In [268]:
```
# Rewriting the dataframe so as to only include columns that also
# exist in the q1_2020 dataframe
q4_2019 = \
q4_2019.loc[:, q4_2019.columns.isin(q1_2020.columns)]
# Validating the result
q4_2019.sample(5)
```

Out[268]:

	ride_id	started_at	ended_at	start_station_id	start_station_name	end_station_id	end_station_name	member_casual
525723	25774709	2019-11-25 10:03:29	2019-11-25 10:09:06	111	Sedgwick St & Huron St	291	Wells St & Evergreen Ave	Subscriber
80945	25308422	2019-10-06 18:43:32	2019-10-06 18:54:49	56	Desplaines St & Kinzie St	359	Larrabee St & Division St	Subscriber
685430	25943393	2019-12-26 15:26:11	2019-12-26 15:44:13	195	Columbus Dr & Randolph St	192	Canal St & Adams St	Subscriber
675234	25932658	2019-12-23 16:55:15	2019-12-23 17:06:14	414	Canal St & Taylor St	178	State St & 19th St	Subscriber
474573	25720612	2019-11-17 21:58:32	2019-11-17 22:19:44	164	Franklin St & Lake St	220	Clark St & Drummond Pl	Subscriber

Now we will proceed with the 2020 dataset and exclude data not present in the other datasets and after that, we will change the orders of q1_2020 to align with the rest.

In [269]:
```
# Now doing the same for the 2020 dataset to exclude data
# not present in previous datasets
q1_2020 = \
q1_2020.loc[:, q1_2020.columns.isin(q4_2019.columns)]
# Validating the result
q1_2020.sample(5)
```

Out[269]:

	ride_id	started_at	ended_at	start_station_name	start_station_id	end_station_name	end_station_id	member_casual
312156	81314B113A348856	2020-03-03 17:01:27	2020-03-03 17:17:18	Canal St & Monroe St	191	Sheffield Ave & Willow St	93.00000	member
216053	E82D225C53B4D1D0	2020-02-03 18:32:56	2020-02-03 18:43:50	Ogden Ave & Chicago Ave	54	Ada St & Washington Blvd	346.00000	member
369600	4783FB07F79FDACA	2020-03-23 13:33:01	2020-03-23 13:59:57	Damen Ave & Melrose Ave	228	Broadway & Argyle St	295.00000	member
141497	978499F14C7AED3F	2020-01-25 23:28:50	2020-01-25 23:51:01	Richmond St & Diversey Ave	501	Wilton Ave & Belmont Ave	117.00000	member
154288	828642C20357D9CB	2020-02-21 12:57:50	2020-02-21 17:25:06	Columbus Dr & Randolph St	195	Southport Ave & Waveland Ave	227.00000	casual

In [270]:
```
# Changing the order of the columns in q1_2020 to conform to the rest
q1_2020 = \
q1_2020.loc[:, q4_2019.columns]
# Validating the result
q1_2020.sample(5)
```

Out[270]:

	ride_id	started_at	ended_at	start_station_id	start_station_name	end_station_id	end_station_name	member_casual
250917	BD477486D610B098	2020-02-08 13:01:36	2020-02-08 13:12:20	142	McClurg Ct & Erie St	81.00000	Daley Center Plaza	member
247856	D89D5B74A1F50E60	2020-02-06 07:33:53	2020-02-06 07:54:15	230	Lincoln Ave & Roscoe St	60.00000	Dayton St & North Ave	member
87562	E76D501403504591	2020-01-05 12:00:16	2020-01-05 12:05:03	140	Dearborn Pkwy & Delaware Pl	364.00000	Larrabee St & Oak St	member
247867	2BFC6464F33F539F	2020-02-07 13:20:24	2020-02-07 13:37:07	658	Leavitt St & Division St (*)	307.00000	Southport Ave & Clybourn Ave	member
94790	13CD6A58F0E01433	2020-01-03 16:39:14	2020-01-03 16:49:53	191	Canal St & Monroe St	24.00000	Fairbanks Ct & Grand Ave	member

To double check our above tasks' success result, we perform a comparison function that will return a boolean 'True' or 'False' value depending on whether or not they are identical.

```
In [17]: # The all() function returns True only if all boolean values
         # in the array are True
         all(q2_2019.columns == q3_2019.columns) \
         and all(q3_2019.columns == q4_2019.columns) \
         and all(q4_2019.columns == q1_2020.columns)
Out[17]: True
```

Since it was a success, we will now proceed to concatenate all of our data into a single dataframe and continue processing it.

```
In [272]: # Concatenating data into a single dataframe, df
          df = pd.concat([q2_2019, q3_2019, q4_2019, q1_2020])
          # Validating the result
          df.sample(5)
```

Out[272]:

	ride_id	started_at	ended_at	start_station_id	start_station_name	end_station_id	end_station_name	member_casual
182448	6BEDAEC1952EAEB6	2020-02-19 20:56:42	2020-02-19 21:03:52	118	Sedgwick St & North Ave	288.00000	Larrabee St & Armitage Ave	member
288153	12A8BD45DA7D9C40	2020-03-01 12:11:37	2020-03-01 13:29:43	57	Clinton St & Roosevelt Rd	247.00000	Shore Dr & 55th St	casual
1301819	24871612	2019-09-10 08:23:18	2019-09-10 08:27:27	133	Kingsbury St & Kinzie St	91.00000	Clinton St & Washington Blvd	Subscriber
1241507	24808799	2019-09-06 12:16:47	2019-09-06 12:55:47	211	St. Clair St & Erie St	219.00000	Damen Ave & Cortland St	Subscriber
291014	25526793	2019-10-22 19:25:22	2019-10-22 19:32:43	623	Michigan Ave & 8th St	90.00000	Millennium Park	Subscriber

After concatenating our data into a single dataframe, we will proceed to check if there are any duplicate rows or missing values.

```
In [275]: # Counting the total amount of duplicates
          df.duplicated().sum()
Out[275]: 0
```

It appears we do not have any duplicate rows so we will proceed to check for any missing values.

```
In [21]: # Showing the amount of missing values by column
         df.isna().sum()

Out[21]: ride_id              0
         started_at           0
         ended_at             0
         start_station_id     0
         start_station_name   0
         end_station_id       1
         end_station_name     1
         member_casual        0
         dtype: int64
```

This function is helping return the sum of any missing values according to the columns in which they are detected. As we can see, we have a total of 2 missing values, 1 in 'end_station_id' and another in the 'end_station_name' column. As a result, we will have to find out the rows that contain these two missing values.

```
In [22]: # Seeing the rows with the missing values
         df.query('end_station_id.isna() \
                  or end_station_name.isna()')
```

Out[22]:

	ride_id	started_at	ended_at	start_station_id	start_station_name	end_station_id	end_station_name	meml
414426	157EAA4C4A3C8D36	2020-03-16 11:23:36	2020-03-16 11:23:24	675	HQ QR	NaN	NaN	

It seems that the 2 missing values belong in the same row. As a result, we will have to delete the whole row which will allow us to manipulate and work with our data more efficiently.

```
In [23]: # Deleting rows with missing values
         df = df[df.end_station_id.notnull()]
         # Validating the result
         df.isna().sum()

Out[23]: ride_id              0
         started_at           0
         ended_at             0
         start_station_id     0
         start_station_name   0
         end_station_id       0
         end_station_name     0
         member_casual        0
         dtype: int64
```

This function enables us to detect and filter out any rows that contain any missing values from our DataFrame. The second function validates the result by checking if there are any missing

values in each column of the filtered DataFrame. As we can see, our columns do not contain any missing values and with the missing values now deleted, we can examine our data types and even change them if necessary to make our analysis more flexible.

```
In [24]: # Printing column data types
         df.dtypes
```

```
Out[24]: ride_id              object
         started_at           object
         ended_at             object
         start_station_id      int64
         start_station_name   object
         end_station_id      float64
         end_station_name     object
         member_casual        object
         dtype: object
```

With this function, we can determine if the data types are in the best shape they can be for optimal analysis. Knowing the data types is extremely important and of vital significance since it affects how we can perform calculations and manipulations on the data. For example, arithmetic operations and calculations in general can only be performed on numeric data types, while string operations can only be performed on string data types. Based on our result, it needs some further cleaning and modification since it appears the time recording in the columns ‘started_at’ and ‘ended_at’ is as object/string format. By changing it, we can extract month, weekday, and hour values from it, as well as being able to sort time data more efficiently by changing it to the datetime format.

```
In [25]: # Formatting time and date values from string to datetime
         df.started_at = pd.to_datetime(df.started_at, format='%Y-%m-%d %H:%M:%S')
         df.ended_at = pd.to_datetime(df.ended_at, format='%Y-%m-%d %H:%M:%S')
```

The above function allows us to convert the values in the ‘started_at’ and ‘ended_at’ columns, from strings to datetime objects. In this case, the format is '%Y-%m-%d %H:%M:%S', which corresponds to a string in the format 'YYYY-MM-DD HH:MM:SS'.

In addition, we need to convert columns containing integer values stored as decimals to the corresponding format. In other words, convert the float64 data type to int64. From the output above, the ‘end_station_id’ column serves as such an example.

```
In [27]: # Converting the column from float to integer numbers
         df['end_station_id'] = df['end_station_id'].astype('int')
```

To further simplify our columns in terms of the customer type situation, we can turn members, and subscribers into members by combining the two since they are essentially referring to the same thing and are treated the same for the purposes of this analysis. We will first rename the ‘member_casual’ column into ‘is_member’and then convert it to the boolean format from its string format.

```
In [283]: # Renaming the 'member_casual' column
          df = df.rename(columns={
              'member_casual': 'is_member'
          })
```

```
In [29]: # Seeing different values in the 'is_member' column
         df['is_member'].value_counts()

Out[29]: Subscriber    2595461
         Customer       857474
         member         378407
         casual          48479
         Name: is_member, dtype: int64
```

Similarly, the same can be said for the Customer and Casual values since two, essentially refer to casual riders. In other words, the four terms correspond only to two categorical values, member and casual rider.

```
In [30]: # Standardizing different member/non-member classifications as 1 and 0
         df.loc[df['is_member'].isin(['Subscriber', 'member']), 'is_member'] = 1
         df.loc[df['is_member'].isin(['Customer', 'casual']), 'is_member'] = 0
```

```
In [31]: # Transforming the column to boolean type
         df['is_member'] = df['is_member'].astype('bool')
         # Validating the transformation
         df['is_member'].value_counts(normalize=True)
```

```
Out[31]: True    0.76650
         False   0.23350
         Name: is_member, dtype: float64
```

To do that, we had to change the ‘is_member’ column from string to the boolean format in order to return ‘true’ or ‘false’. This way, we made our subsequent analysis much more efficient since the values of the column ‘is_member’ will now just be divided between Membership customers and Casual customers which will make the analysis process much smoother and easier to convey the findings to Cyclistic’s marketing team.

Moving on with our data cleaning process, we can definitely work on the duration aspect of this DataFrame. First, let’s calculate each ride’s duration by subtracting the timestamp of its start from the timestamp of its end.

```
In [32]: # Declaring the duration column as the difference
         # between the columns ended_at and started_at
         df['duration'] = df['ended_at'] - df['started_at']
```

After computing ‘duration’, we can proceed with converting its format to integers and rename it to ‘duration_sec’ to indicate its unit of measure.

In [33]:
```
# Extracting the total number of seconds from the datetime data type
df['duration'] = df['duration'].dt.total_seconds().astype(int)
# Eenaming the column to indicate the unit
df = df.rename(columns={
    'duration': 'duration_sec'
                })
# Validating the result
df.duration_sec.sample(5)
```

Out[33]:
```
845342      378
1312774    1514
682211     1827
1026884    1289
208935      209
Name: duration_sec, dtype: int32
```

For further flexibility, it would be useful to create a separate column called'duration_min' that will contain more rounded numbers in the minute unit of measure.

In [34]:
```
# Extracring and rounding the total number of minutes from the
# Duration_sec column
df['duration_min'] = round(df['duration_sec']/60).astype('int')
```

Now that we have created different duration elements we can easily work with, we can create a separate column for weekdays. For our weekday numbers we will give 0 to Monday and 9 to Sunday.

In [69]:
```
# Extracting the weekday number from the started_at column
# Where 0 is Monday and 6 is Sunday
df['weekday'] = pd.DatetimeIndex(df['started_at']).weekday
# Validating the result
df.weekday.sample(5)
```

Out[69]:
```
40272      3
150767     5
351728     3
465406     4
1076591    2
Name: weekday, dtype: int64
```

Similarly, we will add further flexibility to our analysis by creating an additional column for month with numbers from 1-12, each corresponding to the months of the year.

```
In [36]: # Extracting the month number where 1 is January and 12 is December
         df['month'] = pd.DatetimeIndex(df['started_at']).month
         # Validating the result
         df.month.sample(5)

Out[36]: 60821     10
         643087     6
         801161     8
         511878    11
         412038     5
         Name: month, dtype: int64
```

Since we have extracted months and weekdays, it is only logical to proceed with extracting hours as well.

```
In [37]: # Extracting the hour from the started_at column
         df['hour'] = pd.DatetimeIndex(df['started_at']).hour
         # Validating the result
         df.hour.sample(5)

Out[37]: 439294     16
         152495     16
         1639576    20
         1590831    17
         954594     12
         Name: hour, dtype: int64
```

Adding these columns will undoubtedly allow for a much more flexible analysis that permits us to go more in depth and manipulate different layers and levels of our data for more comprehensive and elaborate results.

One of the columns that is the most likely to produce abnormal values (outliers) could be the duration column. We can check for consistency by computing the minimum and maximum values of ride durations.

```
In [38]: print(f'''Maximum ride duration: \
         {round(df["duration_min"].max()/60)} hours,
         Minimum ride duration: \
         {df["duration_min"].min()} min''')

         Maximum ride duration: 2608 hours,
         Minimum ride duration: -56 min
```

Based on this output, our data most definitely requires to be trimmed. For the purposes of this analysis let's assume that any ride shorter than 40 seconds and longer than 8 hours was not a legitimate use of Cyclistic's bike sharing service. But let's, at first, determine whether or not

rows with outliers such as our minimum and maximum through the above output, comprise a sizable portion of our data. To determine this, we will compute the percentiles of these values that fall below the previously determined cutoff.

In [39]:
```
# Sorting ride durations to find percentiles of chosen cutoff values
duration_sorted = df['duration_sec'].sort_values()
```

In [40]:
```
# Finding the percentiles of chosen cutoff values by
# dividing the index of their position in the sorted Series,
# dividing it by the length of the Series,
# multiplying by 100 and rounding to the second decimal
print(f'''40 seconds is in the \
{round(duration_sorted.searchsorted(40)
/len(duration_sorted)*100, 2)}%-th percentile of values in the\
 duration_sec column.
8 hours is in the \
{round(duration_sorted.searchsorted(8*60*60)
/len(duration_sorted)*100, 2)}%-th percentile.''')
```

```
40 seconds is in the 0.18%-th percentile of values in the duration_sec column.
8 hours is in the 99.86%-th percentile.
```

It seems that such outliers are an extremely rare occurrence in this dataset since 40 seconds is in the 0.18%-th percentile of values in the duration_sec column while 8 hours is in the 99.86%-th percentile.

Let's create a new dataframe without the outliers included to easily calculate the percentage of data that was lost during the outlier trimming.

In [41]:
```
# Creating a dataframe without rows with implausible ride durations
df_cleaned = \
df[
    (df['duration_sec'] >= 40)
    & (df['duration_sec'] <= 8*60*60)
]
```

In [42]:
```
# Finding the share of rows deleted from the dataset, rounding to the third decimal
# and expressing as a percentage
print(f'''\
trimming {round((df.shape[0] - df_cleaned.shape[0])
/df.shape[0], 3)*100}\
% of data''')
```

```
trimming 0.3% of data
```

This is perfectly acceptable and we will now continue with the creation of the new dataframe.

In [43]:
```
# Overwriting the original dataframe variable
# with the cleaned data
df = df_cleaned
```

D: Summary Statistics

In [19]:
```
# Printing dataframe's general information
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3879822 entries, 0 to 426886
Data columns (total 8 columns):
 #   Column              Dtype
---  ------              -----
 0   ride_id             object
 1   started_at          object
 2   ended_at            object
 3   start_station_id    int64
 4   start_station_name  object
 5   end_station_id      float64
 6   end_station_name    object
 7   member_casual       object
dtypes: float64(1), int64(1), object(6)
memory usage: 266.4+ MB
```

The above output provides us with an overview of our available data. The resulting data frame has an astonishing 3.9 million entries for each individual ride recorded. With data as expansive as this one, we can definitely expect interesting results that Cyclistic can use to their profitable advantage.

In [56]:
```
# Printing dataframe's description information
df.describe()
```

Out[56]:

	start_station_id	end_station_id	duration_sec	duration_min	weekday	month	hour
count	3867336.00000	3867336.00000	3867336.00000	3867336.00000	3867336.00000	3867336.00000	3867336.00000
mean	202.39678	203.20800	1102.30619	18.37177	2.84472	7.01720	13.75399
std	156.48676	156.50992	1378.29477	22.97339	1.94621	2.67758	4.70956
min	1.00000	1.00000	40.00000	1.00000	0.00000	1.00000	0.00000
25%	77.00000	77.00000	412.00000	7.00000	1.00000	5.00000	10.00000
50%	174.00000	174.00000	712.00000	12.00000	3.00000	7.00000	15.00000
75%	290.00000	291.00000	1286.00000	21.00000	4.00000	9.00000	17.00000
max	675.00000	675.00000	28780.00000	480.00000	6.00000	12.00000	23.00000

With the df.descriptive method, we generated descriptive statistics of the Panda's DataFrame. We have the various statistical measures for each numerical column in the DataFrame, such as count, mean, standard deviation, minimum, maximum, and quartile values. In this output we can get a general idea and can help us identify potential outliers.The average observation occurs on a weekday between Monday and Friday, indicating that most trips are taken for commuting or work-related purposes. It seems the average hour of the day a trip was taken would be 13:45 with the minimum being 0 (midnight), indicating a higher number of rides happening during the daytime, with fewer in the early morning or late evening since the 25th percentile is 10:00, median is 15:00 and the 75th percentile is 17:00. The average month value is 7 (July), with the minimum being 1 (January), and the maximum 12 (December). The 25th percentile appears to be May, median July, and 75th percentile September, indicating most trips were taken in the summer months. The start and end station IDs have a minimum value of 1 and a maximum value of 675, indicating that the bike share service Cyclistic provides covers a large area with many stations. The minimum duration of a trip is 1 minute, and the maximum duration is 480 minutes (8 hours). This suggests that the bike sharing service is used for a wide range of trip purposes, from short, quick rides to longer, more leisurely trips but with most trips being relatively short.

```
In [326]: print(df['duration_min'].mean())
          18.371766249428546
```

By examining our mean value, we can see that the average duration of a trip is 18.37 minutes, with a standard deviation of 22.97 minutes. This suggests that most trips are relatively short, but there is a wide range of trip durations, with some trips lasting up to several hours.

```
In [325]: print(df['duration_min'].median())
          12.0
```

The median value of the trip duration in minutes is 12 minutes. This means that half of the trips in the dataset lasted for 12 minutes or less, and the remaining half lasted for more than 12 minutes. This reinforces our earlier interpretation that most trips in the dataset are relatively short, with a mean duration of 18.37 minutes in the first quartile (25%) value of 7 minutes. The median duration of 12 minutes suggests that there is a significant cluster of trips in the dataset with durations around 12 minutes.

```
In [327]: print(df['duration_min'].mode())
          0    6
          Name: duration_min, dtype: int32
```

The mode of 0 indicates that there are a significant number of trips with durations of 0 minutes, which is highly unlikely for an actual bike ride value. It is possible that these trps represent cases where a bike was checked out but not actually used, or where there was an error in recording the trip duration. It could also represent very short trips within a single station or movement of a bike through other transportation means for maintenance purposes. The mode of 6 minutes suggests that many trips in the dataset are again, relatively short. THis reinforces the earlier interpretations that most trips have a duration of less than 12 minutes. Shorter trip durations may also indicate that users are using the bike sharing service to run errands or quick trips, rather than for extended sightseeing or exercise. Knowing that most trips in the dataset are short can be useful information for bike share providers to better tailor their services to the needs of users, such as optimizing bike station locations, bike types and providing discounts for frequent or shorter trips. The mode is an extremely useful measure of central tendency and in this case, it has returned the most observed minimum and most observed maximum values it seems.

In order to see which are the recorded values holding the maximum and minimum record though, we have to use a minimum and maximum function.

```
In [328]: print(df['duration_min'].std())
          22.973394674181105
```

A high standard deviation value implies that the data points are dispersed over a wider range and are less concentrated around the mean value. In this case, a standard deviation of 22.97 minutes suggests that there are some trips in the dataset that have much longer durations than the mean duration of 18.37 minutes, which can skew the interpretation of the dataset.

```
In [329]: print(df['end_station_id'].min())
          1
```

With this function we can see that the minimum value the ‘end_station_id’ receives is, unsurprisingly, 1. This means Cyclistic has at least one bike station which was not a point of doubt.

```
In [333]: print(df['start_station_id'].max())
          675
```

Here we can see the maximum value of the ‘start_station_id’ which will turn out the maximum number of stations Cyclistic owns. It seems Cyclistic has a maximum (in other words total) number of 675 stations which is a pretty large number overall.

```
In [ ]: #This function returns the cumulative sum value over any given axis.
```

```
In [66]: print(df['end_station_id'].cumsum(axis=0))
```

```
0               56
1              115
2              289
3              422
4              551
             ...
426880    785872822
426883    785873062
426884    785873272
426885    785873536
426886    785873621
Name: end_station_id, Length: 3867336, dtype: int32
```

With this method, we calculated the sum of the ‘end_station_id’ in our DataFrame.

```
In [70]: #This function rerturns the cumulative product for all values over the given axis.
```

```
In [71]: print(df['start_station_id'].cumprod(axis=0))
```

```
0                 81
1              25677
2            7266591
3          188931366
4        38164135932
               ...
426880             0
426883             0
426884             0
426885             0
426886             0
Name: start_station_id, Length: 3867336, dtype: int64
```

With this method we calculated the cumulative product of the column ‘start_station_id’ in our DataFrame.

```
In [72]: #This function calculates the percentage change between the current and the previous element.
         #It by default calculates the percentage change from the immediate previous row.
```

```
In [73]: print(df['start_station_id'].pct_change(axis=0))
```

```
0            NaN
1        2.91358
2       -0.10726
3       -0.90813
4        6.76923
            ...
426880  -0.70927
426883   0.76923
426884  -0.12422
426885   3.76596
426886  -0.83631
Name: start_station_id, Length: 3867336, dtype: float64
```

With this method we calculated the percentage of change between the current and prior element in our Datadrame. It can be used to analyze the rate of change of a variable over time. The first output appears as 'NaN' since there is no prior element to calculate the percentage of change with. This summary fluctuation shows that the values are not changing at a constant rate. Rather, the values are fluctuating with both positive and negative percentage changes. This could indicate changes in the usage or availability of specific bike stations over time. Further analysis would be needed to determine the causes of these fluctuations.

```
In [20]: # Counting the total amount of duplicates
         df.duplicated().sum()

Out[20]: 0
```

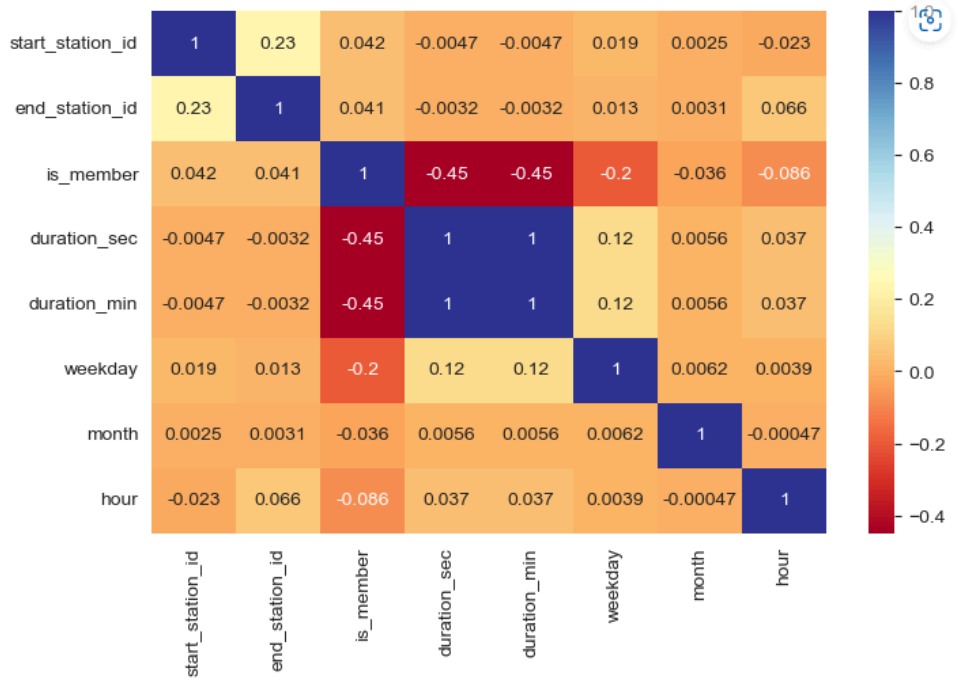
With this function, we are able to receive a count of any duplicated rows. Based on the result, there are no duplicate rows at this point in our DataFrame, meaning we can proceed with the data analysis section.

E: Analysis & Visualizations

Our resulting dataset will center around how casual riders and members differ in their use of Cyclistic's services across months, weekdays, hours, and ride durations. As we proceed with our analysis we decided to plot out a Heatmap to get a general idea of our data's composition.

```
In [77]: #first we plot a heatmap to get a general idea of out dataset for futher analysis
         plt.figure(figsize=(8,5))
         sns.heatmap(df.corr(), annot=True, cmap='RdYlBu')
```

```
Out[77]: <AxesSubplot:>
```



Since the Heatmap might not be the best way to draw results at this stage, we will proceed by analyzing our questions.

1. How many rides per month occur for members and how many for casual riders?

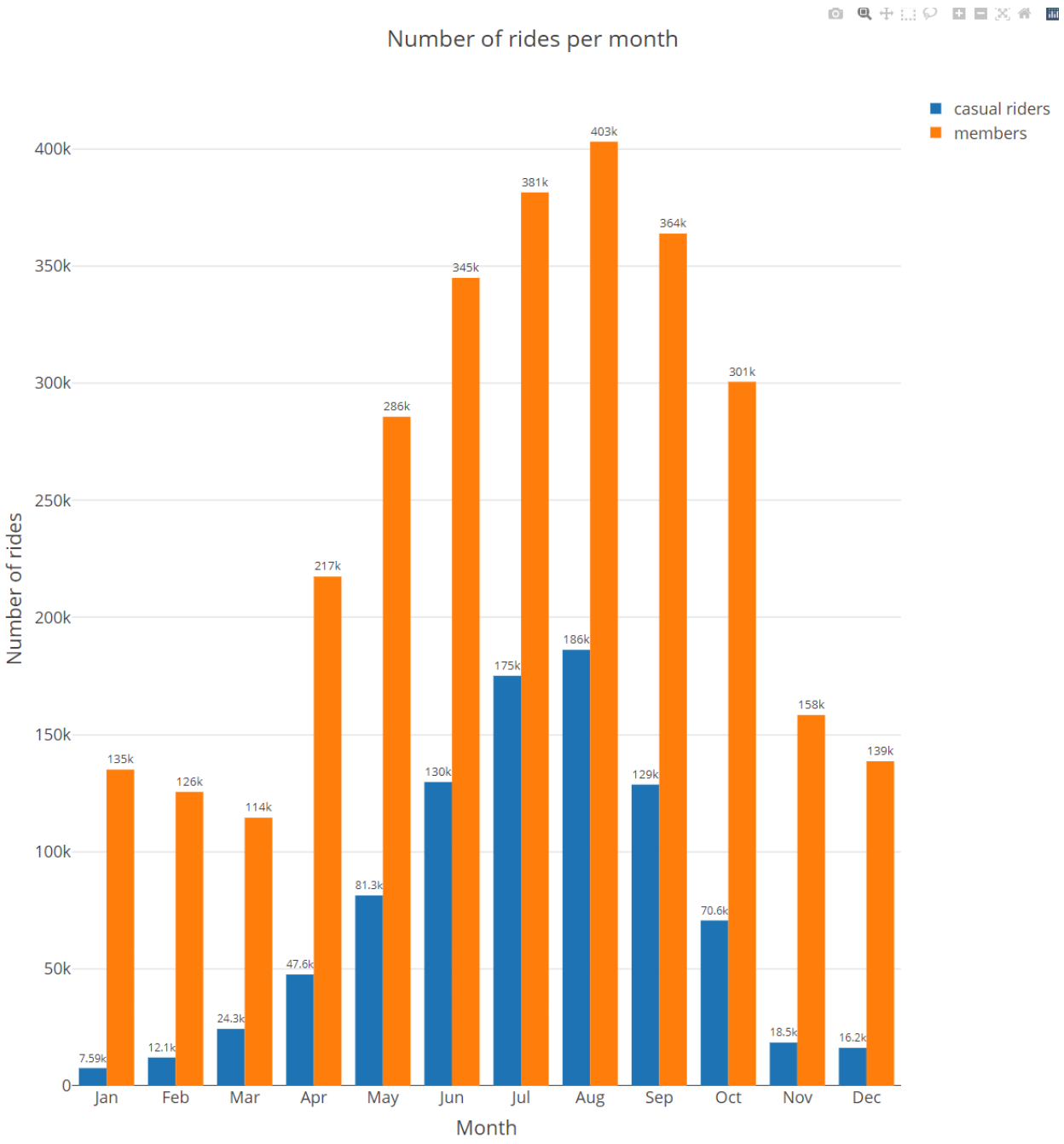
In [46]:
```
# Creating a pivot table for visualization
df_pivot_months = (
    pd.pivot_table(
        # Grouping by month and membership status
        df,
        index=['month', 'is_member'],
        values='ride_id',
        # Counting the number of rides
        aggfunc={
            'ride_id': 'count'
        }).reset_index().rename({'ride_id': 'count'}, axis=1)
    # Replacing boolean values for visualization
    # purposes
    .replace({
        False: 'casual riders',
        True: 'members'
    }))
df_pivot_months['month'] = \
df_pivot_months['month'].replace(
    # Replacing integers with months for better visualization
    {1:'Jan',
     2:'Feb',
     3:'Mar',
     4:'Apr',
     5:'May',
     6:'Jun',
     7:'Jul',
     8:'Aug',
     9:'Sep',
     10:'Oct',
     11:'Nov',
     12:'Dec'})
# Validating the pivot table
df_pivot_months.head(5)
```

Out[46]:

	month	is_member	count
0	Jan	casual riders	7592
1	Jan	members	135057
2	Feb	casual riders	12110
3	Feb	members	125541
4	Mar	casual riders	24335

In [47]:

```
# Plotting differences across months
# with the plotly express library
fig = px.bar(
    df_pivot_months,
    x='month',
    y='count',
    color='is_member',
    # Changing labels for clarity
    labels={
        'count': 'Number of rides',
        'month': 'Month',
        'is_member': ''
    },
    # Unstacking member and
    # casual rider bars
    barmode='group',
    # Formatting the count number output
    text_auto='.3s',
    # Setting the title
    title="Number of rides per month")
# Adjusting count number traces
fig.update_traces(textfont_size=13, textposition="outside")
# Adjusting the grid
fig.update_yaxes(gridwidth=2)
fig.show()
```



This bar chart shows a very apparent difference between members and casual riders in their monthly use of service. It should be noted that an increase in usage is observed from both parties starting January, peaking in August, and decreasing steadily each month through December. This relative consistency could suggest that members use it for regular transportation purposes. Casual riders for the most part stop using the service

during the cold months likely due to the drop in temperature, although members still do even at lower numbers than the rest of their months. Members are always, regardless of month, surpassing casual customers by a staggering difference. To quantify this gap in usage, let's look and compare each group's most active and least active month and then compare the results between the groups. For members, their least active month (March) is about 3.5 less than the amount of rides in their most popular month (August). For casual riders, their least active month (January) is 24.5 less than the amount of their most active month (August). The difference in this dropoff in use is around 7 times greater for casual riders than it is for members.

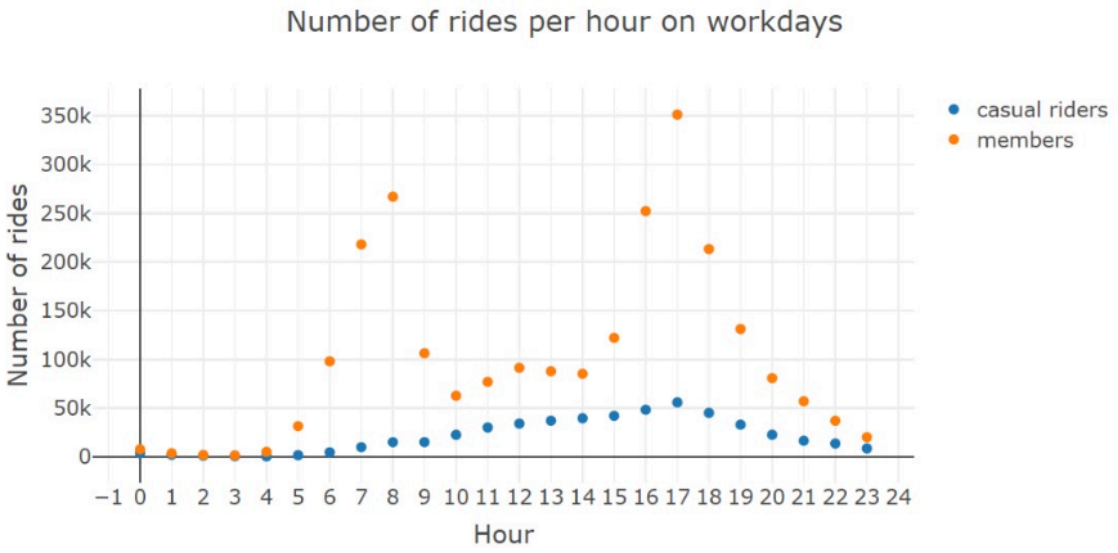
2. Is there any difference between members and casual riders in usage throughout the day?

In [50]:
```
# Creating a pivot table for visualization
df_pivot_hours_workdays = (
    pd.pivot_table(
        # Choosing only weekdays from 0 to 4 (Mon-Fri)
        df[df['weekday'] <= 4],
        # Grouping by hour and membership status
        index=['hour', 'is_member'],
        values=['ride_id', 'duration_min'],
        # Counting the number of rides
        # as well as finding the median duration
        aggfunc={
            'ride_id': 'count',
            'duration_min': 'median'
        }).reset_index()
    # Replacing boolean values for visualization
    # purposes
    .replace({
        False: 'casual riders',
        True: 'members'
    }).rename({'ride_id': 'count'}, axis=1))
# Validating the pivot table
df_pivot_hours_workdays.head(5)
```

Out[50]:

	hour	is_member	duration_min	count
0	0	casual riders	22	3955
1	0	members	9	7927
2	1	casual riders	23	2364
3	1	members	9	3969
4	2	casual riders	21	1324

In [315]:
```
# Plotting differences across hours
fig = px.scatter(
    df_pivot_hours_workdays,
    x='hour',
    y='count',
    color='is_member',
    # Including hover data with ride duration
    hover_data=['duration_min'],
    # Changing labels for clarity
    labels={
        'count': 'Number of rides',
        'hour': 'Hour',
        'is_member': '',
        'duration_min': 'Median duration'
    },
    # Unstacking member and
    # Casual rider bars
    #barmode='group',
    # Adjusting the title
    title='Number of rides per hour on workdays')
# Showing all of the x axis labels
fig.layout.xaxis.dtick = 1
# Adjusting the grid
fig.update_yaxes(gridwidth=2)
fig.show()
```

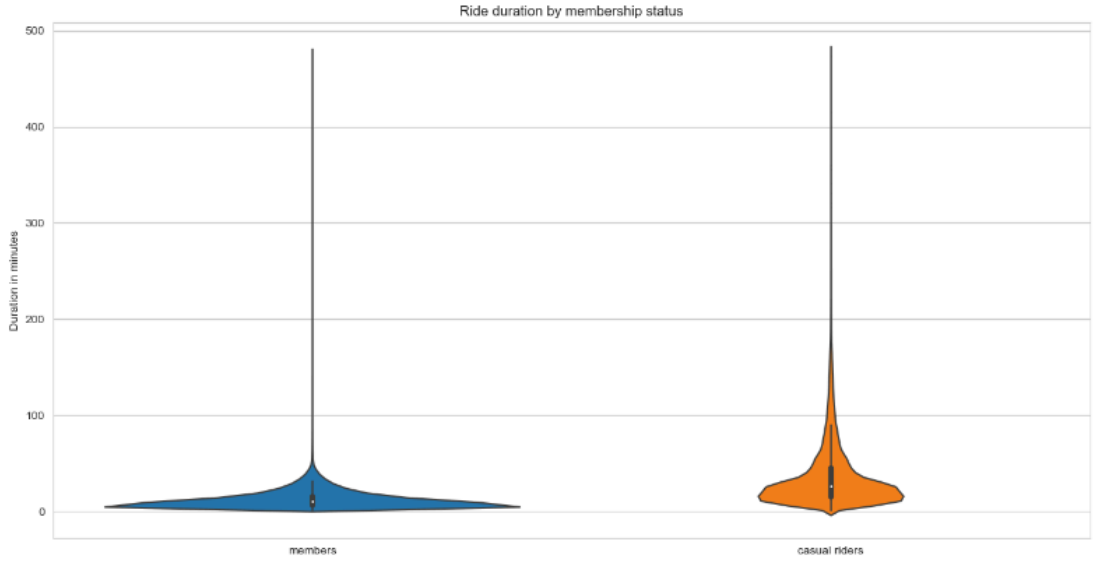


It is now crystal clear that member rides spike during commute times on workdays. For the period of time analyzed in our dataset, we see that there have been more than 350,000 member rides that started at 5:00 PM, as opposed to 90,000 rides that started around midday.

Casual customer rides are more evenly distributed throughout the day with the same peak at 5:00 PM, but evidently, much less of a total ride number.

3. What would the difference between member and casual ride durations in regards to their median and average duration?

```
In [54]: # plotting differences in ride durations
         # using seaborn and not plotly due to
         # plotly crashing when processing the data
         fig = sns.violinplot(data=df,
                              x=df['is_member']\
                              # replacing boolean values for visualization
                              # purposes
                              .replace({False:'casual riders',
                                        True:'members'}),
                              y='duration_min',
                              # standardizing the visual style
                              saturation=0.9)
         fig.set_title('Ride duration by membership status')
         fig.set_xlabel('')
         fig.set_ylabel('Duration in minutes');
```

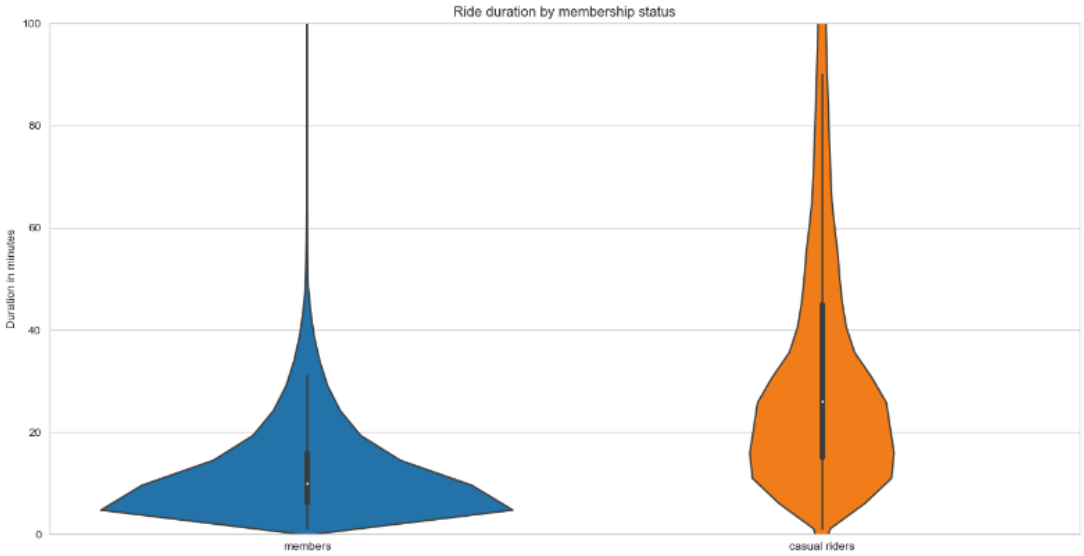


This plot is admittedly not very useful since it is being greatly affected by extreme values such as ride durations for over two hours. We will have to limit the y axis and while reprinting it, include average and median output values for members and casual customers.

In [55]:
```
# plotting differences in ride durations
fig = sns.violinplot(data=df,
                     x=df['is_member']\
                     # replacing boolean values for visualization
                     # purposes
                     .replace({False:'casual riders',
                               True:'members'}),
                     y='duration_min',
                     # standardizing the visual style
                     saturation=0.9)
fig.set_title('Ride duration by membership status')
fig.set_xlabel('')
fig.set_ylabel('Duration in minutes')
# limiting the y axis
plt.ylim(0, 100)

# embedding calculations for menas and medians inside f-strings
print(f'''The median and mean durations of rides for members are \
{round(df.loc[df['is_member'] == 1, 'duration_min'].median())}min and \
{round(df.loc[df['is_member'] == 1, 'duration_min'].mean())}min.
For non-members the statistics are \
{round(df.loc[df['is_member'] == 0, 'duration_min'].median())}min and \
{round(df.loc[df['is_member'] == 0, 'duration_min'].mean())}min,\
respectively.''')
```

```
The median and mean durations of rides for members are 10min and 13min.
For non-members the statistics are 26min and 37min,respectively.
```

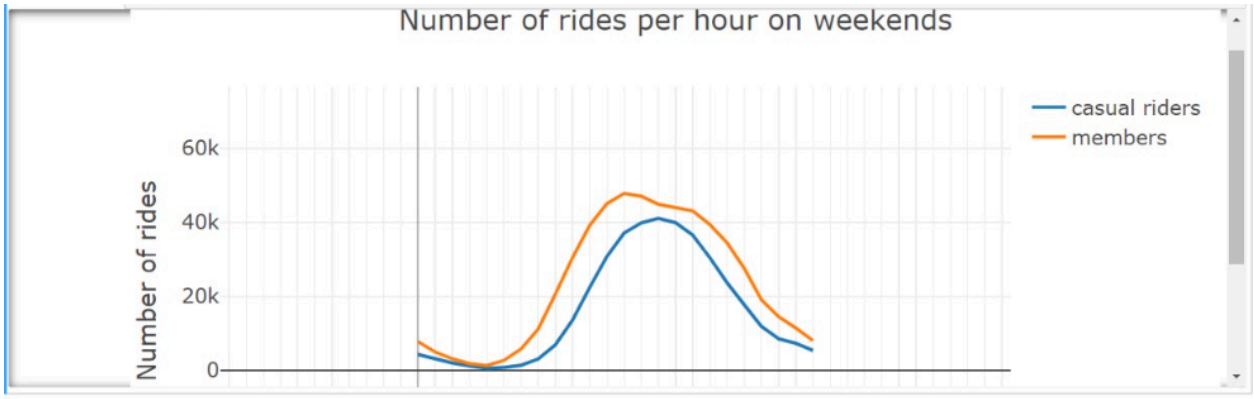


It seems that casual riders are more likely to engage in longer rides. Examining the box inside the left violin plot, more than 80% of member rides last for less than 20 minutes. A reasonable assumption would be that Cyclistic members use the service as an effective way to navigate from one point to another, when casual riders tend to make use of this service in a more recreational way and enjoy it as a leisure activity.

4. Is there any difference between members and casual riders in usage throughout the weekend?

In [1]:
```
# Creating a pivot table for visualization
df_pivot_hours_weekends = (
    pd.pivot_table(
        # Choosing only weekdays 5 and 6 (Sat, Sun)
        df[df['weekday'] >= 5],
        # Grouping by hour and membership status
        index=['hour', 'is_member'],
        values='ride_id',
        # Counting the number of rides
        aggfunc='count').reset_index()
    # Replacing boolean values for visualization
    # Purposes
    .replace({
        False: 'casual riders',
        True: 'members'
    }).rename({'ride_id': 'count'}, axis=1))
# Validating the pivot table
df_pivot_hours_weekends.head(5)
```

In [317]:
```
# Plotting differences across hours
fig = px.line(
    df_pivot_hours_weekends,
    x='hour',
    y='count',
    color='is_member',
    # Changing labels for clarity
    labels={
        'count': 'Number of rides',
        'hour': 'Hour',
        'is_member': ''
    },
    # Unstacking member and
    # casual rider bars
    #barmode='group',
    title="Number of rides per hour on weekends")
# Showing all of the x axis labels
fig.layout.xaxis.dtick = 1
# Adjusting the grid
fig.update_yaxes(gridwidth=2)
fig.show()
```



With this visualization, we had to filter out the weekdays to only include weekends. As we can see through our line chart, frequency distributions for casual riders and members are almost symmetrical on weekends with both categories of users tending to prefer daytime rides over nighttime rides.

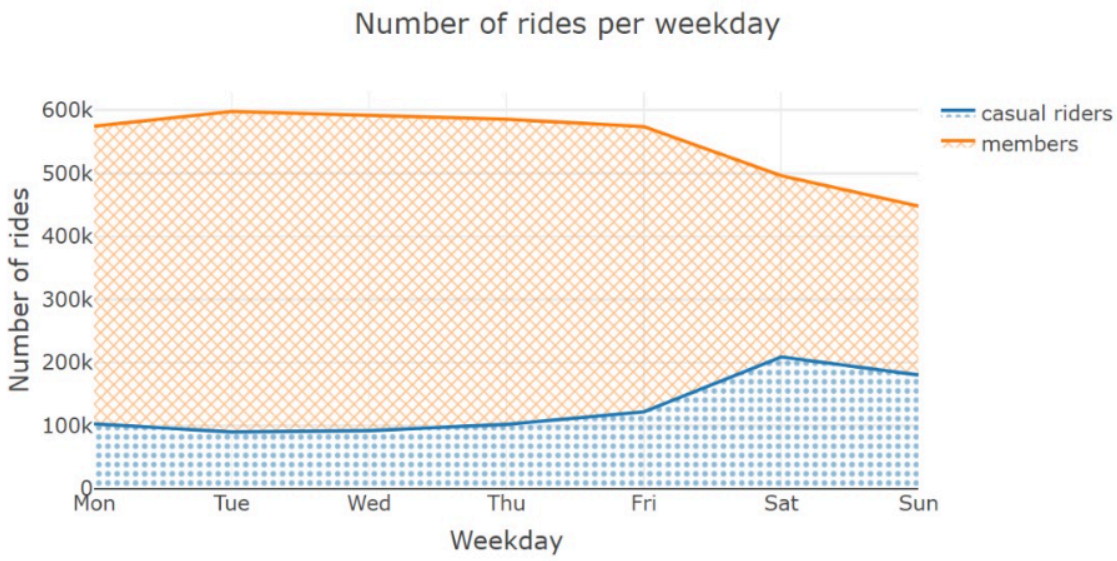
5. Would there be any difference in service usage between casual riders and members over weekdays?

In [312]:
```
# Creating a pivot table for visualization
df_pivot_weekdays = (
    pd.pivot_table(
        df,
        # Grouping by weekday and membership status
        index=['weekday', 'is_member'],
        values='ride_id',
        # Counting the number of rides
        aggfunc='count').reset_index()
    # Replacing boolean values for visualization
    # purposes
    .replace({
        False: 'casual riders',
        True: 'members'
    }).rename({'ride_id': 'count'}, axis=1))
df_pivot_weekdays = df_pivot_weekdays.replace({
    # Replacing integers with weekdays for better visualization
    0: 'Mon',
    1: 'Tue',
    2: 'Wed',
    3: 'Thu',
    4: 'Fri',
    5: 'Sat',
    6: 'Sun'
})
# Validating the pivot table
df_pivot_weekdays.head(5)
```

Out[312]:

	weekday	is_member	count
0	Mon	casual riders	102880
1	Mon	members	471519
2	Tue	casual riders	90037
3	Tue	members	507748
4	Wed	casual riders	91978

In [344]:
```
# Plotting differences across weekdays
fig = px.area(
    df_pivot_weekdays,
    x='weekday',
    y='count',
    color='is_member',
    pattern_shape="is_member",
    pattern_shape_sequence=[".", "x"],
    # Changing labels for clarity
    labels={
        'count': 'Number of rides',
        'weekday': 'Weekday',
        'is_member': ''
    },
    title="Number of rides per weekday")
# Adjusting the grid
fig.update_yaxes(gridwidth=2)
fig.show()
```



It seems that members tend to use Cyclistic a lot more on weekdays as opposed to casual riders who use the service more on weekends. This aligns with our initial hypothesis that since the service is primarily used for work commute by its most loyal group (members), it is normal to witness a decline in the service use over weekends. Most of the population would not work over weekends which makes it expected to see the decline.

Conclusion/Recommendations

Our analysis concludes that members tend to use Cyclistic as a means of transportation and commuting, while casual riders use the service as a leisurely activity. One recommendation for

the marketing team would be emphasizing the convenience of commuting to and from work using a bike. They can work closely with Cyclistic's strategy development team to plan campaigns that underline some crucial overlooked benefits of cycling such as weight loss and lessening our carbon footprint. It is essential to keep in mind the timing of this marketing campaign. Based on our analysis, it is shown that during late spring and early summer when casual rides spike and cycling enthusiasm is on the rise, it would be the perfect time to launch this campaign and maximize profits by converting as many casual riders into members as possible.

References:

1. Elliot Fishman (2016) Bikeshare: A Review of Recent Literature, Transport Reviews, 36:1, 92-113, DOI: 10.1080/01441647.2015.103303
2. Barbeau, S. J., Georggi, N. L., & Winters, P. L. (2010). Global Positioning System Integrated with Personalized Real-Time Transit Information from Automatic Vehicle Location. *Transportation Research Record*, 2143(1), 168–176. https://doi.org/10.3141/2143-21
3. Fishman, E., Washington, S., & Haworth, N. (2013). Bike share: A synthesis of the literature. Transport Reviews, 33(2), 148-165. https://doi.org/10.1080/01441647.2013.775612