Bike sharing membership analysis

Oscar Quiroga

2024

Introduction

In the company there are 2 categories of memberships for each client, an annualized subscription for member riders and a daily subscription or one ride payment for casual riders. The finance analyst determined that full members are more profitable for the company then the recommended step is to develop an effective marketing program targeting casual riders that could become members thus it is necessary a comparison of how casual riders and members use the bikes differently.

The comparison is going to use data of rides for the past 12 months, and beyond, of members and casual rides and the analysis done with MySQL and R with compelling graphics with Tableu of metrics in the data.

The data was sourced from here, The data has been made available by Motivate International Inc. under this license

Prepare the data: Load the data into a SQL database

The Data comes in a series of zip files containing comma-separated value (csv) files, these files are uploaded to a MySQL database with a python script that loads the data with SQL queries. A SQL dataset was chosen because it permanently stores the data saving time not loading the data every time manipulation of data is done, also it can clean and process huge amounts of data which is the case with this database.

The SQL database is created and checks if there's a database with the same name:

```
DROP DATABASE IF EXISTS project_rides;
CREATE DATABASE project_rides;
USE project_rides;
```

Then create the tables for `project_rides`, `rides_recent` for entries starting from 2020 and `rides_old` for rides from 2019 and before, the code to create the tables was sourced from MySQL-workbench import wizard:

```
CREATE TABLE `rides_recent` (
                `ride_id` text,
                `rideable_type` text,
                `started_at` datetime DEFAULT NULL,
                `ended_at` datetime DEFAULT NULL,
                `start_station_name` text,
                `start_station_id` text,
                `end_station_name` text,
                `end_station_id` text,
                `start_lat` double DEFAULT NULL,
                `start_lng` double DEFAULT NULL,
                `end_lat` double DEFAULT NULL,
                `end_lng` double DEFAULT NULL,
                `member_casual` text
                ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

```
CREATE TABLE `rides_old` (
                `trip_id` text,
                `start_time` datetime DEFAULT NULL,
                `end_time` datetime DEFAULT NULL,
                `bikeid` text,
                `tripduration` text,
                `from_station_id` text,
                `from_station_name` text,
                `to_station_id` text,
                `to_station_name` text,
                `usertype` text,
                `gender` text,
                `birthyear` int DEFAULT NULL
                ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

A python script is used to create a SQL script for loading each csv to the correct table, first load the necessary libraries, and define a function to list all the file paths in a folder with a specific file extension:

```
from glob import glob
import os
import csv
from tqdm import tqdm
from datetime import datetime

def get_files_from_path(folder:str, recursive:bool = True, extensions:list =None) -> list:
    if recursive:
        all_files = glob(os.path.join(folder,'**','*'), recursive=True)
    else:
        all_files = glob(os.path.join(folder,'*'), recursive=False)
    if extensions is None:
        files = all_files
    else:
        files = []
        for fil in all_files:
            fil = str(fil)
            if os.path.splitext(fil)[1] in extensions:
                files.append(fil)
    return files
```

create a list with all the lines of the SQL queries and get all the filepaths with csv extension

```
if __name__ == "__main__":
    lines = []
    csv_paths = get_files_from_path('./data', extensions=['.csv'])
```

start iteration, and load the first cell of the csv and evaluate, if it's `ride_id` then are recent rides, if it's `trip_id` or `01 - Rental Details Rental ID` then are old rides, if it's neither of those then ignore the csv:

```
    for path in tqdm(csv_paths):
        with open(path, 'r') as f:
            col_names = f.readline()
        first_cell = col_names.split(',')[0]
        if first_cell[0] == '"' and first_cell[-1] == '"':
            first_cell = first_cell[1:-1]
        if first_cell == 'ride_id':
```

```
            table_name = 'rides_recent'
        elif any([first_cell == x for x in
                  ('trip_id', '01 - Rental Details Rental ID')]):
            table_name = 'rides_old'
        else:
            print(path)
            print('Unrecognized format')
            continue
```

Place `\N` in null spaces in the raw data because that's how MySQL can interpret NULL values, otherwise MySQL uses bad defaults, also make sure all the datetimes have the format `%Y-%m-%d %H:%M:%S`

```
        with open(path) as f:
            readed = list(csv.reader(f))
        for i, entry in enumerate(readed):
            if i == 0:
                continue
            for j, cell in enumerate(entry):
                if cell == '':
                    readed[i][j] = '\\N'
            if table_name == 'rides_old':
                si = 1
                ei = 2
            else:
                si = 2
                ei = 3
            ok_date = False
            for date_format in ['%Y-%m-%d %H:%M:%S', '%m/%d/%Y %H:%M',
                                '%m/%d/%Y %H:%M:%S', '%Y-%m-%d %H:%M']:
                try:
                    datestart = str(datetime.strptime(entry[si], date_format))
                    ok_date = True
                    if date_format != '%Y-%m-%d %H:%M:%S':
                        readed[i][si] = datestart
                        readed[i][ei] = str(datetime.strptime(entry[ei], date_format))
                except:
                    pass
```

Create the SQL query to load the csv, `path` being the csv path and `table_name` the table to insert the rows of data, commas separating the values, removing double quotes if the values are enclosed by double quotes, and each entry separated by a newline character `\r\n` and ignore the first row that has the column names, https://dev.mysql.com/doc/refman/8.0/en/load-data.html, the query is then added to the list of queries:

```
        path = os.path.abspath(path)
        lines += ["LOAD DATA LOCAL INFILE '{}' \n\
                    INTO TABLE {} \n\
                    FIELDS TERMINATED BY ',' \n\
                    ENCLOSED BY '\"' \n\
                    LINES TERMINATED BY '\\r\\n' \n\
                    IGNORE 1 ROWS\
                ".format(path, table_name)]
```

finish the iteration and join all the SQL queries, then write a file to be used in MySQL, copy the contents of `load_csv.sql` file into MySQL-workbench and execute all the queries there.

```
    sql_script = ';\n'.join(lines)
    with open('load_csv.sql', 'w') as w:
        w.write(sql_script)
    print('SQL script ready')
```

One of generated queries as an example:

```
LOAD DATA LOCAL INFILE 'data/Divvy_Trips_2020_Q1.csv'
                    INTO TABLE rides_recent
                    FIELDS TERMINATED BY ','
                    ENCLOSED BY '"'
                    LINES TERMINATED BY '\r\n'
                    IGNORE 1 ROWS;
```

import the most recent station data from `Divvy_Stations_2017_Q3Q4.csv` into a new table named `stations_old`

```
CREATE TABLE `stations_old` (
                `id` text,
                `name` text,
                `city` text,
                `latitude` double DEFAULT NULL,
                `longitude` double DEFAULT NULL,
                `dpcapacity` text,
                `online_date` text
                ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

LOAD DATA LOCAL INFILE 'data/Divvy_Stations_2017_Q3Q4.csv'
                    INTO TABLE `stations_old`
                    FIELDS TERMINATED BY ','
                    ENCLOSED BY '"'
                    LINES TERMINATED BY '\r\n'
                    IGNORE 1 ROWS;
```

create a new table averaging the longitude and latitude of distinct stations, only the start stations where used because some rides didn't have stations, these are filtered out, also empty locations.

```
CREATE TABLE stations_recent AS
    SELECT
        start_station_name AS station_name,
    AVG(start_lat) AS lat,
    AVG(start_lng) AS lng
    FROM
        rides_recent
  WHERE
        start_station_name IS NOT NULL AND
    start_lat IS NOT NULL AND
    start_lng IS NOT NULL
    GROUP BY
        station_name;
```

Process the data: Clean data and merge old data with recent data

get the stations with longitude and latitude that are not in the recent stations, then insert these stations into a new table with recent stations, this with the purpose of joining with rides without latitude and longitude values

```
USE project_rides;
DROP TABLE IF EXISTS only_old_stations;
CREATE TEMPORARY TABLE only_old_stations
    SELECT
        name AS station_name,
    latitude AS lat,
    longitude AS lng
    FROM
        stations_old
    WHERE
        name NOT IN (SELECT
            station_name
        FROM
            stations_recent);
DROP TABLE IF EXISTS stations;
CREATE TABLE stations AS
    SELECT *
    FROM stations_recent;

INSERT INTO stations
    SELECT *
    FROM only_old_stations;
DROP TABLE IF EXISTS only_old_stations;
```

now that all stations have coordinates is time to join these coordinates with the stations of old rides that lack coordinates data and create a new table `rides_old_coord`

```
CREATE TABLE rides_old_coord AS
SELECT * FROM rides_old
    LEFT JOIN
        (
            SELECT
                station_name AS from_station_name,
                lat AS start_lat,
                lng AS start_lng
            FROM
                stations
        ) AS stations_start USING (from_station_name)
    LEFT JOIN
        (
            SELECT
                station_name AS to_station_name,
                lat AS end_lat,
                lng AS end_lng
            FROM
                stations
        ) AS stations_end USING (to_station_name)
```

Duplicate `rides_recent` table and insert the entries from `rides_old_coord` with the correct values and column names

```
CREATE TABLE rides AS
    SELECT *
    FROM rides_recent;
INSERT INTO rides
```

```
    SELECT
        CONCAT('OLD', trip_id) AS ride_id,
        NULL AS rideable_type,
        start_time AS started_at,
            end_time AS ended_at,
        from_station_name AS start_station_name,
        from_station_id AS start_station_id,
        to_station_name AS end_station_name,
        to_station_id AS end_station_id,
        start_lat,
        start_lng,
        end_lat,
        end_lng,
        CASE usertype
            WHEN "Subscriber" THEN "member"
            WHEN "Customer" THEN "casual"
            ELSE "casual"
        END AS member_casual
    FROM
        rides_old_coord;
```

Export the csv of each year, first get the column names by the table order, then export to the data folder of the MySQL server filtering out the entries where the start date is later than the end date, for this project the analysis will be made to the year 2023 only, but a more throughout analysis could be done starting from 2013

```
USE project_rides;
SELECT group_concat(CONCAT("'", COLUMN_NAME , "'") ORDER BY ORDINAL_POSITION)
  FROM INFORMATION_SCHEMA.COLUMNS
  WHERE TABLE_NAME = 'rides'; ---copy the columns' names and paste below
SELECT *
FROM (
        SELECT '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'
        UNION ALL
        SELECT * FROM rides
        ) AS dat
WHERE
    ride_id = 'ride_id' OR ---keeps column headers
  (UNIX_TIMESTAMP(ended_at) - UNIX_TIMESTAMP(started_at)) > 0
    AND YEAR(started_at) = 2023
INTO OUTFILE '/var/lib/mysql/full_data.csv'
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n';
```

Analyze data: metrics and visualizations

Download all the necessary libraries

```
install.packages("tidyverse")
install.packages("tidyr")
install.packages("rlist")
```

```
install.packages("reshape2")
install.packages("dplyr")
install.packages("tibble")
install.packages("lubridate")
install.packages("ggplot2")
install.packages("scales")
install.packages("knitr")
```

and load the libraries. Includes libraries to manipulate data like `dplyr`, `reshape2` and load data like `tidyr`, also for creating visualizations like `ggplot2`

```
library("tidyverse")
library("tidyr")
library("rlist")
library("reshape2")
library("dplyr")
library("tibble")
library("lubridate")
library("ggplot2")
library("scales")
library("knitr")
```

Define `types_memberships` with the types of memberships later used for iterations, also the path where the csv with the rides are located and load the database with the `read_csv`, and `\N` as null cells.

```
types_memberships <- c('casual', 'member')
csv_path <- "./full_data.csv"
rides_df <- read_csv(csv_path, na = c("", "NA", "\\N"))
```

Preview the data frame so all the data types are correct, specially the ones related to dates `started_at` and `ended_at` should be data frame column variables `<dttm>` and latitudes and longitudes should be continuous data types `<dbl>`. The glimpse should display around 5 million entries and 13 variables for this dataset.

```
glimpse(rides_df)
```

```
## Rows: 5,718,608
## Columns: 13
## $ ride_id            <chr> "C9BD54F578F57246", "CDBD92F067FA620E", "ABC0858E52~
## $ rideable_type      <chr> "electric_bike", "electric_bike", "electric_bike", ~
## $ started_at         <dttm> 2023-12-02 18:44:01, 2023-12-02 18:48:19, 2023-12-~
## $ ended_at           <dttm> 2023-12-02 18:47:51, 2023-12-02 18:54:48, 2023-12-~
## $ start_station_name <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
## $ start_station_id   <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
## $ end_station_name   <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
## $ end_station_id     <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
## $ start_lat          <dbl> 41.92, 41.92, 41.89, 41.95, 41.92, 41.91, 41.99, 42~
## $ start_lng          <dbl> -87.66, -87.66, -87.62, -87.65, -87.64, -87.63, -87~
## $ end_lat            <dbl> 41.92, 41.89, 41.90, 41.94, 41.93, 41.88, 42.00, 41~
## $ end_lng            <dbl> -87.66, -87.64, -87.64, -87.65, -87.64, -87.65, -87~
## $ member_casual      <chr> "member", "member", "member", "member", "member", "~
```

The number of rides and the percentage for each type of membership is analysed for differences in the year, the `count` function groups and counts the entries in months and the type of membership then with `dcast` the data is converted from long format to wide format and a new calculated field `total` sums the memberships to create the percentages of each membership

```r
month_rides_tv <- rides_df %>%
  count(month = month(started_at), member_casual) %>% #summarizes, no need to summarize later
  dcast(month ~ member_casual, value.var = "n") %>%
  mutate(total = member + casual) %>%
  mutate(casual_percent=100*casual/total, member_percent= 100*member/total)
```

A simple inspection of the table shows casual customers are less likely than member customers to use the service at winter months

```r
kable(month_rides_tv)
```

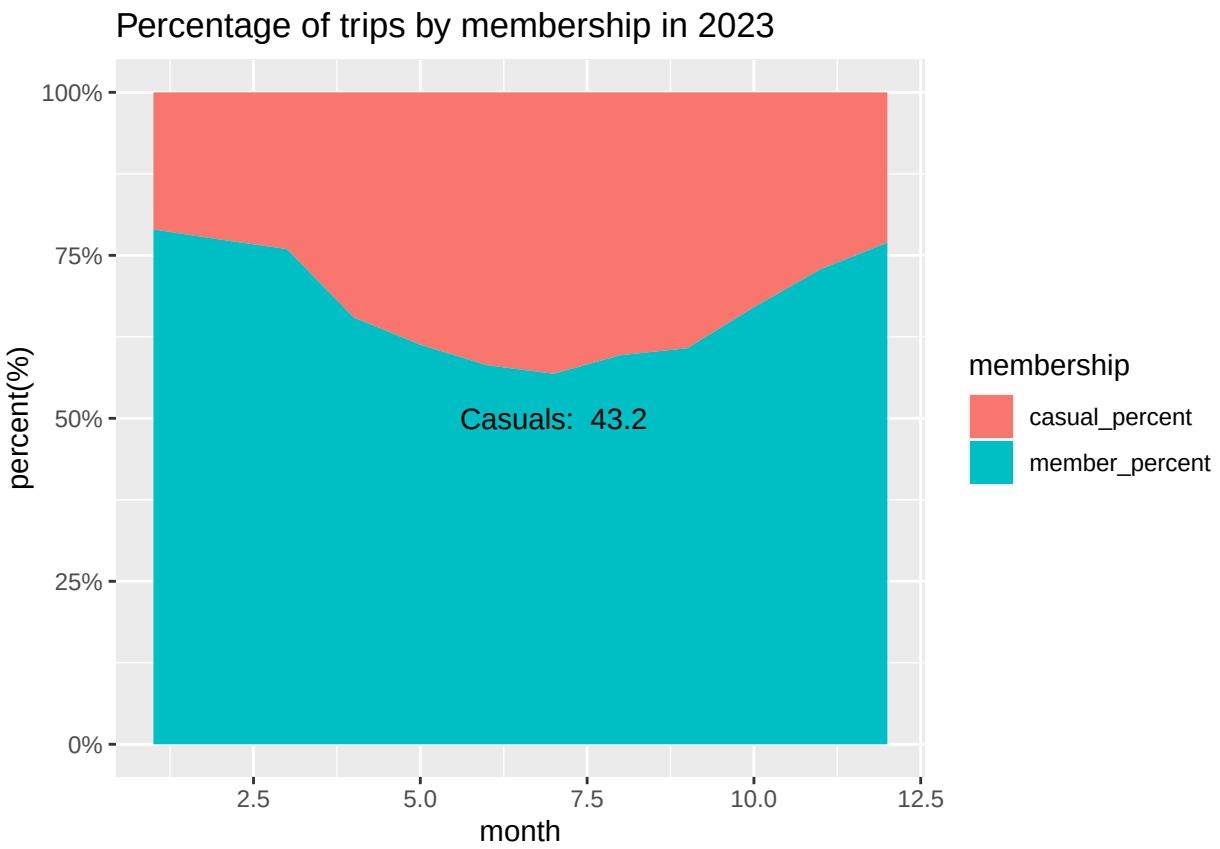
month	casual	member	total	casual_percent	member_percent
1	40005	150288	190293	21.02284	78.97716
2	43014	147422	190436	22.58712	77.41288
3	62194	196465	258659	24.04478	75.95522
4	147269	279285	426554	34.52529	65.47471
5	234153	370603	604756	38.71859	61.28141
6	301198	418347	719545	41.85951	58.14049
7	331252	436185	767437	43.16341	56.83659
8	311006	460430	771436	40.31520	59.68480
9	261534	404617	666151	39.26047	60.73953
10	177007	359947	536954	32.96502	67.03498
11	98327	264042	362369	27.13450	72.86550
12	51662	172356	224018	23.06154	76.93846

So a visualization of the number of rides of each membership can visually confirm this trend, the data is converted back to long format with `melt`, apparently both memberships increase in summer and decrease in winter.

```r
ggplot(month_rides_tv %>%
         select(month, casual, member)  %>%
         melt(id.vars = "month", variable.name = "membership")
       )+
  labs(title="Number of trips by membership in 2023") +
  ylab("Trips") +
  geom_line(mapping =aes(x= month, y = value, color = membership)) +
  scale_y_continuous(labels = scales::comma) +
  scale_x_continuous(breaks=seq(0, 12, 2)) +
  expand_limits(x=1, y=0)
```

Another visualization with percentage is used, and this shows that although the number of rides increase and decrease equally the proportion is different and casual customers make up 43.2% at their peak in July(7)

```
max_p_casuals <- as.character(
  round(max(month_rides_tv$casual_percent), 1)
)
ggplot(month_rides_tv %>%
         select(month, casual_percent, member_percent)  %>%
         melt(id.vars = "month", variable.name = "membership")
       ) +
  labs(title="Percentage of trips by membership in 2023") +
  ylab("percent(%)") +
  geom_area(position = position_fill(), mapping = aes(x= month, y = value, fill = membership)) +
  scale_y_continuous(labels = scales::percent) +
  annotate("text", x = 7, y = 0.5, label=paste("Casuals: ", max_p_casuals) )
```



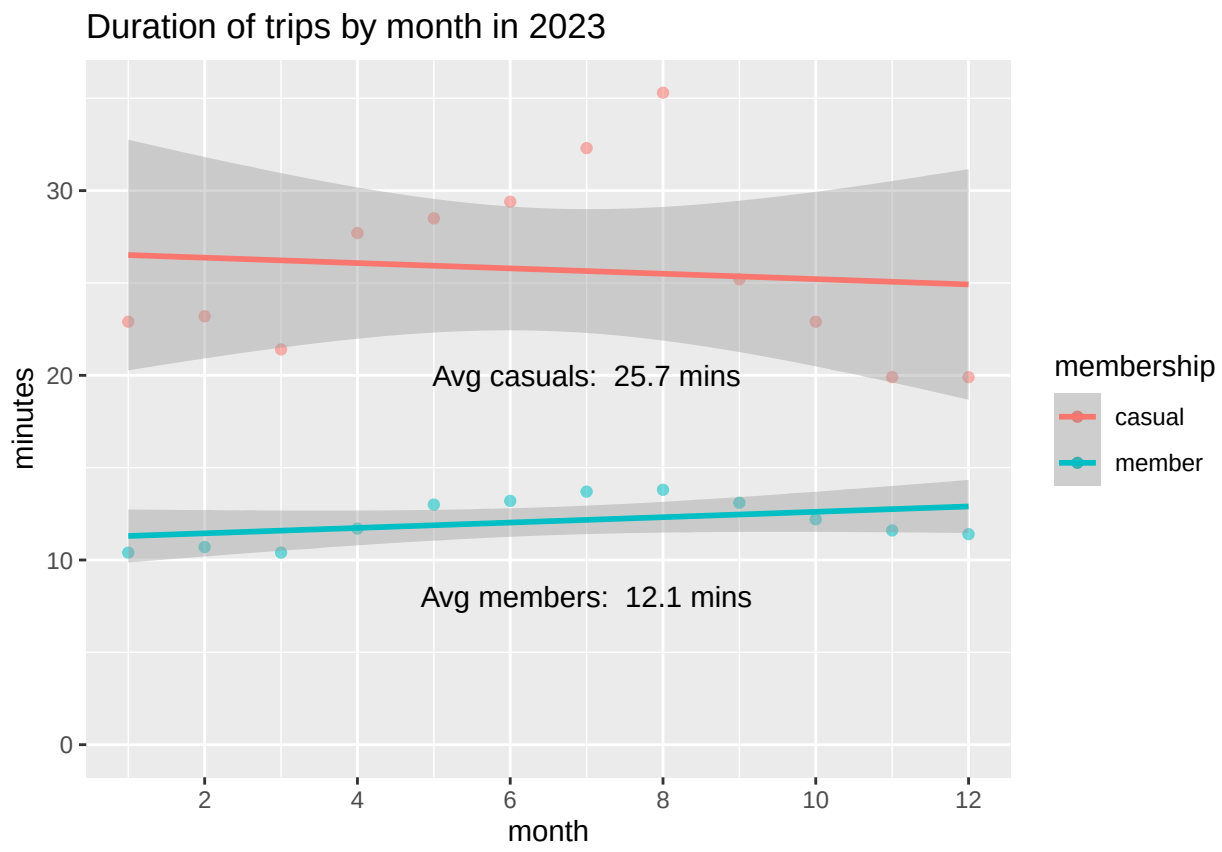
The duration of each ride was compared, the duration was taken as the time taken from start to finish, and on average casual customers take longer, almost double, in each ride, spiking in summer months and dipping in winter months.

```
duration_rides_m <- rides_df %>%
  mutate(duration = difftime(ended_at, started_at, units="mins")) %>%
  mutate(month = month(started_at)) %>%
  select(member_casual, month, duration) %>%
  group_by(member_casual, month) %>%
  summarize(avg_duration = round(mean(duration), digits= 1) ) %>%
  arrange(month, member_casual)

avg_duration_casual <- as.character( round( (duration_rides_m %>%
  filter(member_casual== "casual") %>%
  summarize(result = mean(avg_duration))) $ result, 1) )
avg_duration_member <- as.character( round( (duration_rides_m %>%
  filter(member_casual== "member") %>%
  summarize(result = mean(avg_duration))) $ result, 1) )

ggplot(duration_rides_m)+
  labs(title="Duration of trips by month in 2023") +
  ylab("minutes") +
  geom_point(mapping = aes(x= month, y = avg_duration, color = member_casual, alpha=0.5)) +
  guides(alpha = "none", color = guide_legend(title="membership")) +
  annotate("text", x = 7, y = 20, label=paste("Avg casuals: ", avg_duration_casual, "mins") ) +
  annotate("text", x = 7, y = 8, label=paste("Avg members: ", avg_duration_member, "mins") ) +
  scale_x_continuous(breaks=seq(0, 12, 2)) +
```

```
  geom_smooth(method = "lm", mapping = aes(x= month, y = avg_duration, color = member_casual)) +
  expand_limits(x=1, y=0)
```



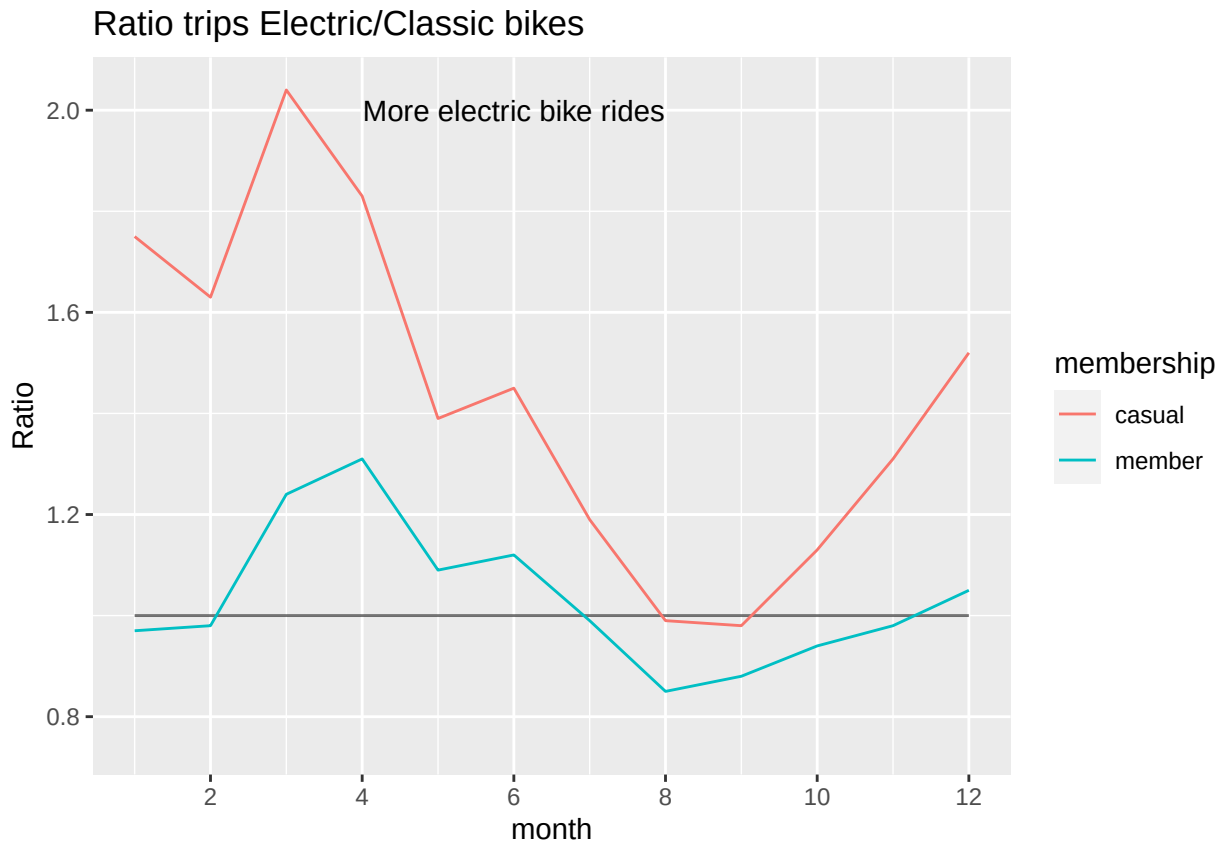
The type of bikes each customer uses is compared, docked bikes are ignored because these only appear with casual customers and the number of rides were small compared to classic and electric bikes, like in the analysis of number of rides the data is converted to wide format to check for trends, the ratio of electric and classic bikes was chosen to be visualized, and it was found that casual costumers use electrical bikes more often overall, it spikes in March(3) maybe because these are more available when demand is low.

```
rtype_ratio <- rides_df %>%
  filter(rideable_type != "docked_bike") %>%
  count(month = month(started_at), member_casual, rideable_type) %>%
  dcast(month + member_casual ~ rideable_type , value.var = "n") %>%
  mutate(ratio = round(electric_bike/classic_bike, 2)) %>%
  select(month, member_casual, ratio) %>%
  dcast(month ~ member_casual , value.var = "ratio")

ggplot(rtype_ratio %>%
         melt(id.vars = "month", variable.name = "membership")
       ) +
  labs(title="Ratio trips Electric/Classic bikes") +
  ylab("Ratio") +
  geom_line(mapping = aes(x= month, y = 1.0, alpha= 0.33)) +
  guides(alpha = FALSE) +
  geom_line(mapping = aes(x= month, y = value, color = membership)) +
  annotate("text", x= 6, y = 2, label="More electric bike rides") +
```

```
  scale_x_continuous(breaks=seq(0, 12, 2)) +
  expand_limits(y=0.75)
```

```
## Warning: The ‘<scale>‘ argument of ‘guides()‘ cannot be ‘FALSE‘. Use "none" instead as
## of ggplot2 3.3.4.
## This warning is displayed once every 8 hours.
## Call ‘lifecycle::last_lifecycle_warnings()‘ to see where this warning was
## generated.
```

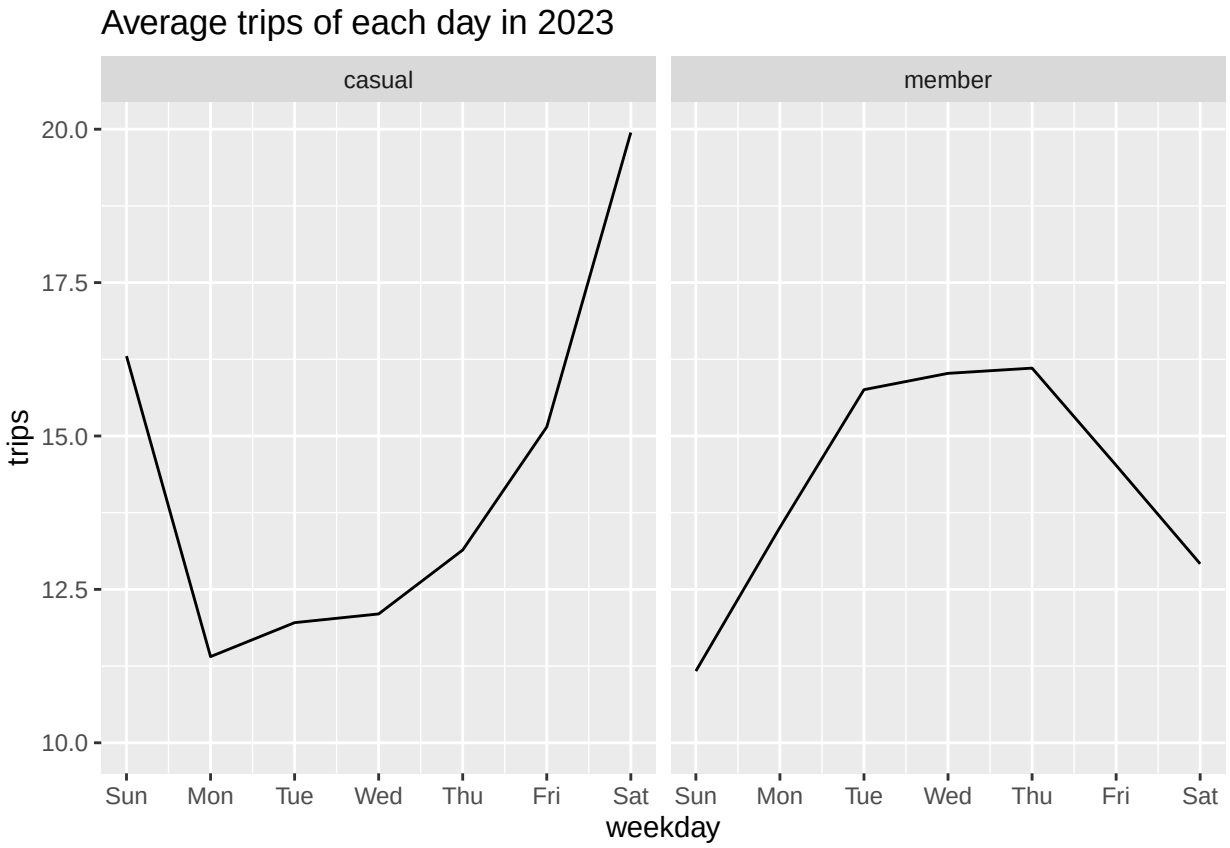


The weekdays each customer ride bikes were compared and each type of customer is completely opposite to the other, because casual customers ride more often on Friday, Saturday and Sunday, while member customers ride more often on Tuesday, Wednesday and Thursday.

```
hours_rides <- rides_df %>%
  count(weekday = wday(started_at), member_casual) %>% #summarizes, no need to summarize later
  dcast(weekday ~ member_casual, value.var = "n") %>%
  mutate(casual=100*casual/sum(casual), member= 100*member/sum(member))

ggplot(hours_rides %>%
         melt(id.vars = "weekday", variable.name = "membership")
       ) +
  labs(title="Average trips of each day in 2023") +
  ylab("trips") +
  geom_line(mapping = aes(x= weekday, y = value)) +
  facet_wrap(~membership)+
  scale_x_continuous(breaks = 1:7,
                     labels = c("Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat")
```

```
                    ) +
  expand_limits(y=10)
```



The stations most used were compared, for casual customers the most used stations are Streeter Dr & Grand Ave and DuSable Lake Shore Dr & Monroe St, and for member customer are Clinton St & Washington Blvd and Kingsbury St & Kinzie St

```
station_rides <- rides_df %>%
  drop_na(start_station_name) %>%
  count(start_station_name, member_casual)

most_used <- list()
for (member_type in types_memberships){
  most_used_member <- station_rides %>%
    filter(member_casual == member_type) %>%
    arrange(desc(n)) %>%
    head(10) %>%
    select(start_station_name, n)
  most_used <- list.append(most_used, most_used_member)
}
names(most_used) <- types_memberships
kable(data.frame(most_used))
```

casual.start_station_name	casual.n	member.start_station_name	member.n
Streeter Dr & Grand Ave	46019	Clinton St & Washington Blvd	26207
DuSable Lake Shore Dr & Monroe St	30482	Kingsbury St & Kinzie St	26168

casual.start_station_name	casual.n	member.start_station_name	member.n
Michigan Ave & Oak St	22661	Clark St & Elm St	24996
DuSable Lake Shore Dr & North Blvd	20337	Wells St & Concord Ln	21417
Millennium Park	20219	Clinton St & Madison St	20591
Shedd Aquarium	17777	Wells St & Elm St	20394
Theater on the Lake	16355	University Ave & 57th St	20037
Dusable Harbor	15487	Broadway & Barry Ave	18955
Wells St & Concord Ln	12168	Loomis St & Lexington St	18898
Montrose Harbor	11986	State St & Chicago Ave	18484

And a map with the top 250 stations used show casual customers take rides bordering the shoreline, while members concentrate inwards, live visualization

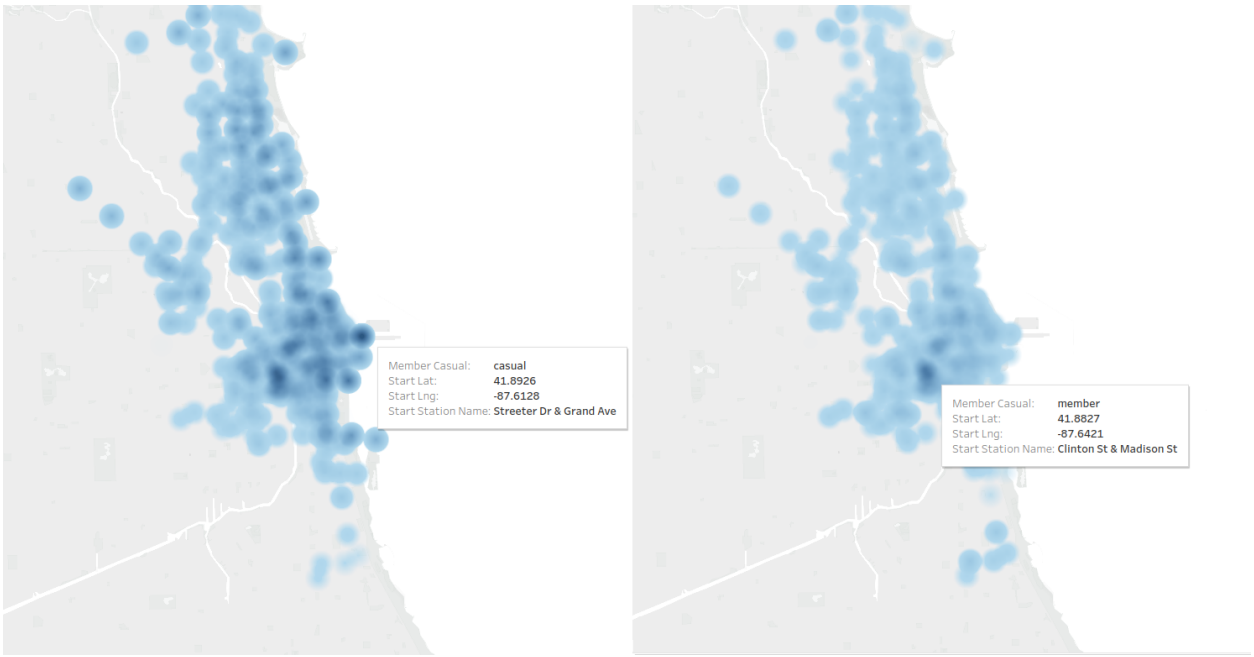


Figure 1: Density map of stations

Act: conclusions and recommendations

After analysis of the data, the recommendations and conclusions for targeting the casual customers:

- Member and casual customers use the service more in summer months, but specially casual customers that make up 43.2% of all trips in the peak month of July(7), it's better to do the capaign in these months.
- Casual customers take much longer in their trips doubling member customers, offering incentives to become a member based in the duration of the trip could make trips even longer reducing the availability of the service with less bikes in stations.
- Casual members use the service most often Friday, Saturday and Sunday, the campaign could be more effective these days and also offer a discount for members these days, actual members ride to a lesser degree these days.
- Casual rides concentrate around the shoreline and specially in Streeter Dr & Grand Ave and DuSable Lake Shore Dr & Monroe St stations, the campaign has to be more prominent in these stations.

- Casual customers ride electrical bikes more often than classical bikes, in months with high demand not so much maybe because of not enough electrical bikes at stations.