# Cyclistic Case Study Report

By David-Praise Ebiringa

2023-08-03

This is a case study that I completed using RStudio as part of the Google Data Analytics Professional Certificate Course. Kindly follow along each step of the data analysis process (**Ask**, **Prepare**, **Process**, **Analyse**, **Share**, and **Act**) to unpack the rest of my analysis.

## Background

You are a junior data analyst working in the marketing analyst team at Cyclistic, a bike-share company in Chicago.

The director of marketing believes the company's future success depends on maximizing the number of annual memberships.

Therefore, your team wants to understand how casual riders and annual members use Cyclistic bikes differently.

From these insights, your team will design a new marketing strategy to convert casual riders into annual members. But first, Cyclistic executives must approve your recommendations, so they must be backed up with compelling data insights and professional data visualization.

### About the company

In 2016, Cyclistic launched a successful bike-share offering. Since then, the program has grown to a fleet of 5,824 bicycles that are geotracked and locked into a network of 692 stations across Chicago. The bikes can be unlocked from one station and returned to any other station in the system anytime.

Until now, Cyclistic's marketing strategy relied on building general awareness and appealing to broad consumer segments. One approach that helped make these things possible was the flexibility of its pricing plans: single-ride passes, full-day passes, and annual memberships. Customers who purchase single-ride or full-day passes are referred to as casual riders. Customers who purchase annual memberships are Cyclistic members.

Cyclistic's finance analysts have concluded that annual members are much more profitable than casual riders. Although the pricing flexibility helps Cyclistic attract more customers, Moreno believes that maximizing the number of annual members will be key to future growth. Rather than creating a marketing campaign that targets all-new customers, Moreno believes there is a very good chance to convert casual riders into members. She notes that casual riders are already aware of the Cyclistic program and have chosen Cyclistic for their mobility needs.

Moreno has set a clear goal: Design marketing strategies aimed at converting casual riders into annual members. In order to do that, however, the marketing analyst team needs to better understand how annual members and casual riders differ, why casual riders would buy a membership, and how digital media could affect their marketing tactics. Moreno and her team are interested in analyzing the Cyclistic historical bike trip data to identify trends.

## Phase 1: Ask

**Business Task**: Cyclistic's current marketing strategy rooted in the flexibility of the price of its plans approach has contributed to its growing customer base. However, this approach is not financially sustainable as members are much more profitable to the company than casual riders. There is a need to adopt a new marketing approach using digital media that would convert existing casual riders to members.

**Key Question**:

- How do annual members and casual riders use bikes differently?
- In what other ways do they differ?

**Assumptions**:

- Usage differences are based on where casual riders go with the bikes vs. where members go.
- Another difference could be that casual riders typically prefer a certain kind of bike compared to members.

**Key stakeholders**:

- Lily Moreno – Director of Marketing
- Cyclistic executive team
- Cyclistic marketing analytics team

## Phase 2: Prepare

### Data Source

Cyclistic's historical trip data is based on the last 12 months upon the start of this project (i.e June 2022 - May 2023) The data set can be found here.

This data set limits the options for data analysis because it does not uniquely identify each rider apart from the "ride id" number category. There is no information about the age, gender, or sex of the riders. Thus, differences in ride usage can only be analyzed based on the type of customer, or by conducting descriptive analysis on ride categories.

## Phase 3: Process

The data was imported and cleaned in RStudio. R was chosen for this analysis because the data set is too large. In addition, R is functionally sound for cleaning data sets like these, making calculations, creating data visualizations as well as documenting and sharing the results of the analysis with R markdown. It was a no brainer! Below are the steps used within this phase:

### (a). Importing Data into R

The first step is to load the necessary packages before uploading each of the 12 .csv files into R.

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

```
## Installing package into '/cloud/lib/x86_64-pc-linux-gnu-library/4.3'
## (as 'lib' is unspecified)
```

```
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.2     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v ggplot2   3.4.2     v tibble    3.2.1
## v lubridate 1.9.2     v tidyr     1.3.0
## v purrr     1.0.1
```

```
## -- Conflicts --------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become error
```

```
library(readr) #for importing data into R with read_csv function
library(dplyr) #for data manipulation tasks
library(tidyr) #for data cleaning
library(lubridate) #Useful for date functions
library(ggplot2) #for data visualizations
library(hms) #for extracting hours from datetime
```

```
##
## Attaching package: 'hms'
##
## The following object is masked from 'package:lubridate':
##
##     hms
```

```
#Note: packages on lines 4-98are embedded in the tidyverse package already

install.packages("geosphere") #to calculate the distance between two geo points
```

```
## Installing package into '/cloud/lib/x86_64-pc-linux-gnu-library/4.3'
## (as 'lib' is unspecified)
```

```
library(geosphere)
```

```
## The legacy packages maptools, rgdal, and rgeos, underpinning the sp package,
## which was just loaded, will retire in October 2023.
## Please refer to R-spatial evolution reports for details, especially
## https://r-spatial.org/r/2023/05/15/evolution4.html.
## It may be desirable to make the sf package available;
## package maintainers should consider adding sf to Suggests:.
## The sp package is now running under evolution status 2
##      (status 2 uses the sf package in place of rgdal)
```

```
install.packages("skimr")#for summary statistics on data
```

```
## Installing package into '/cloud/lib/x86_64-pc-linux-gnu-library/4.3'
## (as 'lib' is unspecified)
```

```
library(skimr)

install.packages("rmarkdown") #for documentation
```

```
## Installing package into '/cloud/lib/x86_64-pc-linux-gnu-library/4.3'
## (as 'lib' is unspecified)
```

Next, import the data files into R.

```
#Import data files into R
data_06_2022 <- read_csv("202206-divvy-tripdata.csv")
```

```
## Rows: 769204 Columns: 13
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (7): ride_id, rideable_type, start_station_name, start_station_id, end_...
## dbl  (4): start_lat, start_lng, end_lat, end_lng
```

```
## dttm (2): started_at, ended_at
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
data_07_2022 <- read_csv("202207-divvy-tripdata.csv")
```

```
## Rows: 823488 Columns: 13
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (7): ride_id, rideable_type, start_station_name, start_station_id, end_...
## dbl  (4): start_lat, start_lng, end_lat, end_lng
## dttm (2): started_at, ended_at
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
data_08_2022 <- read_csv("202208-divvy-tripdata.csv")
```

```
## Rows: 785932 Columns: 13
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (9): ride_id, rideable_type, started_at, ended_at, start_station_name, s...
## dbl (4): start_lat, start_lng, end_lat, end_lng
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
data_09_2022 <- read_csv("202209-divvy-tripdata.csv")
```

```
## Rows: 701339 Columns: 13
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (7): ride_id, rideable_type, start_station_name, start_station_id, end_...
## dbl  (4): start_lat, start_lng, end_lat, end_lng
## dttm (2): started_at, ended_at
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
data_10_2022 <- read_csv("202210-divvy-tripdata.csv")
```

```
## Rows: 558685 Columns: 13
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (7): ride_id, rideable_type, start_station_name, start_station_id, end_...
## dbl  (4): start_lat, start_lng, end_lat, end_lng
## dttm (2): started_at, ended_at
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
data_11_2022 <- read_csv("202211-divvy-tripdata.csv")
```

```
## Rows: 337735 Columns: 13
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (7): ride_id, rideable_type, start_station_name, start_station_id, end_...
```

```
## dbl  (4): start_lat, start_lng, end_lat, end_lng
## dttm (2): started_at, ended_at
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
data_12_2022 <- read_csv("202212-divvy-tripdata.csv")
```

```
## Rows: 181806 Columns: 13
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (7): ride_id, rideable_type, start_station_name, start_station_id, end_...
## dbl  (4): start_lat, start_lng, end_lat, end_lng
## dttm (2): started_at, ended_at
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
data_01_2023 <- read_csv("202301-divvy-tripdata.csv")
```

```
## Rows: 190301 Columns: 13
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (7): ride_id, rideable_type, start_station_name, start_station_id, end_...
## dbl  (4): start_lat, start_lng, end_lat, end_lng
## dttm (2): started_at, ended_at
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
data_02_2023 <- read_csv("202302-divvy-tripdata.csv")
```

```
## Rows: 190445 Columns: 13
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (7): ride_id, rideable_type, start_station_name, start_station_id, end_...
## dbl  (4): start_lat, start_lng, end_lat, end_lng
## dttm (2): started_at, ended_at
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
data_03_2023 <- read_csv("202303-divvy-tripdata.csv")
```

```
## Rows: 258678 Columns: 13
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (7): ride_id, rideable_type, start_station_name, start_station_id, end_...
## dbl  (4): start_lat, start_lng, end_lat, end_lng
## dttm (2): started_at, ended_at
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
data_04_2023 <- read_csv("202304-divvy-tripdata.csv")
```

```
## Rows: 426590 Columns: 13
## -- Column specification --------------------------------------------------------
```

```
## Delimiter: ","
## chr  (7): ride_id, rideable_type, start_station_name, start_station_id, end_...
## dbl  (4): start_lat, start_lng, end_lat, end_lng
## dttm (2): started_at, ended_at
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
data_05_2023 <- read_csv("202305-divvy-tripdata.csv")
```

```
## Rows: 604827 Columns: 13
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (7): ride_id, rideable_type, start_station_name, start_station_id, end_...
## dbl  (4): start_lat, start_lng, end_lat, end_lng
## dttm (2): started_at, ended_at
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Next, combine the contents of each of the data files into one large data frame. However, there was an issue with the 'data_08_2022' file that prevented R from being able to combine all the data sets into one. This issue may have occured while I was trying to view each data set on other platforms (e.g. Microsoft Excel).

The 'started_at' and 'ended_at' columns were in the character format instead of datetime. This is how I rectified this issue:

```
data_08_2022$started_at <- as.POSIXct(data_08_2022$started_at, format ="%Y-%m-%d, %H:%M:%S") #To rectif
data_08_2022$ended_at <- as.POSIXct(data_08_2022$ended_at, format ="%Y-%m-%d, %H:%M:%S")
```

Now the data sets can be combined into one:

```
all_trips_12months <- bind_rows(data_06_2022,data_07_2022,data_08_2022,
                                data_09_2022,data_10_2022, data_11_2022,
                                data_12_2022,data_01_2023, data_02_2023, data_03_2023,
                                data_04_2023, data_05_2023)
```

### (b). Inspect Data and Prepare for Cleaning

```
View(all_trips_12months)
glimpse(all_trips_12months) #To have a glance at the data set
skim_without_charts(all_trips_12months) #Provides comprehensive summary of data set
nrow(all_trips_12months) #Shows the number of rows in df(which is a collection of columns)
head(all_trips_12months) #Shows the first 6 rows of df
colnames(all_trips) #Shows no. of columns
```

Three columns were renamed

```
(all_trips_12months <- rename(all_trips_12months
          ,trip_id = ride_id
          ,bike_type = rideable_type
          ,user_type = member_casual))
```

### (c). Data Cleaning Process

*1. Check and remove N/A values:* The drop_na() function was used to remove all roles with N/A values from the data.

```
all_trips_clean <- drop_na(all_trips_12months)
nrow(all_trips_clean)
```

```
## [1] 3889356
```

There are now 3889356 rows compared to 5829030 rows before.

*2. Check and remove duplicates*

```
skim_without_charts(all_trips_clean)
```

Table 1: Data summary

| | |
|---|---|
| Name | all_trips_clean |
| Number of rows | 3889356 |
| Number of columns | 13 |
| _____ | |
| Column type frequency: | |
| character | 7 |
| numeric | 4 |
| POSIXct | 2 |
| _____ | |
| Group variables | None |

**Variable type: character**

| skim_variable | n_missing | complete_rate | min | max | empty | n_unique | whitespace |
|---|---|---|---|---|---|---|---|
| trip_id | 0 | 1 | 16 | 16 | 0 | 3889356 | 0 |
| bike_type | 0 | 1 | 11 | 13 | 0 | 3 | 0 |
| start_station_name | 0 | 1 | 3 | 64 | 0 | 1705 | 0 |
| start_station_id | 0 | 1 | 3 | 35 | 0 | 1402 | 0 |
| end_station_name | 0 | 1 | 3 | 64 | 0 | 1740 | 0 |
| end_station_id | 0 | 1 | 3 | 36 | 0 | 1409 | 0 |
| user_type | 0 | 1 | 6 | 6 | 0 | 2 | 0 |

**Variable type: numeric**

| skim_variable | n_missing | complete_rate | mean | sd | p0 | p25 | p50 | p75 | p100 |
|---|---|---|---|---|---|---|---|---|---|
| start_lat | 0 | 1 | 41.90 | 0.04 | 41.65 | 41.88 | 41.90 | 41.93 | 42.06 |
| start_lng | 0 | 1 | -87.64 | 0.03 | -87.84 | -87.66 | -87.64 | -87.63 | -87.53 |
| end_lat | 0 | 1 | 41.90 | 0.07 | 0.00 | 41.88 | 41.90 | 41.93 | 42.06 |
| end_lng | 0 | 1 | -87.65 | 0.13 | -87.84 | -87.66 | -87.64 | -87.63 | 0.00 |

**Variable type: POSIXct**

| skim_variable | n_missing | complete_rate | min | max | median | n_unique |
|---|---|---|---|---|---|---|
| started_at | 0 | 1 | 2022-06-01 00:00:04 | 2023-05-31 23:59:49 | 2022-10-10 15:26:41 | 3395612 |
| ended_at | 0 | 1 | 2022-06-01 00:02:38 | 2023-06-07 23:04:26 | 2022-10-10 15:46:34 | 3406890 |

No need to remove any duplicate row because the unique values matched the number or rows.

*3. Check for incorrect values*

```
skim_without_charts(all_trips_clean)
```

*(a) Check for values that fall outside of the start and end date-time range*

```
all_trips_clean <- all_trips_clean %>%
  filter(started_at < ended_at) #for rides that only fall between the range
nrow(all_trips_clean) #3889067 rows left, 289 rows deleted
```

```
## [1] 3889067
```

There are now 3889067 rows left, 289 rows were deleted.

*(b) Create 'ride_length' column and remove any outlier*

This is because there are **86,400** seconds in a day and the ride duration should be around this number so that it makes sense.

```
#To create ride_length column

all_trips_clean2 <- all_trips_clean %>%
  mutate(ride_length = as.numeric(as.character(difftime(ended_at, started_at))))


all_trips_clean2 %>%
  select(c(ride_length)) %>%
  summary()
```

```
##   ride_length
##  Min.   :      1.0
##  1st Qu.:    342.0
##  Median :    600.0
##  Mean   :    964.6
##  3rd Qu.:   1074.0
##  Max.   :1922127.0
```

The output shows a max ride_length of 19222127 seconds which is abnormal. This means there are outliers that need to be removed.

```
#Now let's remove the outliers
all_trips_clean <- all_trips_clean2 %>%
  filter(ride_length
         <= 126000)
```

```
#Check
all_trips_clean %>%
  select(c(ride_length)) %>%
  summary()
```

```
##   ride_length
##  Min.   :     1.0
##  1st Qu.:   342.0
##  Median :   600.0
##  Mean   :   963.2
##  3rd Qu.:  1074.0
##  Max.   :106043.0
```

The result shows that the maximum ride_length now is 106043 seconds (converts to approx. 29.5 hours), which is more reasonable. It is possible that a customer returned the bike late to the station after 24 hours or the customer booked a bike for more than a day.

*(c) Check coordinates to ensure they fall within the the correct range*

Chicago's coordinates lie between (41.8781N,87.6298W) so the values should fall within this range.

```
all_trips_clean %>%
  select(c(start_lat, start_lng, end_lat, end_lng)) %>%
  summary()
```

```
##    start_lat       start_lng         end_lat         end_lng
##  Min.   :41.65   Min.   :-87.84   Min.   : 0.00   Min.   :-87.84
##  1st Qu.:41.88   1st Qu.:-87.66   1st Qu.:41.88   1st Qu.:-87.66
##  Median :41.90   Median :-87.64   Median :41.90   Median :-87.64
##  Mean   :41.90   Mean   :-87.64   Mean   :41.90   Mean   :-87.65
##  3rd Qu.:41.93   3rd Qu.:-87.63   3rd Qu.:41.93   3rd Qu.:-87.63
##  Max.   :42.06   Max.   :-87.53   Max.   :42.06   Max.   :  0.00
```

There are no issues because the coordinates make sense.

(d) Remove entries that have '0' values in "end_lat" and "end_lng" columns

```
all_trips_clean <- all_trips_clean %>%
  filter(end_lat!=0, end_lng !=0)

nrow(all_trips_clean)
```

```
## [1] 3889049
```

18 rows were deleted and now there are 3889049 rows left.

## Phase 4: Analyse

Questions to answer for descriptive analysis:

- How much time did users spend riding the bikes? (Avg, median, max, min)
- On what weekday do users ride the most?
- How long did users ride bikes for per day?
- Customer bike usage by month and year
- How far did the users ride?
- What kind of bikes did users prefer?

### (a). How much time did customers spend while riding the bikes?

```
all_trips_clean %>%
  group_by(user_type) %>%
  summarize(Avg_ride_length = mean(ride_length),
            Median_ride_length = median (ride_length),
            Max_ride_length = max(ride_length),
            Min_ride_length = min(ride_length))
```

```
## # A tibble: 2 x 5
##   user_type Avg_ride_length Median_ride_length Max_ride_length Min_ride_length
##   <chr>               <dbl>              <dbl>           <dbl>           <dbl>
## 1 casual              1353.                778          106043               1
## 2 member               724.                518           89872               1
```

On average, casual riders had longer ride durations compared to members.

(b). On what weekday do users ride the most?

```
#First we need to create a 'weekday' variable
Updated_all_trips_clean <- all_trips_clean %>%
  mutate(all_trips_clean,week_day =weekdays(all_trips_clean$started_at))
```

```
#Code chunk to answer q.2
Updated_all_trips_clean %>%
  group_by (user_type) %>%
  summarize(Max_week_day = max(week_day), #Most occurring weekday
            Min_Week_day = min(week_day)) #Least occurring weekday
```

```
## # A tibble: 2 x 3
##   user_type Max_week_day Min_Week_day
##   <chr>     <chr>        <chr>
## 1 casual    Wednesday    Friday
## 2 member    Wednesday    Friday
```

Both casual and member riders seemed to prefer Wednesdays.

(c). How long did customers ride bikes per day?

```
Updated_all_trips_clean %>%
  group_by(week_day,user_type) %>%
  summarize(Avg_ride_length = mean(ride_length),
            Max_ride_length = max(ride_length),#Longest ride
            Min_ride_length = min(ride_length))#Shortest ride
```

```
## `summarise()` has grouped output by 'week_day'. You can override using the
## `.groups` argument.
```

(d). Customer bike usage per month and year

```
#First create 'month' and 'year' variables

Updated_all_trips_clean <- Updated_all_trips_clean %>%
  mutate(all_trips_clean, start_month = month(started_at, label=TRUE, abbr = FALSE)) #For start_month v

Updated_all_trips_clean <- Updated_all_trips_clean %>%
  mutate(all_trips_clean, start_year = format(as.Date(started_at),"%Y")) #For start_year variable
```

```
#Customer bike usage by month & Year code chunk
bike_usage_data <-Updated_all_trips_clean %>%
  group_by(start_month, start_year, user_type) %>%
  summarize(Avg_ride_length = mean(ride_length),
            Max_ride_length = max(ride_length),
            Min_ride_length = min(ride_length))
```

```
## `summarise()` has grouped output by 'start_month', 'start_year'. You can
## override using the `.groups` argument.
```

```
#Total Number of rides Per Customer type
Updated_all_trips_clean %>%
  group_by(user_type) %>%
  summarize(Total_rides = n())
```

```
## # A tibble: 2 x 2
##   user_type Total_rides
##   <chr>           <int>
```

```
## 1 casual        1477671
## 2 member        2411378
```

Member riders took more rides compared to casual riders.

(e). How far customers rode

```
#But first we need to create a distance column using the Haversine function
Updated_all_trips_clean <- Updated_all_trips_clean %>%
  mutate(Updated_all_trips_clean, distance_m = distHaversine(cbind(start_lat, start_lng),
                                                             cbind(end_lat, end_lng)))
```

```
#Ride distance calculations
Updated_all_trips_clean %>%
  group_by(user_type) %>%
  summarize(Average_ride_distance = mean(distance_m),
            Max_ride_distance = max(distance_m),
            Min_ride_distance = min(distance_m))
```

```
## # A tibble: 2 x 4
##   user_type Average_ride_distance Max_ride_distance Min_ride_distance
##   <chr>                     <dbl>             <dbl>             <dbl>
## 1 casual                    1385.            22125.                 0
## 2 member                    1393.            20235.                 0
```

Casual riders seemed to have journeyed the farthest compared to members. However, members had a higher ride distance on average.

(f). Types of bikes used by each kind of user

```
Bike_type_bar <-Updated_all_trips_clean %>%
  group_by(bike_type, user_type, start_month) %>%
  summarize(Total_rides = n())
```

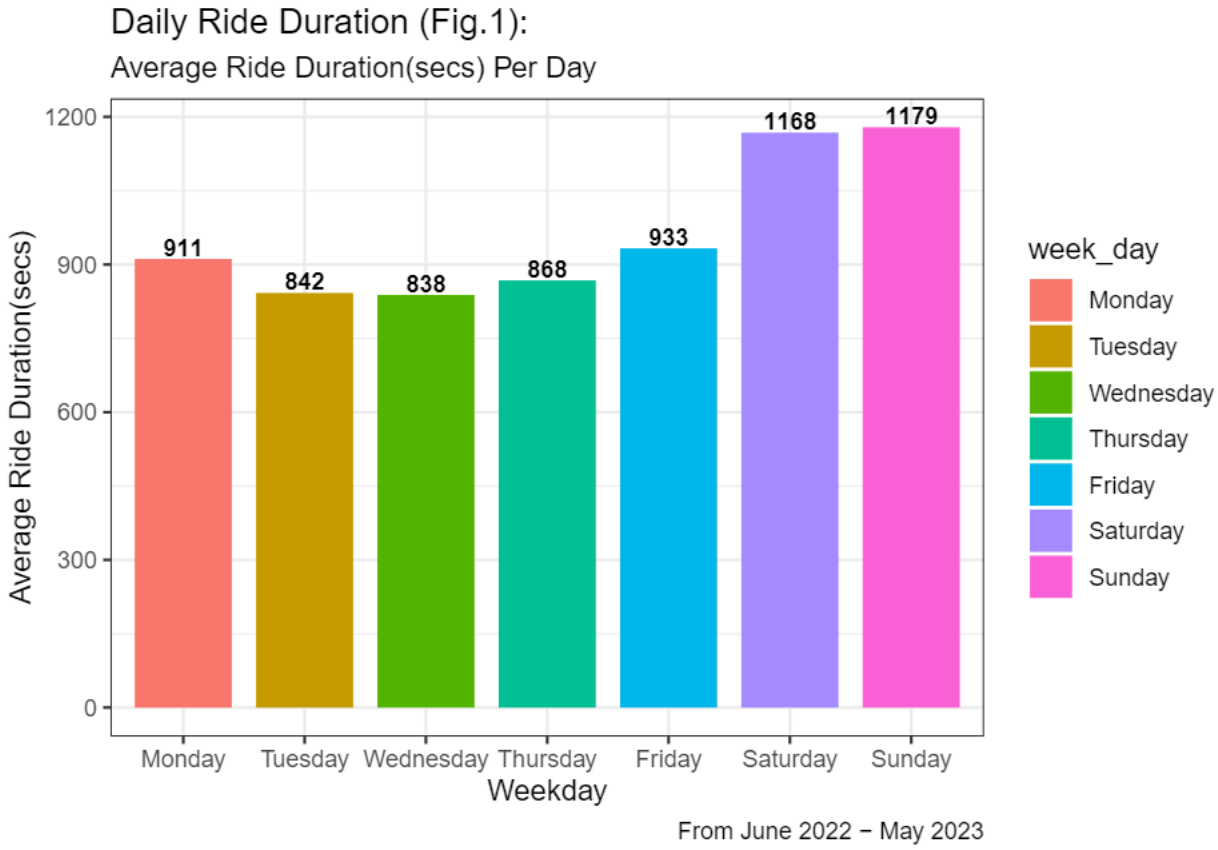
```
## `summarise()` has grouped output by 'bike_type', 'user_type'. You can override
## using the `.groups` argument.
```

## Phase 5: Share

### Customer ride duration per day?

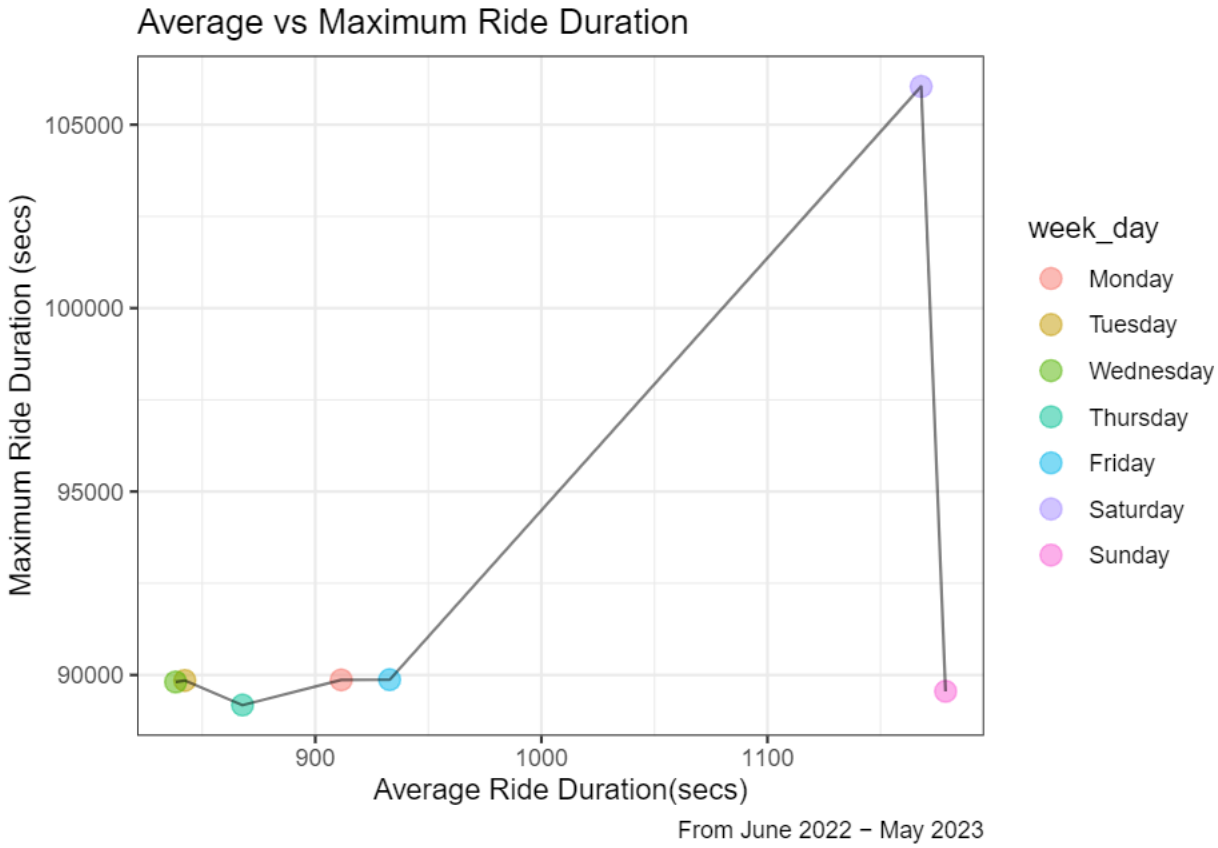
```
#Change ordering of weekday manually
Updated_all_trips_clean$week_day <- factor(Updated_all_trips_clean$week_day, levels = c("Monday", "Tues
                                    "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"))

Ride_length_daily <- Updated_all_trips_clean %>%
  group_by(week_day) %>%
  summarize(Avg_ride_length = mean(ride_length),
            Max_ride_length = max(ride_length),#Longest ride
            Min_ride_length = min(ride_length))#Shortest ride


Ride_length_daily %>%
  ggplot(aes(week_day, Avg_ride_length, fill = week_day))+
geom_bar(stat='identity', width=0.8)+ #identity tells r to use Avg_ride_length as the value for y axis
  geom_text(aes(label =round(Avg_ride_length)), vjust = -0.18,
            fontface="bold", size = 3.2)+
  labs(title ="Daily Ride Duration (Fig.1):", subtitle="Average Ride Duration(secs) Per Day",
       caption="From June 2022 - May 2023")+
```

```
  xlab("Weekday")+
  ylab("Average Ride Duration(secs)")+
theme_bw()
```




It is clear to see that customers generally preferred to ride during the weekends especially on Saturdays and Sundays.

```
#Compare Average ride duration to maximum ride duration per week_day(scatterplot)
Ride_length_daily %>%
  ggplot(aes(Avg_ride_length, Max_ride_length, color=week_day))+
  geom_point(size=3.5, alpha=0.5)+
  geom_line(color = "black", alpha=0.5)+
  labs(title= "Average vs Maximum Ride Duration", caption="From June 2022 - May 2023")+
  xlab("Average Ride Duration(secs)")+
  ylab("Maximum Ride Duration (secs)")+
  theme_bw()
```

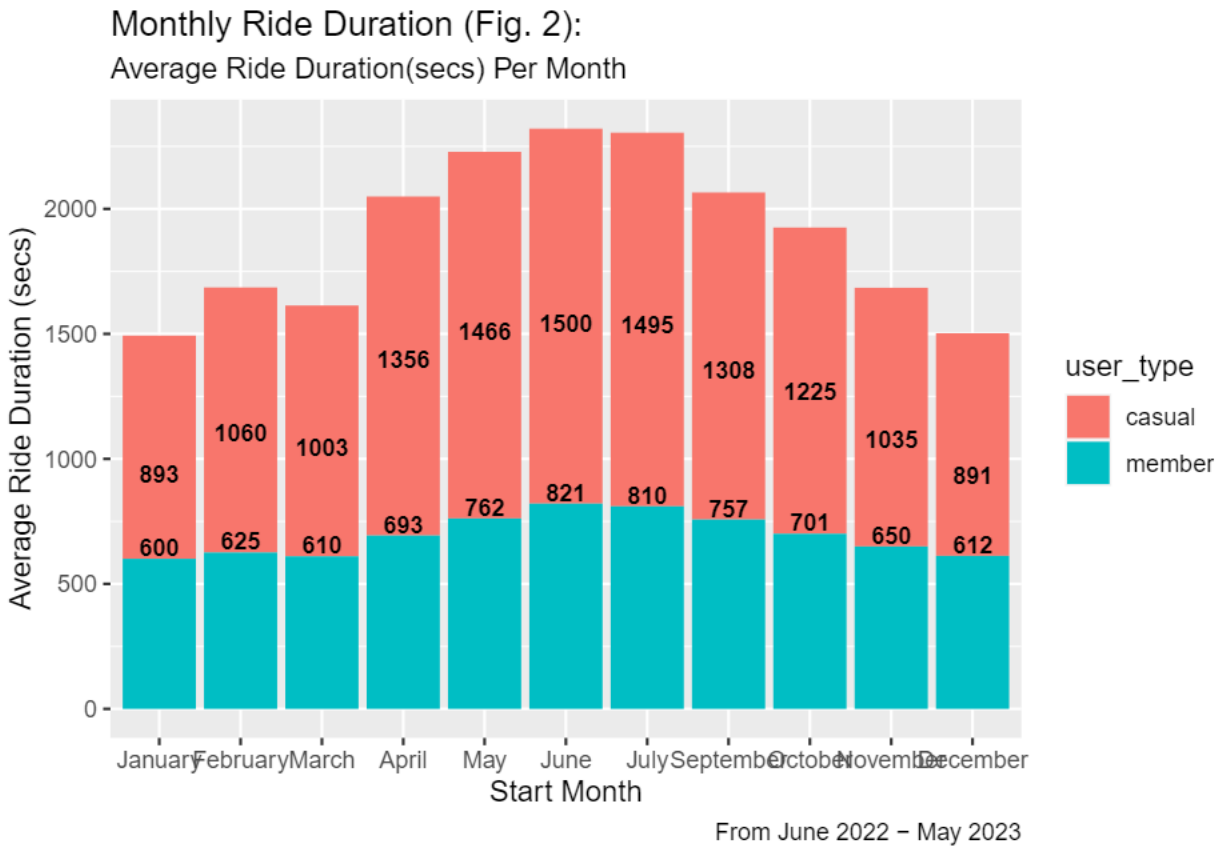



Customer bike usage per month and year

```
bike_usage_data <-Updated_all_trips_clean %>%
  group_by(start_month, start_year, user_type) %>%
  summarize(Avg_ride_length = mean(ride_length),
            Max_ride_length = max(ride_length),
            Min_ride_length = min(ride_length))
```

```
## `summarise()` has grouped output by 'start_month', 'start_year'. You can
## override using the `.groups` argument.
```

```
#By Month
ggplot(bike_usage_data)+
  geom_col(mapping=aes(x=start_month, y=Avg_ride_length, fill=user_type))+
  labs(title ="Monthly Ride Duration (Fig. 2):", subtitle="Average Ride Duration(secs) Per Month",
       caption="From June 2022 - May 2023")+
  geom_text(aes(x= start_month, y= Avg_ride_length, label =round(Avg_ride_length)),
                vjust=-0.15, fontface="bold", size=3.2)+
  xlab("Start Month")+
  ylab("Average Ride Duration (secs)")
```




Bike activity was at its peak in June 2022. Data shows that casual riders used Cyclistic's bikes more than members did.

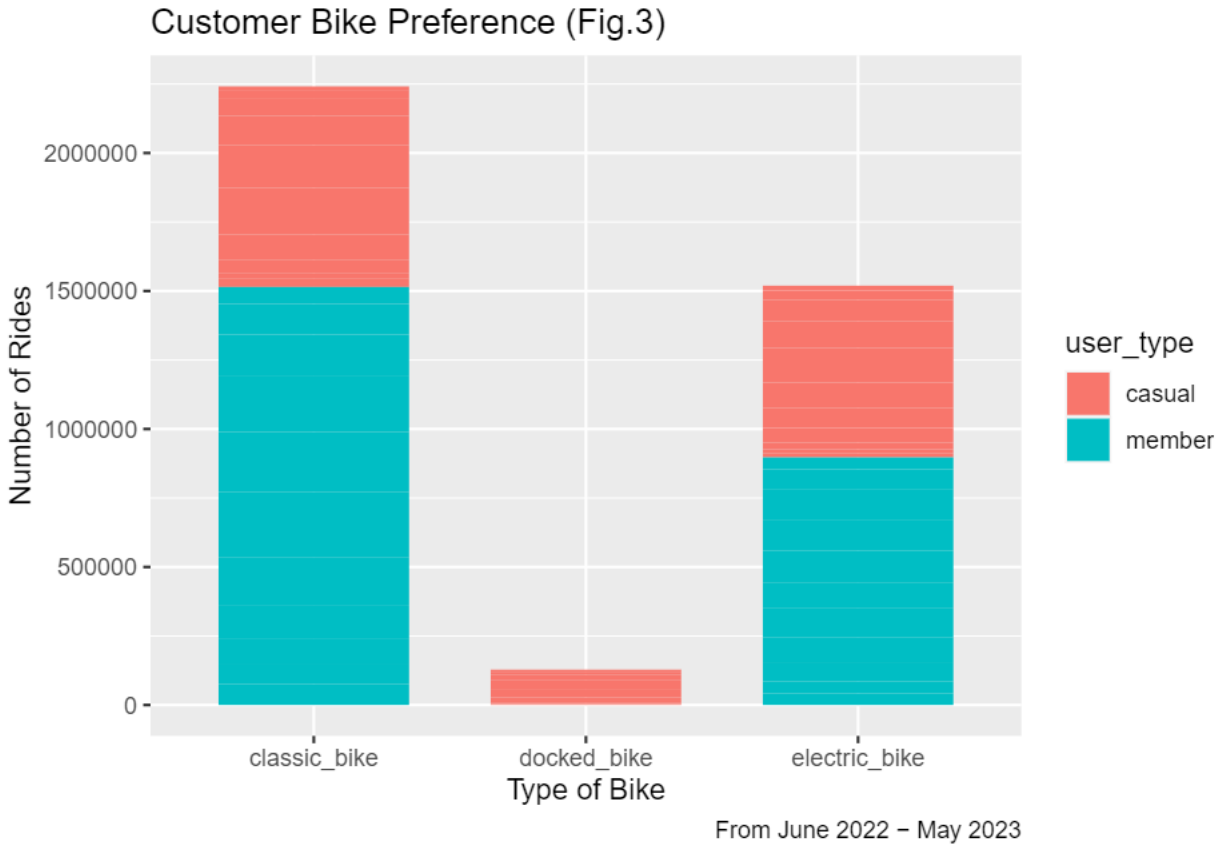
### Types of Bikes used by each kind of user

```
Bike_type_bar <-Updated_all_trips_clean %>%
  group_by(bike_type, user_type, start_month) %>%
  summarize(Total_rides = n())
```

```
## `summarise()` has grouped output by 'bike_type', 'user_type'. You can override
## using the `.groups` argument.
```

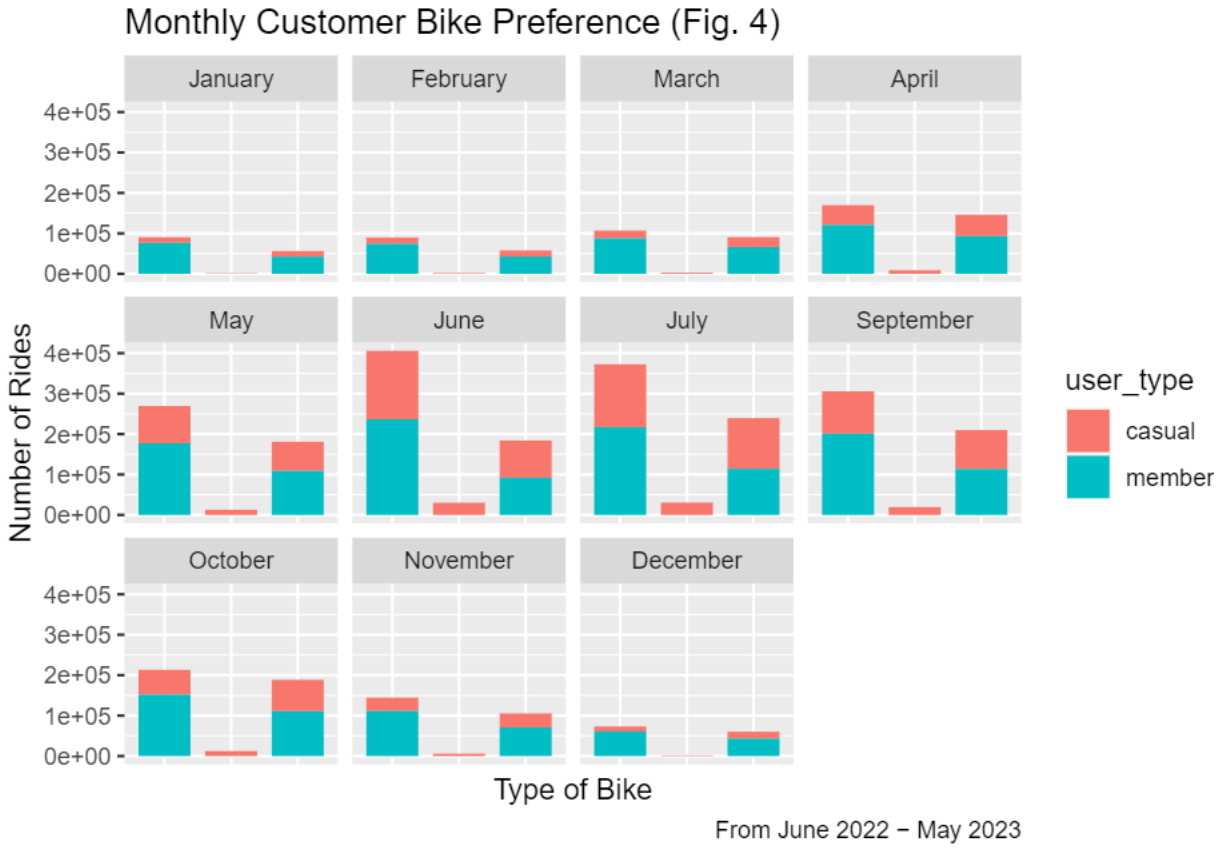
```
ggplot(Bike_type_bar)+
geom_col(mapping = aes(x=bike_type, y=Total_rides, fill=user_type, width=0.70))+
  labs(title="Customer Bike Preference (Fig.3)", caption="From June 2022 - May 2023")+
  xlab("Type of Bike")+
  ylab("Number of Rides")
```

```
## Warning in geom_col(mapping = aes(x = bike_type, y = Total_rides, fill =
## user_type, : Ignoring unknown aesthetics: width
```




```
#Types of Bikes Used Per Month
ggplot(Bike_type_bar)+
  geom_col(mapping = aes(x=bike_type, y=Total_rides, fill=user_type, width=0.78))+
  facet_wrap(~start_month)+
  labs(title="Monthly Customer Bike Preference (Fig. 4)", caption="From June 2022 - May 2023")+
  xlab("Type of Bike")+
  ylab("Number of Rides")+
  theme(axis.text.x = element_blank(), axis.ticks.x = element_blank())#To remove x axis labels
```

```
## Warning in geom_col(mapping = aes(x = bike_type, y = Total_rides, fill =
## user_type, : Ignoring unknown aesthetics: width
```




Customer bike preferences seemed to vary on a monthly basis between casual riders and riders with annual memberships (see Fig. 4). However the data shows that overall, customers preferred to use the classic bikes instead of electric or docked bikes.

## Phase 6: Act

**Conclusion** The findings from my analysis justify the marketing team's desire/need to convert casual riders to members. This is because casual riders seemingly use the bikes a lot more and so convincing them to become annual members could potentially bring in a lot of revenue. Cyclistic would not only make money from increased subcriptions, but also increased member usage assuming that the newly converted members maintain the same rate of usage as when they casual riders.

**Recommendations/Next Steps**

1. Marketing campaigns targeted towards both customer types but mainly casual riders should include more bike usage offerings on the weekends. Especially on Sundays.
2. Conduct a survey among existing casual riders to understand why they generally prefer classic bikes over the rest. This could help form a new marketing campaign specifically tailored to meet their preferences.
3. Start a loyalty program with rewards, discounts and other special offers as a way to incentivize casual riders to purchase annual memberships.