Case study: How does a bike-share navigate speedy success?

Introduction

This case study is the capstone of the Google Data Analytics Professional Certificate. In this case study, we work for a fictional company.

We are acting as a junior data analyst working on 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, our team wants to understand how casual riders and annual members use Cyclistic bikes differently.

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

In order to answer the business questions, the steps of the data analysis process: Ask, Prepare, Process, Analyze, Share, and Act, will be followed.

Business Case

The Cyclistic 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.
It sets itself apart by also offering reclining bikes, hand tricycles, and cargo bikes, making bike-share more inclusive to people with disabilities and riders who can't use a standard two-wheeled bike.

The pricing plan is based on flexibility: 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.

Problematic

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, maximizing the number of annual members would be key to future growth.

Rather than creating a marketing campaign that targets all-new customers, there is a solid opportunity to convert casual riders into members. Casual riders are already aware of the Cyclistic program and have chosen Cyclistic for their mobility needs.

Ask

The goal is to design marketing strategies aimed at converting casual riders into annual members. In order to do that, however, the team needs to better understand:

1. How do annual members and casual riders use Cyclistic bikes differently?
2. Why would casual riders buy Cyclistic annual memberships?
3. How can Cyclistic use digital media to influence casual riders to become members?

The business task is to study the differences of usage between members and casual riders. Based on these differences, we will identify how members prefer to ride, and how casual riders use the services of Cyclistic. In that way we can identify insights that would help Cyclistic to encourage casual riders to become members.

We will have to analyze the Cyclistic historical bike trip data to identify trends.

Prepare

Data location

The datasets are located on an Amazon server. The datasets have a different name because Cyclisticis a fictional company. The data has been made available by Divvy, a program of the Chicago Department of Transportation, which owns the city’s bikes, stations and vehicles, under license.

Source

This is public data that we are going to use to explore how different customer types are using Cyclistic bikes. The data can be considered as credible.

Good data?

Data is ROCCC compliant as it is:

- **R**eliable: data represents the usage of all stations for all users, members or casual, with no bias.
- **O**riginal: The data comes from a direct source, not from an intermediate or a tier source.
- **C**omprehensive:The datasets are comprehensive and all information we need.
- **C**urrent: The datasets are up to date and relevant. Data is refreshed each month.
- **C**ited: The data are cited. The source is known and reliable.

Data organization

The datasets are split by months, in a zip format. It is required to download the last 12 previous months. The case study started in June 2024.

202306-divvy-tripdata.zip

202307-divvy-tripdata.zip

202308-divvy-tripdata.zip

202309-divvy-tripdata.zip

202310-divvy-tripdata.zip

202311-divvy-tripdata.zip

202312-divvy-tripdata.zip

202401-divvy-tripdata.zip

202402-divvy-tripdata.zip

202403-divvy-tripdata.zip

202404-divvy-tripdata.zip

202405-divvy-tripdata.zip

Datasets structure

Each dataset contains a CSV file displaying data by ride from a start station to an end station:

Column	Description	Example
ride_id	Identification of the ride.	*6F1682AC40EB6F71*
rideable_type	3 different types of bike used (electric, docked, classic).	*electric_bike*
started_at	Date and time of the ride start (format : yyyy-mm-dd h24:mn:ss).	*2023-06-05 13:34:12*
ended_at	Date and time of the ride end (format : yyyy-mm-dd h24:mn:ss).	*2023-06-05 14:31:56*
start_station_name	Name of the start station.	*2112 W Peterson Ave*
start_station_id	Identification of the start station.	*KA1504000155*
end_station_name	Name of the end station.	*Clark St & Bryn Mawr Ave*

Column	Description	Example
end_station_id	Identification of the end station.	*KA1504000151*
start_lat	Latitude of the start station.	*41.991220117*
start_lng	Longitude of the start station.	*41.9840446107*
end_lat	Latitude of the end station.	*41.983593*
end_lng	Longitude of the end station.	*-87.669154*
member_casual	Type of user: casual or member.	*member*

First Data integrity Check

A first data integrity for each file is performed with a spreadsheet like Excel or Google Spreadsheet.

- Some fields have no values for an entry; this is a case for start_station_name, start_station_id, end_station_name, end_station_id, end_lat, end_lng
- Some dates in a monthly file give information about the next month or the previous month. For example, a member takes a bike on August 31st and gives it back on September 1st.
- Some end_lat and end_lng values equal 0
- Some ride_id have a different format (e.g. 1886432520245480 when other values are hexadecimal on 16 characters
- The format of start_station_id and end_station_id is not consistent (TA1309000033, 866, WL-011, …)

These remarks must be taken into account in the next steps and see if actions are needed or not.

Data credibility

These datasets are coming from a trusted source. They are ROCCC compliant. We can use them in our study. In the next steps more in-depth integrity checks will be performed.

Process

In this step, we use R as a tool to check, clean and transform data to be ready for analysis. I would like to use SQL but I could not use Big Queries with the large dataset that we have to work on.

Read CSV files

I will use the tidyverse package:

```
if(!require(tidyverse)){
install.packages("tidyverse",repos = "http://cran.us.r-project.org")
library(tidyverse)
}
```

Then all the 12 CSV files are read

```
setwd("./datasets")
df2306 <- read.csv("202306-divvy-tripdata.csv")
df2307 <- read.csv("202307-divvy-tripdata.csv")
df2308 <- read.csv("202308-divvy-tripdata.csv")
df2309 <- read.csv("202309-divvy-tripdata.csv")
df2310 <- read.csv("202310-divvy-tripdata.csv")
df2311 <- read.csv("202311-divvy-tripdata.csv")
df2312 <- read.csv("202312-divvy-tripdata.csv")
df2401 <- read.csv("202401-divvy-tripdata.csv")
df2402 <- read.csv("202402-divvy-tripdata.csv")
df2403 <- read.csv("202403-divvy-tripdata.csv")
df2404 <- read.csv("202404-divvy-tripdata.csv")
df2405 <- read.csv("202405-divvy-tripdata.csv")
setwd("../")
```

All datasets are merged into one data frame dfRide.

```
dfRide = rbind(df2306,df2307,df2308,df2309,df2310,df2311,df2312,df2401,df2402,df2403,df2404,d
f2405)
rm(df2306,df2307,df2308,df2309,df2310,df2311,df2312,df2401,df2402,df2403,df2404,df2405)
```

We check the dimension of the dataframe

```
dim(dfRide)
```

```
## [1] 5743278      13
```

5,743,278 rows for 13 columns. Let’s see the structure with str(dfRide)

```
 str(dfRide)
```

```
## 'data.frame':    5743278 obs. of  13 variables:
##  $ ride_id           : chr  "6F1682AC40EB6F71" "622A1686D64948EB" "3C88859D926253B4" "EAD8
A5E0259DEC88" ...
##  $ rideable_type     : chr  "electric_bike" "electric_bike" "electric_bike" "electric_bike
" ...
##  $ started_at        : chr  "2023-06-05 13:34:12" "2023-06-05 01:30:22" "2023-06-20 18:15:
49" "2023-06-19 14:56:00" ...
##  $ ended_at          : chr  "2023-06-05 14:31:56" "2023-06-05 01:33:06" "2023-06-20 18:32:
05" "2023-06-19 15:00:35" ...
##  $ start_station_name: chr  "" "" "" "" ...
##  $ start_station_id  : chr  "" "" "" "" ...
##  $ end_station_name  : chr  "" "" "" "" ...
##  $ end_station_id    : chr  "" "" "" "" ...
##  $ start_lat         : num  41.9 41.9 42 42 42 ...
##  $ start_lng         : num  -87.7 -87.7 -87.7 -87.7 -87.7 ...
##  $ end_lat           : num  41.9 41.9 41.9 42 42 ...
##  $ end_lng           : num  -87.7 -87.7 -87.6 -87.7 -87.7 ...
##  $ member_casual     : chr  "member" "member" "member" "member" ...
```

Looking for inconsistencies

We first check for duplicated rows.

```
dfRide[duplicated(dfRide),]
```

```
##  [1] ride_id            rideable_type      started_at         ended_at
##  [5] start_station_name start_station_id   end_station_name   end_station_id
##  [9] start_lat          start_lng          end_lat            end_lng
## [13] member_casual
## <0 rows> (or 0-length row.names)
```

There are no duplicated rows.

ride_id

ride_id acts as the primary key of the dataset. Hopefully it is never empty.

```
dfRide %>% filter(is.na(rideable_type)) %>% count()
```

```
##   n
## 1 0
```

It is also unique.

```
dfRide$ride_id[duplicated(dfRide$ride_id)]
```

```
## character(0)
```

rideable_type

rideable_type is never empty.

```
dfRide %>% filter(is.na(rideable_type)) %>% count()
```

```
##   n
## 1 0
```

It contains only 3 values.

```
unique(dfRide$rideable_type)
```

```
## [1] "electric_bike" "classic_bike"  "docked_bike"
```

49.27% of classic bikes, 0.86% of decked bikes, and 49.87% of electric bikes.

```
dfRide %>%
  group_by(rideable_type) %>%
  count()
```

```
## # A tibble: 3 × 2
## # Groups:   rideable_type [3]
##   rideable_type       n
##   <chr>           <int>
## 1 classic_bike  2829728
## 2 docked_bike     49355
## 3 electric_bike 2864195
```

started_at, ended_at

The fields are never empty.

```
dfRide %>%
  filter(is.na(started_at)) %>%
  count()
```

```
##   n
## 1 0
```

```
dfRide %>%
  filter(is.na(ended_at)) %>%
  count()
```

```
##   n
## 1 0
```

Some observations are impossible: case when the end is before the start.

start_station_id, start_station_name, end_station_id, end_station_name

They are not null.

```
dfRide %>%
  filter(is.na(start_station_id)) %>%
  count()
```

```
##   n
## 1 0
```

```
dfRide %>%
  filter(is.na(end_station_id)) %>%
  count()
```

```
##   n
## 1 0
```

```
dfRide %>%
  filter(is.na(start_station_name)) %>%
  count()
```

```
##   n
## 1 0
```

```
dfRide %>%
  filter(is.na(end_station_name)) %>%
  count()
```

```
##   n
## 1 0
```

However, they can store an empty string.

```
dfRide %>%
  filter(str_length(start_station_id) == 0) %>%
  count()
```

```
##        n
## 1 905237
```

```
dfRide %>%
  filter(str_length(start_station_name) == 0) %>%
  count()
```

```
##        n
## 1 905237
```

```
dfRide %>%
  filter(str_length(end_station_id) == 0) %>%
  count()
```

```
##        n
## 1 956579
```

```
dfRide %>%
  filter(str_length(end_station_name) == 0) %>%
  count()
```

```
##        n
## 1 956579
```

Empty start station fields are due to electric bike usage mainly.

```
dfRide %>%
  filter(str_length(start_station_name) == 0 | str_length(start_station_id) == 0) %>%
group_by(rideable_type) %>%
  count()
```

```
## # A tibble: 2 × 2
## # Groups:   rideable_type [2]
##   rideable_type      n
##   <chr>          <int>
## 1 classic_bike      34
## 2 electric_bike 905203
```

The same is happening for end station fields

```
dfRide %>%
  filter(str_length(end_station_name) == 0 & str_length(end_station_id) == 0) %>%
  group_by(rideable_type) %>%
  count()
```

```
## # A tibble: 3 × 2
## # Groups:   rideable_type [3]
##   rideable_type      n
##   <chr>          <int>
## 1 classic_bike    6615
## 2 docked_bike     1360
## 3 electric_bike 948604
```

When one of the start station fields is empty, the other one is empty too.

```
dfRide %>%
  filter(str_length(start_station_name) == 0 & str_length(start_station_id) != 0)
```

```
##  [1] ride_id            rideable_type      started_at         ended_at
##  [5] start_station_name start_station_id   end_station_name   end_station_id
##  [9] start_lat          start_lng          end_lat            end_lng
## [13] member_casual
## <0 rows> (or 0-length row.names)
```

```
dfRide %>%
  filter(str_length(start_station_name) != 0 & str_length(start_station_id) == 0)
```

```
##  [1] ride_id            rideable_type      started_at         ended_at
##  [5] start_station_name start_station_id   end_station_name   end_station_id
##  [9] start_lat          start_lng          end_lat            end_lng
## [13] member_casual
## <0 rows> (or 0-length row.names)
```

We observe the same behaviour for end station fields.

```
dfRide %>%
  filter(str_length(end_station_name) == 0 & str_length(end_station_id) != 0)
```

```
##  [1] ride_id            rideable_type      started_at         ended_at
##  [5] start_station_name start_station_id   end_station_name   end_station_id
##  [9] start_lat          start_lng          end_lat            end_lng
## [13] member_casual
## <0 rows> (or 0-length row.names)
```

```
dfRide %>%
  filter(str_length(end_station_name) != 0 & str_length(end_station_id) == 0)
```

```
##  [1] ride_id            rideable_type      started_at         ended_at
##  [5] start_station_name start_station_id   end_station_name   end_station_id
##  [9] start_lat          start_lng          end_lat            end_lng
## [13] member_casual
## <0 rows> (or 0-length row.names)
```

The start station fields and the end station fields do not follow the same logic. If one of the start station fields is empty, it does not mean that the corresponding end station field will be empty too, and vice versa.

```
dfRide %>%
  filter(str_length(start_station_id) != 0 & str_length(end_station_id) == 0) %>%
  count()
```

```
##        n
## 1 523145
```

```
dfRide %>%
   filter(str_length(start_station_id) == 0 & str_length(end_station_id) != 0) %>%
   count()
```

```
##        n
## 1 471803
```

The next code section tests if a name of a start station is associated with one and unique station id. We notice that several start station id fields share different names. Some are different because of different wording or orthography, but others are totally different.

```
df1 <- dfRide %>%
  group_by(start_station_id, start_station_name) %>%
  summarise (n1 = n(), .groups = 'drop')

df1 <- df1 %>%
  group_by(start_station_id) %>%
  summarise (n2 = n(), .groups = 'drop') %>%
  filter (n2 >1)

dfRide %>%
  select (start_station_id, start_station_name) %>%
  filter (start_station_id %in% df1$start_station_id) %>%
  unique() %>%
  arrange(start_station_id, start_station_name)
```

```
##     start_station_id                       start_station_name
## 1              13290                  Noble St & Milwaukee Ave
## 2              13290           Noble St & Milwaukee Ave (Temp)
## 3              15541                       Buckingham Fountain
## 4              15541      Buckingham Fountain (Columbus/Balbo)
## 5              15541       Buckingham Fountain (Michigan/11th)
## 6              15541                Buckingham Fountain (Temp)
## 7         15541.1.1                     Buckingham - Fountain
## 8         15541.1.1                       Buckingham Fountain
## 9              21322                            Grace & Cicero
## 10             21322                     Grace St & Cicero Ave
## 11             21366                      Spaulding Ave & 16th
## 12             21366                   Spaulding Ave & 16th St
## 13             21371                     Kildare & Chicago Ave
## 14             21371                 Kildare Ave & Chicago Ave
## 15             23215                Lexington & California Ave
## 16             23215             Lexington St & California Ave
## 17             24156                 Glenlake Ave & Pulaski Rd
## 18             24156                Granville Ave & Pulaski Rd
## 19               514       Public Rack - Hamlin Ave & Grand Ave
## 20               514                    Ridge Blvd & Howard St
## 21               515                    Paulina St & Howard St
## 22               515     Public Rack - Hamlin Ave & Chicago Ave
## 23               517                     Clark St & Jarvis Ave
## 24               517    Public Rack - Pulaski Rd & Armitage Ave
## 25               518                 Conservatory Dr & Lake St
## 26               518     Public Rack - Keystone Ave & North Ave
## 27               519      Public Rack - Kostner Ave & North Ave
## 28               519                   Wolcott Ave & Fargo Ave
## 29               520                Greenview Ave & Jarvis Ave
## 30               520    Public Rack - Karlov Ave & Kamerling Ave
## 31               523                  Eastlake Ter & Howard St
## 32               523    Public Rack - Pulaski Rd & Roosevelt Rd
## 33               525                  Glenwood Ave & Touhy Ave
## 34               525    Public Rack - Kedzie Ave & Arthington St
## 35               528         Public Rack - Pulaski Rd & 15th St
## 36               528                     Pulaski Rd & Lake St
## 37               534                    Karlov Ave & Madison St
## 38               534    Public Rack - California Ave & Ogden Ave
## 39               535              Public Rack - Zapata Academy
## 40               535                Pulaski Rd & Congress Pkwy
## 41               536                     Kostner Ave & Lake St
## 42               536         Public Rack - Keeler Ave & 26th St
## 43               537                   Kenton Ave & Madison St
## 44               537           Public Rack - 2302 S Pulaski Rd
## 45               543                  Laramie Ave & Gladys Ave
## 46               543       Public Rack - Cicero Ave & Roscoe St
## 47               545                    Kostner Ave & Adams St
## 48               545      Public Rack - Linder Ave & Belmont Ave
## 49               546                   Damen Ave & Pershing Rd
## 50               546   Public Rack - Cicero Ave & Wellington Ave
```

```
## 51                549                                Marshfield Ave & 44th St
## 52                549                 Public Rack - Laramie Ave & Fullerton Ave
## 53                553                                  Elizabeth St & 47th St
## 54                553                     Public Rack - Lorel Ave & Chicago Ave
## 55                554                                     Damen Ave & 51st St
## 56                554 Public Rack - Cicero Ave & Le Moyne St - midblock
## 57                559    Public Rack - Menard Ave & Dakin St - midblock
## 58                559                               Racine Ave & Garfield Blvd
## 59                560                                Marshfield Ave & 59th St
## 60                560                      Public Rack - Austin Ave & Roscoe St
## 61                561                                     Damen Ave & 59th St
## 62                561                  Public Rack - Melvina Ave & Belmont Ave
## 63                562                   Public Rack - Menard Ave & Belmont Ave
## 64                562                                    Racine Ave & 61st St
## 65                564                 Public Rack - Austin Ave & Wellington Ave
## 66                564                                    Racine Ave & 65th St
## 67                567                                        May St & 69th St
## 68                567                     Public Rack - Harvey Ave & North Ave
## 69                569                     Public Rack - Menard Ave & Grand Ave
## 70                569                                  Woodlawn Ave & 75th St
## 71                570                                     Evans Ave & 75th St
## 72                570                   Public Rack - McVicker Ave & Grand Ave
## 73                571                    Public Rack - Austin Blvd & North Ave
## 74                571                                    Vernon Ave & 75th St
## 75                572                              Public Rack - Hiawatha Park
## 76                572                                      State St & 76th St
## 77                573    Public Rack - Panama Ave & Forest Preserve Ave
## 78                573                                      State St & 79th St
## 79                574                    Public Rack - Canty Elementary School
## 80                574                                    Vernon Ave & 79th St
## 81                575                              Cottage Grove Ave & 78th St
## 82                575               Public Rack - Pittsburgh Ave & Irving Park
## 83                577                     Public Rack - Ozark Ave & Addison St
## 84                577                     Stony Island Ave & South Chicago Ave
## 85                579                                  Phillips Ave & 79th St
## 86                579                    Public Rack - Oketo Ave & Belmont Ave
## 87                583                   Public Rack - Baltimore Ave & 134th St
## 88                583                               Stony Island Ave & 82nd St
## 89                584                                     Ellis Ave & 83rd St
## 90                584                   Public Rack - Baltimore Ave & 132nd St
## 91                585                              Cottage Grove Ave & 83rd St
## 92                585                     Public Rack - Houston Ave & 131st St
## 93                586                                      MLK Jr Dr & 83rd St
## 94                586                     Public Rack - Stewart Ave & 123rd St
## 95                590                             Kilbourn Ave & Irving Park Rd
## 96                590                          Public Rack - Ada St & 117th St
## 97                594                     Public Rack - Indiana Ave & 111th St
## 98                594                                  Western Blvd & 48th Pl
## 99                599                      Public Rack - Avenue J & 112th St
## 100               599                            Valli Produce - Evanston Plaza
## 101               604                   Public Rack - Wentworth Ave & 103rd St
```

```
## 102          604                 Sheridan Rd & Noyes St (NU)
## 103          620          Orleans St & Chestnut St (NEXT Apts)
## 104          620                Public Rack - Ada St & 95th St
## 105          623                      Michigan Ave & 8th St
## 106          623            Public Rack - Halsted St & 102nd St
## 107          624                   Dearborn St & Van Buren St
## 108          624            Public Rack - Parnell Ave & 98th St
## 109          631                            Malcolm X College
## 110          631             Public Rack - Yates Ave & 100th St
## 111          636                      Orleans St & Hubbard St
## 112          636             Public Rack - Ewing Ave & 101st St
## 113          637            Public Rack - Ewing Ave & 96th St N
## 114          637                        Wood St & Chicago Ave
## 115          638                     Clinton St & Jackson Blvd
## 116          638      Public Rack - Ewing Ave & Indianapolis Ave
## 117          639                  Lakefront Trail & Wilson Ave
## 118          639              Public Rack - Ewing Ave & 99th St
## 119          642                     Latrobe Ave & Chicago Ave
## 120          642             Public Rack - Justine St & 87th St
## 121          643           Public Rack - Vincennes Ave & 87th St
## 122          643                                   Smith Park
## 123          644             Public Rack - Wabash Ave & 87th St
## 124          644                     Western Ave & Fillmore St
## 125          646     Public Rack - Cottage Grove Ave & 87th St
## 126          646                          State St & 54th St
## 127          647                      Elizabeth St & 59th St
## 128          647                        Racine Ave & 57th St
## 129          650                     Eggleston Ave & 69th St
## 130          650            Public Rack - Houston Ave & 91st St
## 131          651                      Michigan Ave & 71st St
## 132          651         Public Rack - Commercial Ave & 89th St
## 133          654             Public Rack - Racine Ave & 83rd St
## 134          654                  Racine Ave & Washington Blvd
## 135          655                      Hoyne Ave & Balmoral Ave
## 136          655            Public Rack - Sangamon St & 79th St
## 137          657           Public Rack - Wentworth Ave & 79th St
## 138          657                       Wood St & Augusta Blvd
## 139          658                      Leavitt St & Division St
## 140          658                Public Rack - King Dr & 83rd St
## 141          660            Public Rack - Prairie Ave & 85th St
## 142          660                   Sheridan Rd & Columbia Ave
## 143          661                         Evanston Civic Center
## 144          661            Public Rack - Langley Ave & 79th St
## 145          662                      Dodge Ave & Mulford St
## 146          662           Public Rack - Cottage Grove & 86th St
## 147          665     Public Rack - 83rd St (Avalon Park) Metra
## 148          665                South Chicago Ave & Elliot Ave
## 149  KA1503000074         Griffin Museum of Science and Industry
## 150  KA1503000074                Museum of Science and Industry
## 151  TA1305000030                      Clark St & Randolph St
## 152  TA1305000030                      Wells St & Randolph St
```

```
## 153     TA1309000042                    Lincoln Ave & Belmont Ave (Temp)
## 154     TA1309000042                            Lincoln Ave & Melrose St
```

154 observations for start station fields are in this case. The same problem is happening with end station id and end station name fields for 157 observations.

Start_lat, end_lat, start_lng, and end_lng

The fields and_lat and end_lng can be empty.

```
dfRide %>% filter(is.na(start_lat)) %>% count()
```

```
##   n
## 1 0
```

```
dfRide %>% filter(is.na(end_lat)) %>% count()
```

```
##      n
## 1 7684
```

```
dfRide %>% filter(is.na(start_lng)) %>% count()
```

```
##   n
## 1 0
```

```
dfRide %>% filter(is.na(end_lng)) %>% count()
```

```
##      n
## 1 7684
```

```
dfRide %>% filter(is.na(end_lat) & is.na(end_lng)) %>% count()
```

```
##      n
## 1 7684
```

They can be empty even if we have the information about the end station.

```
dfRide %>%
  summarise (max_start_lat = max(start_lat),
  min_start_lat = min(start_lat),
  max_start_lng = max(start_lng),
  min_start_lng = min(start_lng),
  max_end_lat = max(end_lat),
  min_end_lat = min(end_lat),
  max_end_lng = max(end_lng),
  min_end_lng = min(end_lng))
```

```
##   max_start_lat min_start_lat max_start_lng min_start_lng max_end_lat
## 1         42.07         41.63        -87.46        -87.94          NA
##   min_end_lat max_end_lng min_end_lng
## 1          NA          NA          NA
```

member_casual

The field is never empty and stores only 2 values for member or casual.

```
dfRide %>%
  filter(is.na(member_casual)) %>% count()
```

```
##   n
## 1 0
```

```
unique(dfRide$member_casual)
```

```
## [1] "member" "casual"
```

I was wondering if each station has one latitude and longitude value. It is not the case. Here is an example with the start station.

```
dfRide %>%
  filter (start_station_id != "") %>%
  select (start_station_id, start_lat) %>%
  group_by(start_station_id, as.character(start_lat)) %>%
  summarise (n1 = n())
```

```
## # A tibble: 1,082,174 × 3
## # Groups:   start_station_id [1,610]
##    start_station_id `as.character(start_lat)`    n1
##    <chr>            <chr>                     <int>
##  1 021320           41.889609814                  1
##  2 021320           41.889626503                  1
##  3 021320           41.889638                     1
##  4 021320           41.889638901                  1
##  5 021320           41.8896485                    1
##  6 021320           41.889652014                  1
##  7 021320           41.88966012                   1
##  8 021320           41.889662266                  1
##  9 021320           41.8896626666667              1
## 10 021320           41.889664                     1
## # ℹ 1,082,164 more rows
```

Transform

We transform started_at and ended_at into datetime type as they are characters.

```
dfRide <- mutate(dfRide,
        started_at = as_datetime(dfRide$started_at),
        ended_at = as_datetime(dfRide$ended_at))
```

We compute the ride_length for the duration of the ride in seconds.

```
dfRide <- mutate(dfRide, ride_length = dfRide$ended_at - dfRide$started_at )
```

We add day_of_the_week (Mon, Tue, …) and the ride_month (Jun, Jul, …)

```
dfRide <- mutate(dfRide, day_of_week = wday(started_at, label = TRUE))
dfRide <- mutate(dfRide, ride_month = month(started_at, label = TRUE))
```

```
head(dfRide)
```

```
##            ride_id rideable_type          started_at            ended_at
## 1 6F1682AC40EB6F71 electric_bike 2023-06-05 13:34:12 2023-06-05 14:31:56
## 2 622A1686D64948EB electric_bike 2023-06-05 01:30:22 2023-06-05 01:33:06
## 3 3C88859D926253B4 electric_bike 2023-06-20 18:15:49 2023-06-20 18:32:05
## 4 EAD8A5E0259DEC88 electric_bike 2023-06-19 14:56:00 2023-06-19 15:00:35
## 5 5A36F21930D6A55C electric_bike 2023-06-19 15:03:34 2023-06-19 15:07:16
## 6 CF682EA7D0F961DB electric_bike 2023-06-09 21:30:25 2023-06-09 21:49:52
##   start_station_name start_station_id end_station_name end_station_id start_lat
## 1                                                                          41.91
## 2                                                                          41.94
## 3                                                                          41.95
## 4                                                                          41.99
## 5                                                                          41.98
## 6                                                                          41.99
##   start_lng end_lat end_lng member_casual ride_length day_of_week ride_month
## 1    -87.69   41.91  -87.70        member   3464 secs         Mon        Jun
## 2    -87.65   41.94  -87.65        member    164 secs         Mon        Jun
## 3    -87.68   41.92  -87.63        member    976 secs         Tue        Jun
## 4    -87.65   41.98  -87.66        member    275 secs         Mon        Jun
## 5    -87.66   41.99  -87.65        member    222 secs         Mon        Jun
## 6    -87.68   41.94  -87.65        member   1167 secs         Fri        Jun
```

Cleaning

We start with 5,743,278 observations

Removal of negative duration

441 observations have a negative duration (0.77%)

```
dfRide <- dfRide %>% filter(ended_at >= started_at)
```

A lot of rides do not last more than 30 seconds. A minimum duration should be considered as a realistic duration.

Removal of unknown starting or unknown ending station

1428013 observations would be filtered out if both ending and starting points are unknown, which represent 24.86% of observations. 994721 observations would be filtered out if we filter out if one of the ending or starting points is unknown, which represents 17.32% of observations. In both cases, that is a lot. In real life, we would go back to stakeholders to understand the business rules that cause all the inconsistencies.

Removal of observations without any ending geographical location.

7684 observations will be filtered out if we remove observations with any geographical locations, which represents 0.13%.

```
dfRide <- dfRide %>% filter(!is.na(end_lat) & !is.na(end_lng))
```

We also remove 3 observations where end_lat and end_lng equals 0.

```
dfRide <- dfRide %>% filter(end_lng != 0 & end_lat != 0)
```

Analyze

Calculation

A few calculations about ride_lenght are performed to get a better sense of the data.

```
Modes <- function(x) {
    ux <- unique(x)
    tab <- tabulate(match(x, ux))
    ux[tab == max(tab)]
    }
dfRide %>%
    summarise (mean_ride_length = mean(ride_length),
    max_ride_length = max(ride_length),
    min_ride_length = min(ride_length),
    mode_week_day = Modes(day_of_week))
```

```
##   mean_ride_length max_ride_length min_ride_length mode_week_day
## 1    928.3626 secs     669136 secs          0 secs           Sat
```

Regarding ride_length, a concern can be raised as the minimum is 0 second and the maximum is 669136 seconds (equivalent to 7 days 17 hours 52 minutes and 16 seconds)

Let’s check the average ride_length for members and casual riders.

```
dfRide %>%
    group_by(member_casual ) %>%
    summarise (mean_ride_length = mean(ride_length))
```

```
## # A tibble: 2 × 2
##   member_casual mean_ride_length
##   <chr>         <drtn>
## 1 casual        1271.3517 secs
## 2 member         738.5573 secs
```

Casual users tend to ride longer than members which is very interesting to learn.

Let’s check the average ride_length per day of the week.

```
dfRide %>%
    group_by(day_of_week) %>%
    summarise (mean_ride_length = mean(ride_length))
```

```
## # A tibble: 7 × 2
##   day_of_week mean_ride_length
##   <ord>       <drtn>
## 1 Sun         1118.1475 secs
## 2 Mon          882.5106 secs
## 3 Tue          831.8758 secs
## 4 Wed          820.9508 secs
## 5 Thu          819.3179 secs
## 6 Fri          911.8409 secs
## 7 Sat         1112.9017 secs
```

The rides are longer during the weekend, which is not surprising.

Let’s now check the number of rides for users by day_of_week.

```
dfRide %>% count(day_of_week)
```

```
##   day_of_week      n
## 1         Sun 739625
## 2         Mon 748727
## 3         Tue 799463
## 4         Wed 831180
## 5         Thu 857045
## 6         Fri 849449
## 7         Sat 909661
```

Surprisingly, many users cycle between Wednesday and Friday. Saturday is the day with the most users and Sunday is the day with the fewest users.

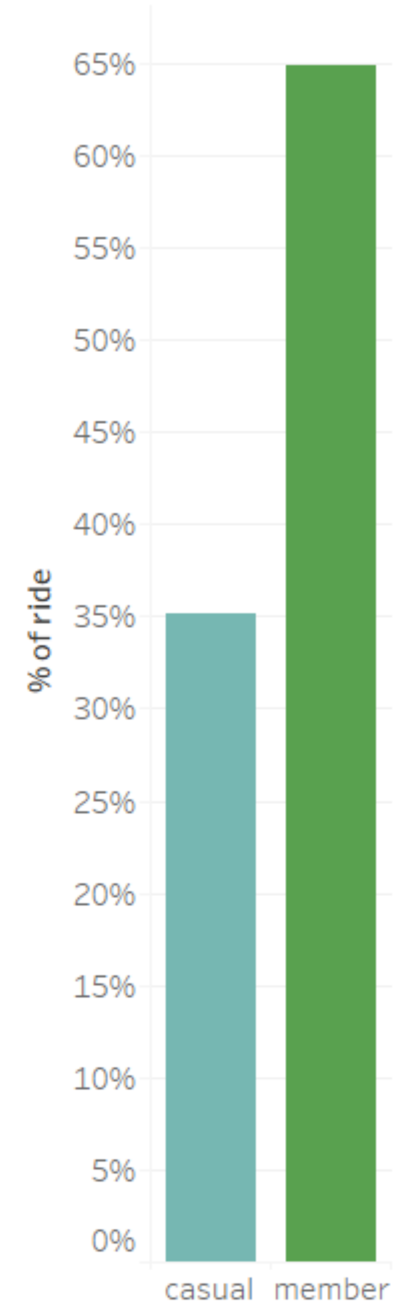
Output CSV file

After being merged, cleaned and transformed, the dataset will be shared as a CSV file in the following analyses.

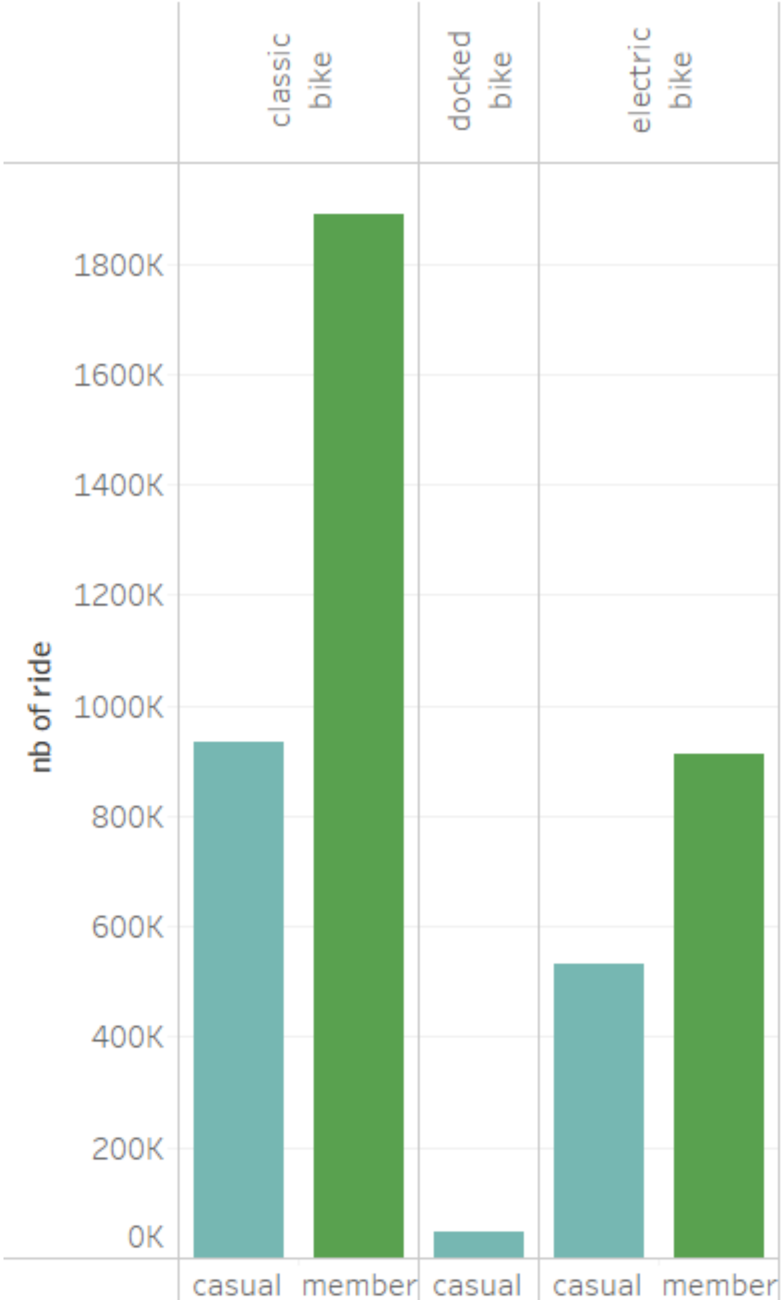
```
write.csv(dfRide, "cyclistic-tripdata.csv")
```

The final CSV file will be used as an entry for the next analysis steps with Tableau.

Share

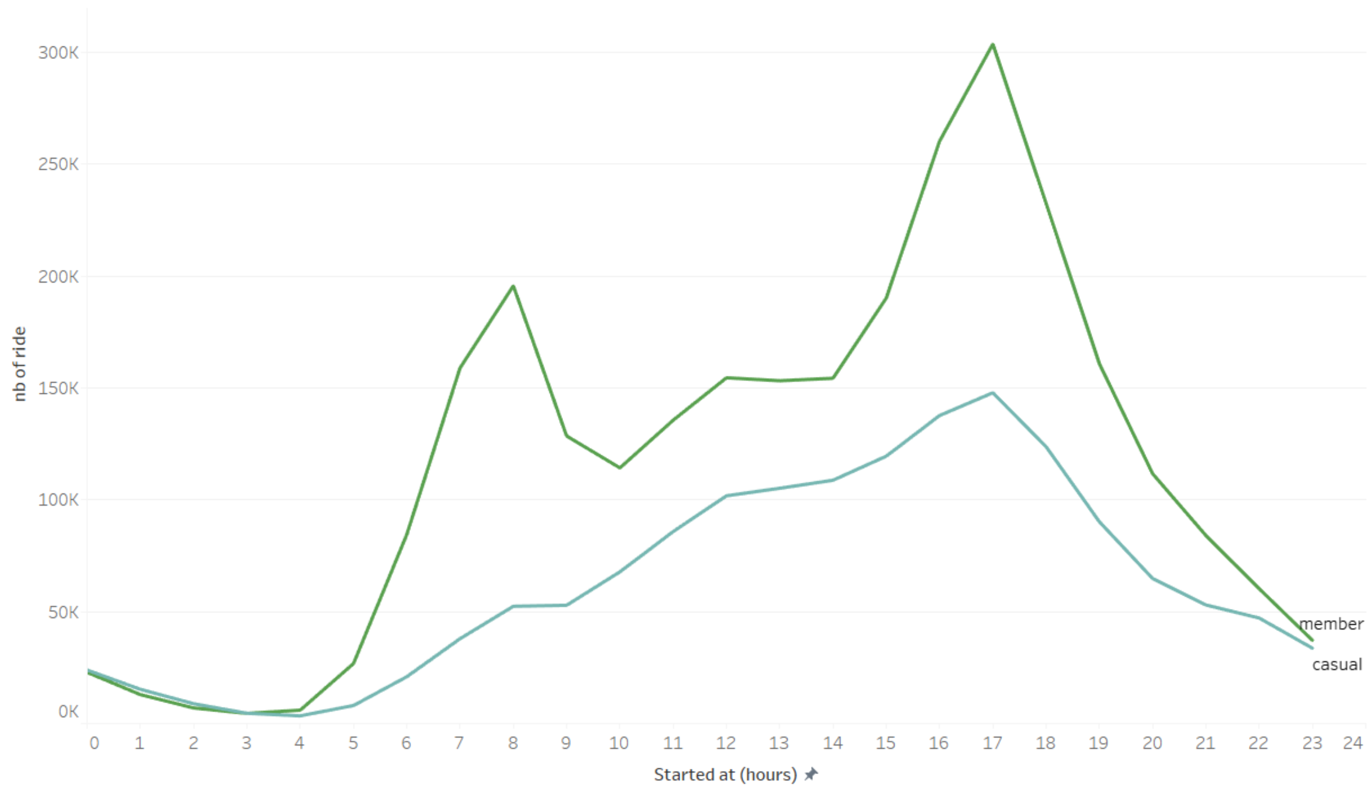


A majority of users are **members**(65 %). A small portion represents **casual** users(35 %).

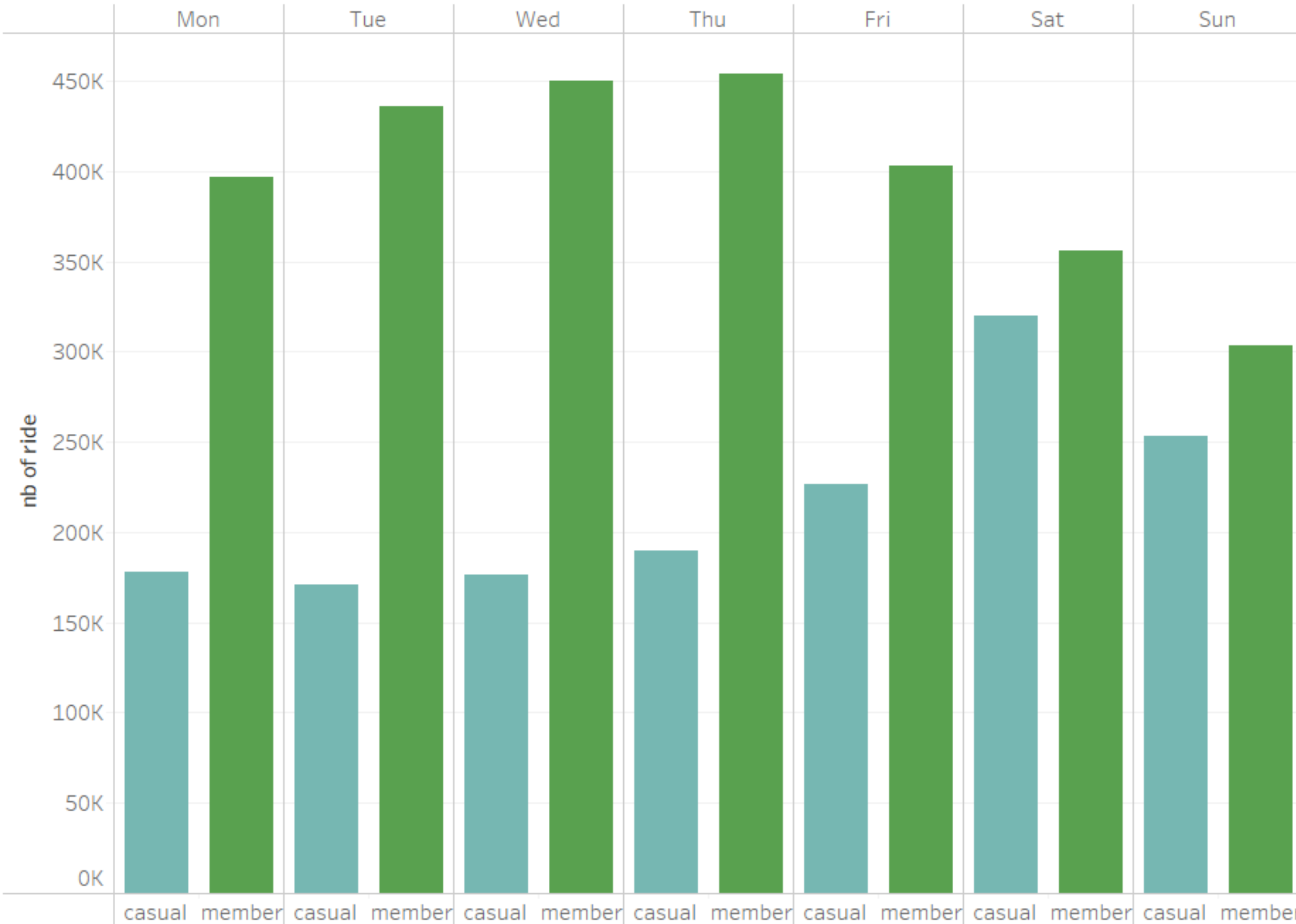


Docked bikes are never used, except a little bit by **casual** users (4%), probably for the simple pleasure of trying an original means of transport as part of a visit.

The preference remains for classic bikes (71%) whatever the type of user, compared to the electronic bikes (25%).



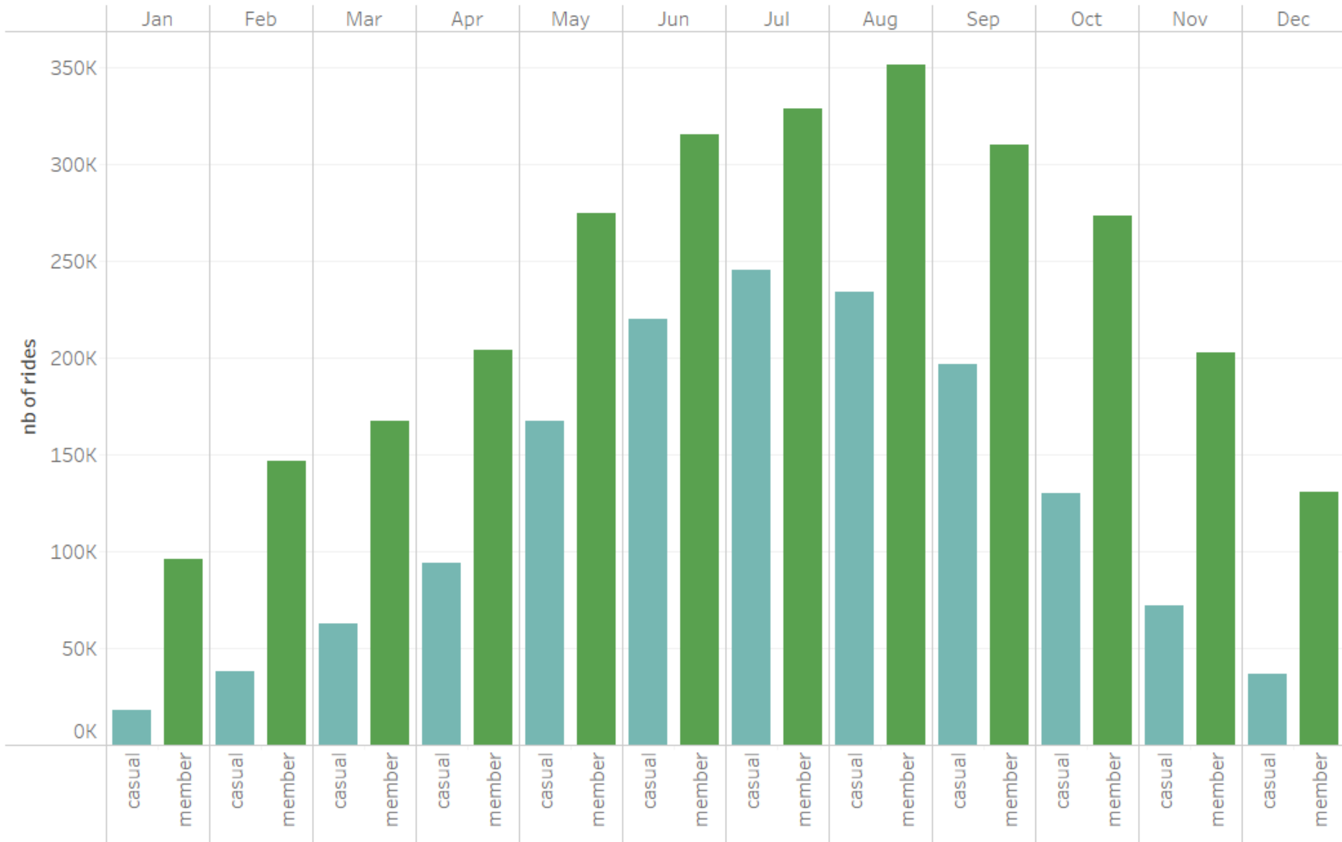
We can see spikes in bike usage during peak hours for **members** users. Their use is linked to the journey home to work (7h to 8h and 17h to 18h). As for **casual** users, their number increases continuously from hour to hour until reaching a peak around 17 h., at the end of the day. The analysis was carried out on the start time of the journey. However, a similar result is observed at the same time if we look at the end time of the journey.



Members use the bikes during the week from Monday to Friday with a peak on Thursday. Then it drops on the weekend. Conversely, **casual** users use bicycles mainly on weekends, with a peak on Saturdays.This seems to be stable during the week. Friday is the day when trends start to reverse.

When do they use it ?

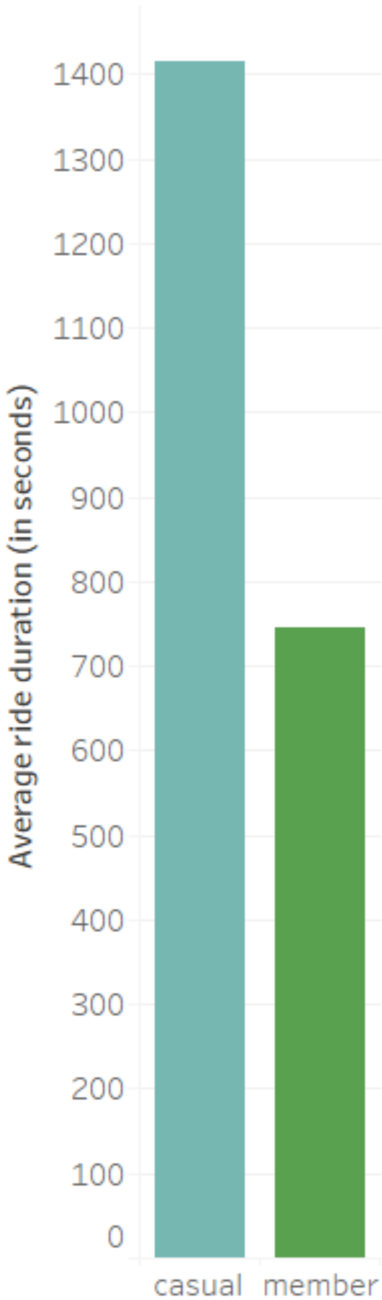
In the last graph, we distinguish users by their use during a day or a week and see that they use Cyclistic's bikes for different usages.
However, how is their behaviour for longer terms like year?



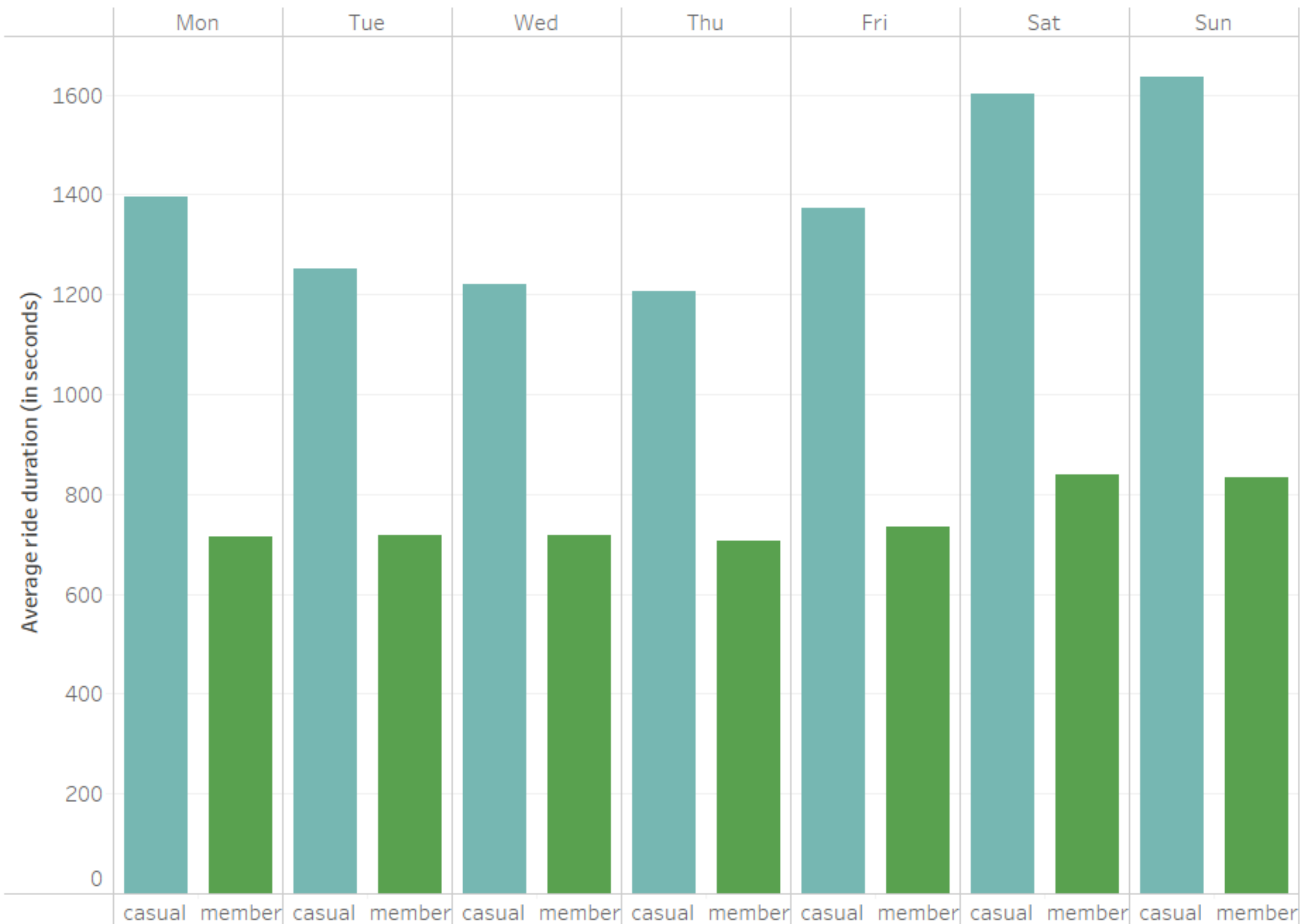
Both types, **members** or **casual** riders, use bicycles during periods of good weather, with no rain, warm and sunny, with a peak in August. Conversely, they use bicycles very little during the cold or rainy months, with the lowest level reached in January.

How do they use it?

We start to get an idea of who Cyclistic's users are, how each type of user uses the service and when. Now let's look at how and why each of them use these bikes.

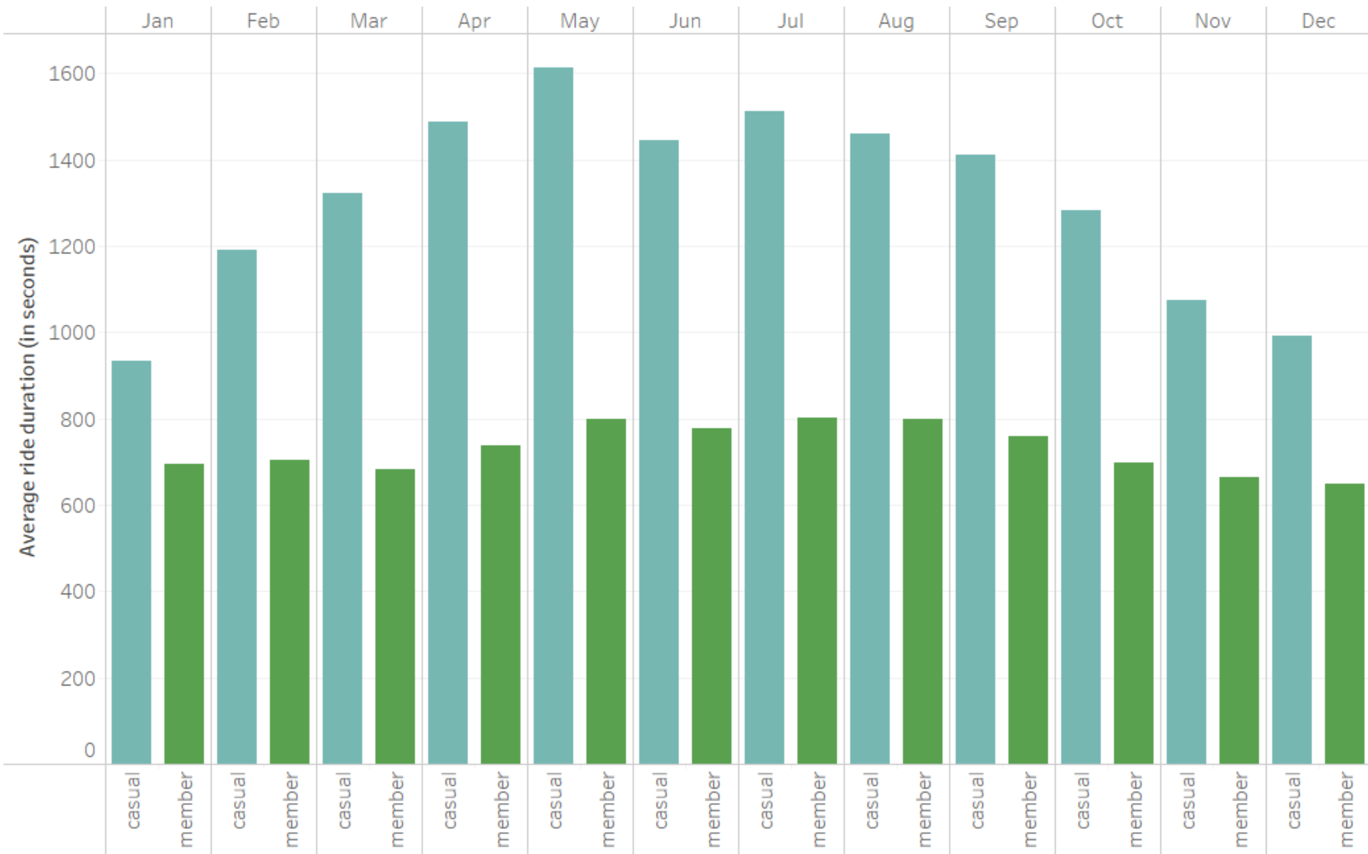


Members use the bikes the least amount of time on average. **Casuals** spend more time on their bikes. **Members** probably have to use bikes to get from point A to point B when commuting, **casual** people use them more for leisure.



Member users use the bikes for the same amount of time on average over the entire week. They use the bikes slightly longer during the weekend. However, we have seen that fewer **member** users use bikes at the end of the week. This should be a small portion that needs to use the bikes for longer periods of time over the weekend.

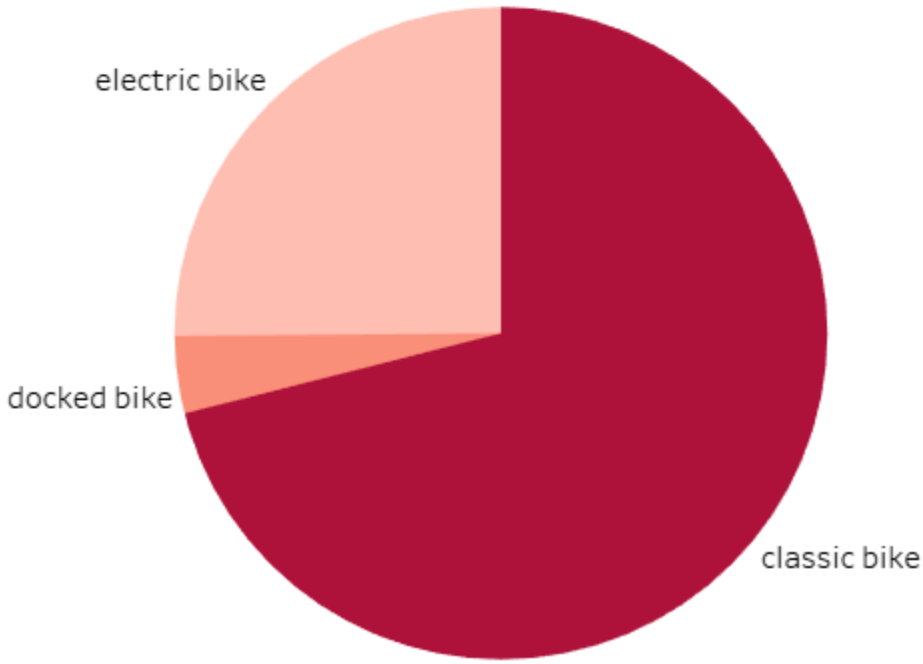
Casual users, even if more in the minority than **members**, use the bikes longer, especially on weekends.



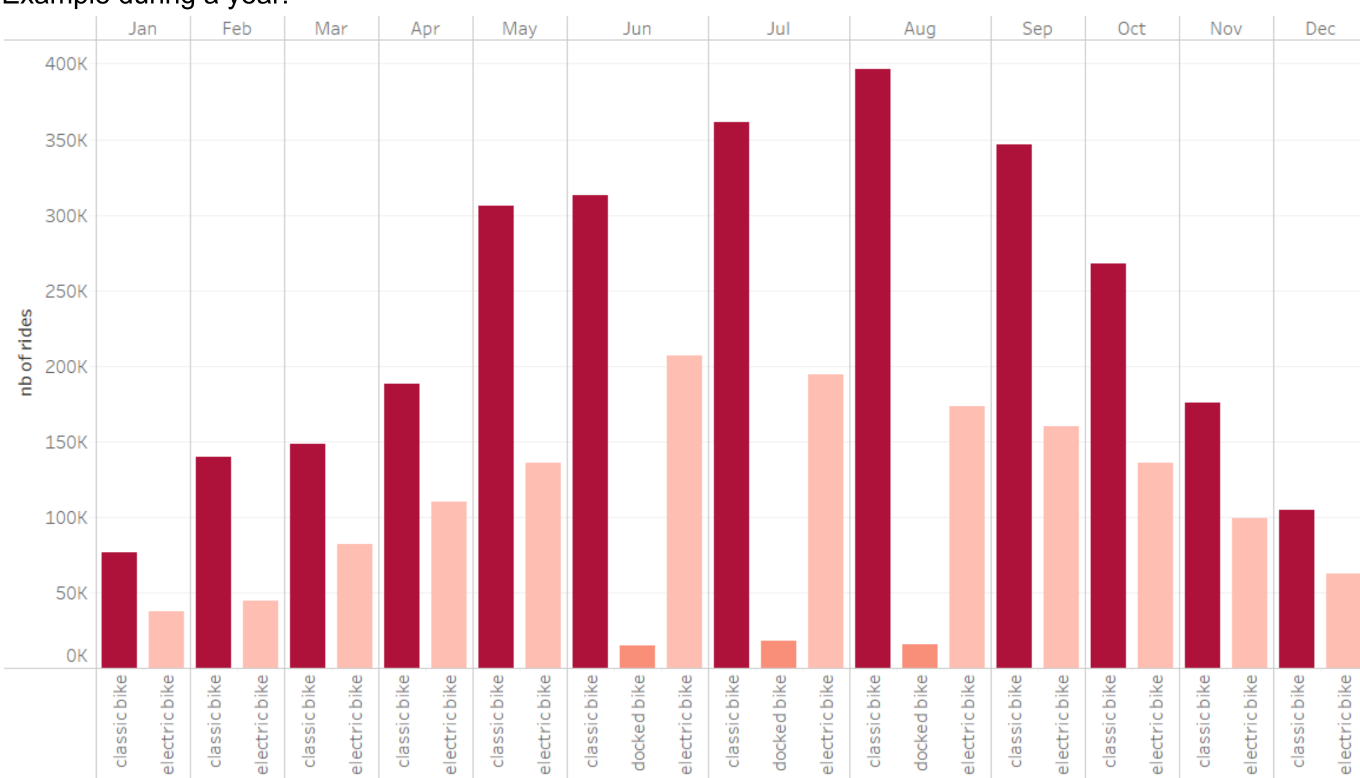
The time **members** use the bikes is fairly constant over the course of a year, even if they use the bikes more in summer than in winter.
Casual users use the bikes longer on sunny days. The peak of time used is reached in May, even if the number of **casual** users is at its peak in August.

What do they use?

As seen previously, **classic bikes** are widely used (71%) compared to **electric bikes** (25%). **Docked bikes** are never used, except a little bit by **casual** users only between June and August(4%), probably for the simple pleasure of trying an original means of transport as part of a visit.
On every graph that I made to find any insight (eg. by ride length, by average duration, by hours, during a week, during a year, …), **classic bikes** are the main options.



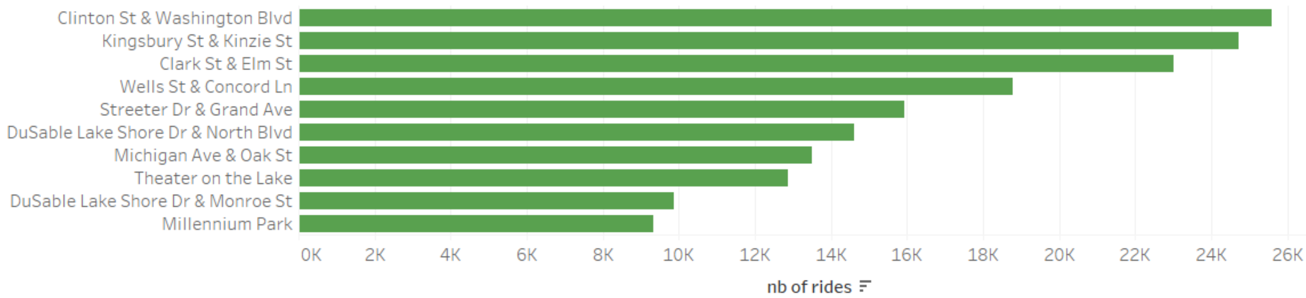
Example during a year:



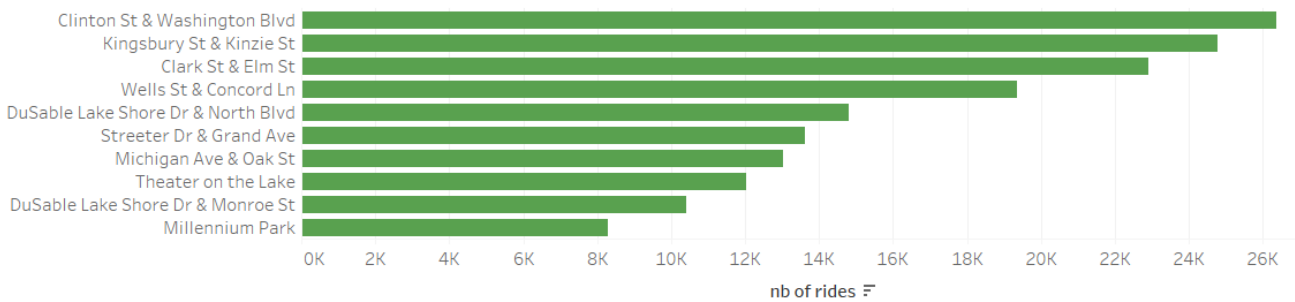
Where do they use it the most?

Here are the top 10 stations used as a start and end for all **members**.

Top start station

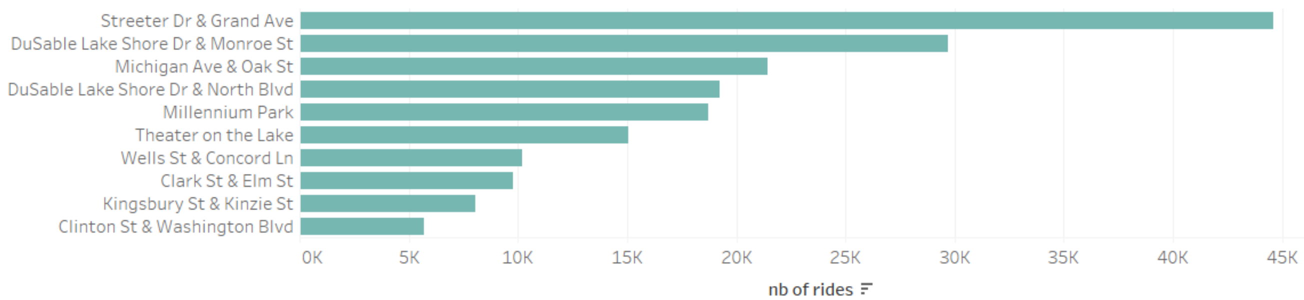


Top end station

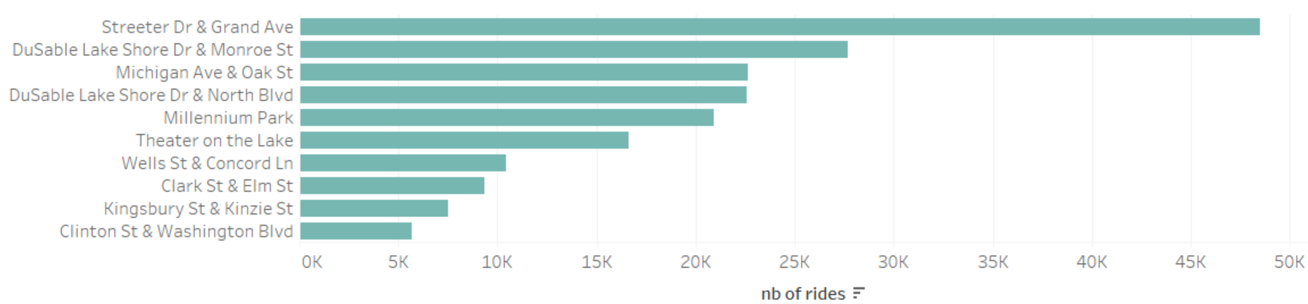


Here are the top 10 stations used as a start and end for all **casual users**.

Top start station

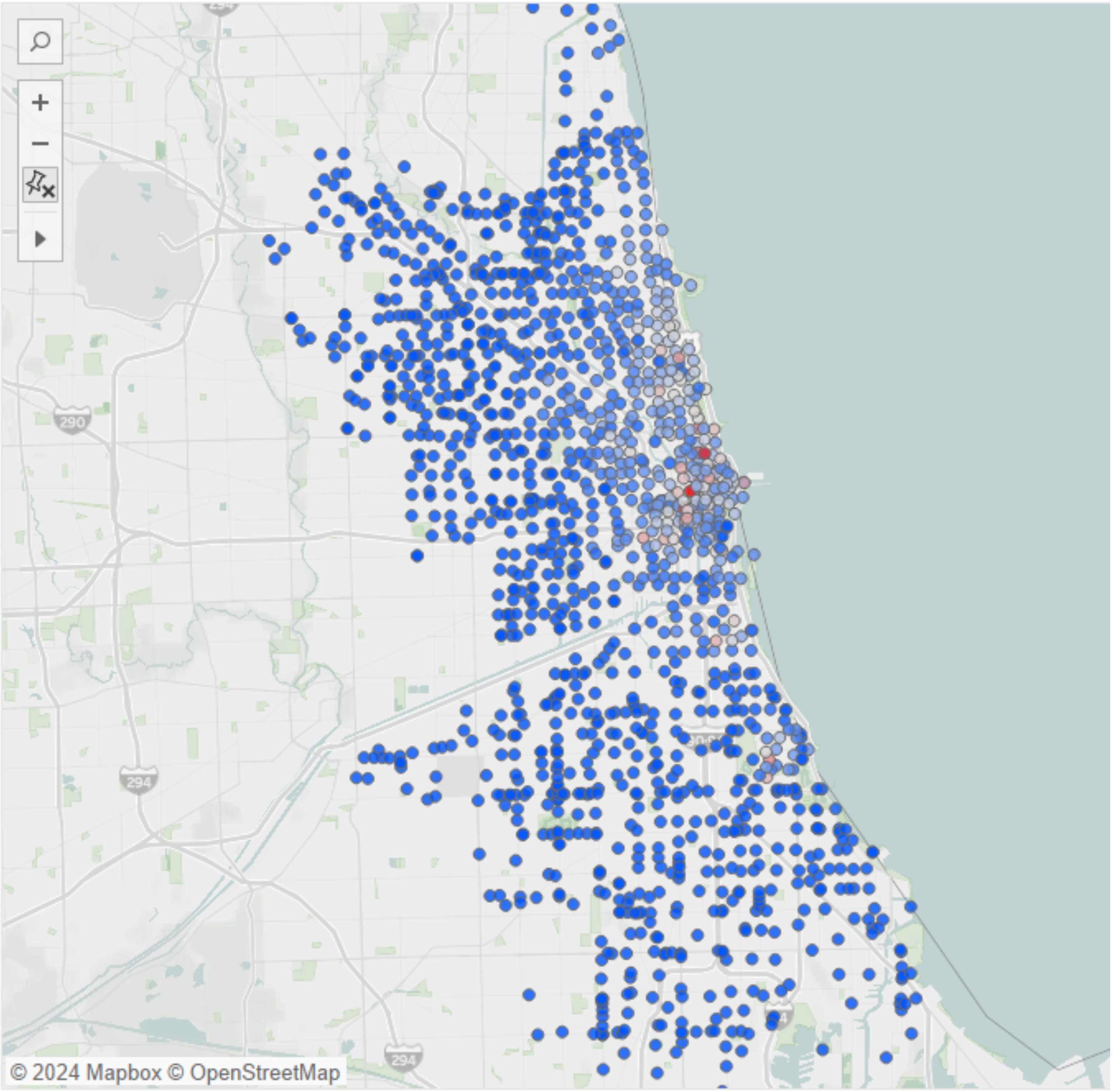


Top end station

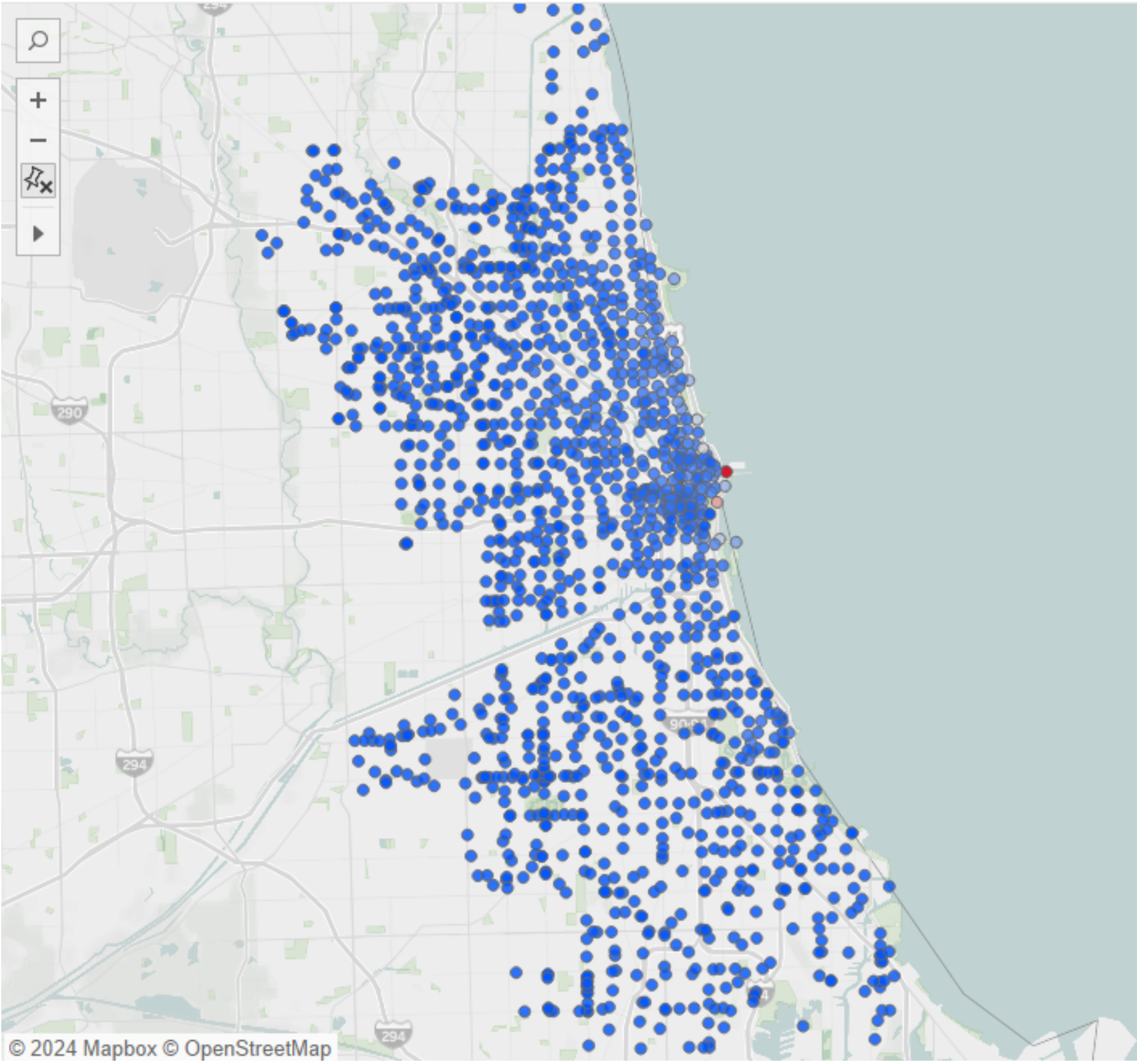


For each type of user, in both cases, we can find the same station as the start and end of rides.
Since I'mnot familiar with the names, I display the city maps for those you may know.

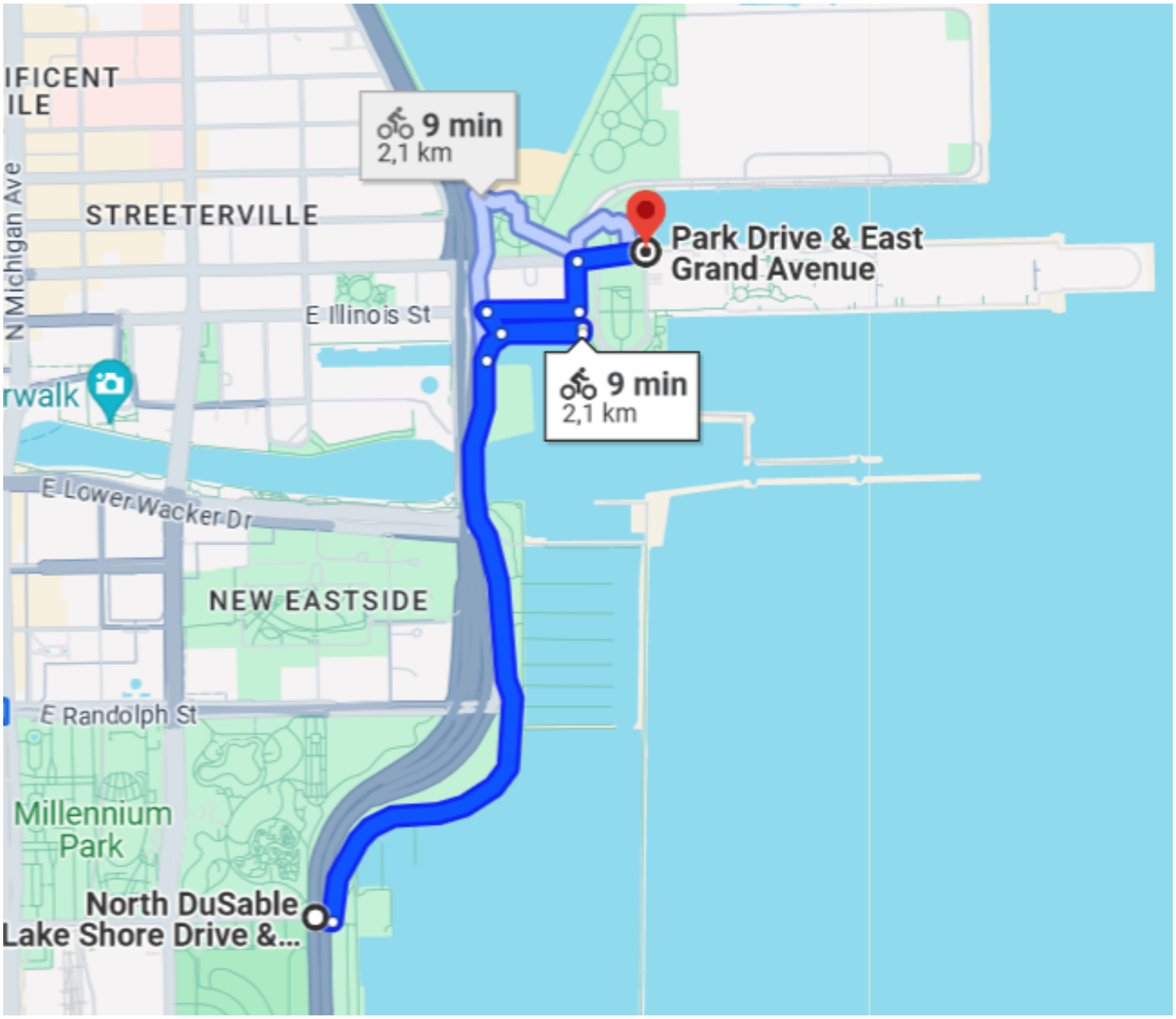
Stations used by members



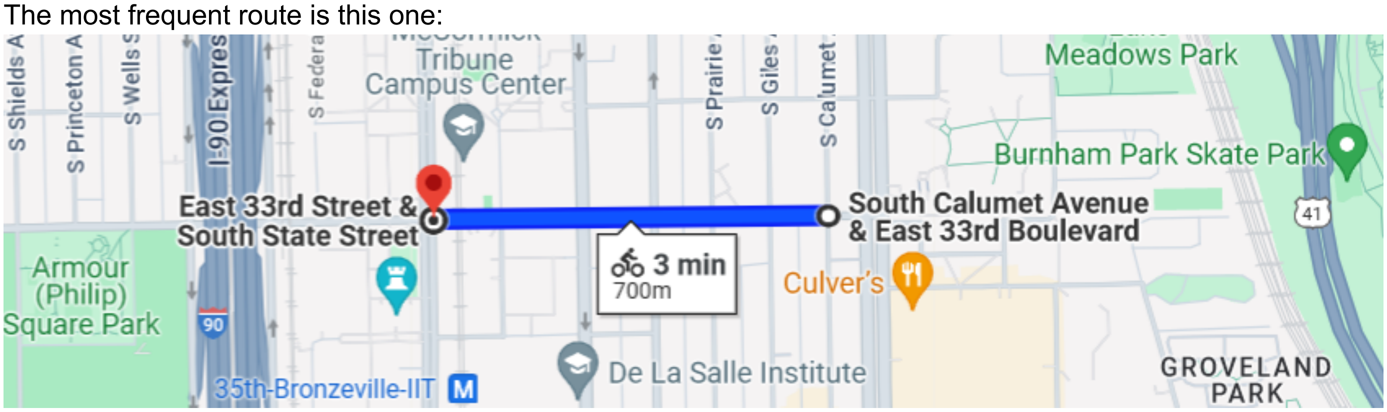
Stations used by casual users



The most used stations are located on the waterfront, in the business district, and in the tourist center.
Casual users go to E Grand Ave & N Streeter Dr which is the center of leisure activities.
The most frequent route is scenic or it can be a round trip from and to these stations:



Members use station in a large area with buildings and several stations for different means of transport
The most frequent route is this one:



Act

The analysis shows how annual members and casual riders use Cyclistic bikes differently.
To summarize the previous analysis in broad terms, members use bicycles to commute, and or for pleasure during the weekend. It is the most numerous users who travel on smaller journeys. Casual users are mostly users who use bikes for tourism.They are fewer in number but use the bikes longer.

We should focus on the casual users that could be members. A lot of casual users are probably not interested in a membershing as they will use the bikes occasionally for some days. However, some of them can be locals that use bikes during the evening or during the weekend. We may focus on them to become members.

Three main recommendations that we can make could be:

- We could offer them **a new type of member account** for evenings and weekends with some advantages that should be between those who are members, the “premiums”, and those who are casual users. For example, we should offer advantageous prices for their favorite routes during evening and weekend journeys.
- The marketing campaign should be carried out **during the spring and summer, or even until the beginning of autumn**, because this is when these users are circulating.
- The **application** should suggest the winnings won if they were passed to the member account. Displays should be made **around the stations most used** by casual users.