# Cyclistic Case Study

Anthony R. Poggioli

2023-12-12

## Introduction

This case study was completed as part of the Google Data Analytics Professional Certificate program offered by Google through Coursera. The data is available publicly from Motivate International Inc. under this licence agreement. It is used with permission by Google in their Data Analytics Capstone.

I have taken the scenario and business question from Case Study 1 (as of December, 2023) of the Google Data Analytics Capstone Course. However, the analysis is unguided and entirely my own.

### The Scenario

The director of marketing at Cyclistic, a Chicago-based bicycle-share company, believes that future growth of the company hinges on the conversion of casual customers – who purchase one-time single-ride or full-day passes – into Cyclistic annual members. To this end, the marketing analyst team has been tasked with determining what differentiates casual customers from annual members. Specifically, I am tasked with answering the following question: How do annual members and casual riders use Cyclistic bikes differently?

### The Business Task

This analysis aims to use individual trip-based data collected in-house by Cyclistic to determine how use patterns differ between casual customers and annual members. This is in service of the design of a marketing campaign focused on the conversion of casual customers into annual members.

### A Note on my Choice of R

I have chosen to conduct this analysis in R in order to better learn the language. I have significant past programming experience in Python (my language of choice) and MATLAB, and some experience in C++ and Java, but I first used R in the Google Data Analytics Certificate program. This has been a useful and frustrating exercise. This analysis took me about eight days, but much of that was figuring out how to clean data in R – which really comes down to how to fruitfully interact with data frames in R. It is likely that some or even much of this code is clunky and does not represent best practices.

## Into the Tidyverse

We begin by loading the packages we will use in the subsequent data exploration, data cleaning, and data analysis. The most important of these are the core `tidyverse` packages. We also load the `hms` library for dealing with times in `lubridate` (itself a core `tidyverse` package) and the `ggmap` package, which we will use to make map visualizations with Google Maps.

```r
library(tidyverse) # core tidyverse packages
library(hms) # for dealing with hms-time in lubridate
library(ggmap) # make maps using Google Maps
```

## The Data

Our first task is to familiarize ourselves with the available data. (See the link in the Introduction above to view and download this publicly available data.)

Looking through the data, we find that it is inconsistently named and formatted. Additionally, certain attributes are recorded for some years and not for others. Accordingly, we need to acquaint ourselves with how the data's structure, file naming conventions, and attributes change over time. A few preliminary observations:

- The data structure appears to have been standardized from 2020-04 up to most recent data (2023-10): it is contained in a single table containing 13 attributes, and the data tables are segregated by month, rather than quarter.
- From the first quarter of 2018 through the first quarter of 2020, data for each quarter is stored in a single table, each containing what appear to be the same 11 attributes. However, the attribute names are inconsistent; I assume that '01 - Rental Details Rental ID' and 'trip_id' refer to the same unique identifier for a given trip and therefore function as primary key.
- Data table attributes change during different time periods.
- I converted the station table for first two quarters of 2014 (2014-Q1Q2) from xlsx (Excel) to csv format for consistency with other data files.
- Before the first quarter of 2018, data is stored in two separate tables. A trip table with a primary key given by the ID of individual trips, and a station table with a primary key given by station IDs. The apparent exception is the third and fourth quarters of 2015, for which there are trip tables, but there is no explicitly associated station table.
- Data tables are sometimes segregated by quarter, and sometimes by month.
- Some of the station tables contain the attribute 'landmark', which contains integer values up to three digits, similar to the station IDs. However, this attribute is not defined in any of the READMEs, and it is unclear what it represents.

### Station Data for the Last Two Quarters of 2015

The folder 'Divvy_Trips_2015_Q3Q4' contains four trips data tables for July, August, September, and the fourth quarter of 2015 but no station data table, though every other trips data table has an explicitly associated station data table until the first quarter of 2018. It's possible that all relevant station information (i.e., information for all stations appearing in trips from 2015) is contained in the file './Divvy_Trips_2015_Q1Q2/Divvy_Stations_2015.csv'; however, it apears possible that not all station IDs are contained in this table:

```r
df_stations_2015_Q1Q2 <- read_csv("./data/Divvy_Trips_2015_Q1Q2/Divvy_Stations_2015.csv")
```

```r
df_stations_2015_Q1Q2 %>%
  summarize(minimum_station_id = min(id), maximum_station_id = max(id),
            number_of_unique_stations = length(unique(id)), number_of_entries = length(id),
            number_of_missing_ids = maximum_station_id - number_of_unique_stations)
```

```
## # A tibble: 1 x 5
##   minimum_station_id maximum_station_id number_of_unique_sta~1 number_of_entries
##                <dbl>              <dbl>                  <int>             <int>
## 1                  2                511                    474               474
## # i abbreviated name: 1: number_of_unique_stations
## # i 1 more variable: number_of_missing_ids <dbl>
```

We see that the minimum station ID is two, the maximum is 511, and the station ID attribute contains 474 entries, all of which are unique. Therefore, there are 37 potential station IDs between 1 and 511 (inclusive) that are not recorded. These are:

```r
max_station_id <- max(df_stations_2015_Q1Q2$id)
col_range <- data.frame(col = 1:max_station_id)
```

```r
missing_ids <- col_range %>%
  filter(!(col_range$col %in% df_stations_2015_Q1Q2$id)) %>%
  print()
```

```
##    col
## 1    1
## 2    8
## 3   10
## 4   63
## 5   64
## 6   65
## 7   70
## 8   78
## 9   79
## 10  82
## 11  83
## 12 104
## 13 105
## 14 139
## 15 151
## 16 155
## 17 187
## 18 189
## 19 221
## 20 235
## 21 266
## 22 269
## 23 357
## 24 358
## 25 360
## 26 361
## 27 362
## 28 363
## 29 371
## 30 379
## 31 380
## 32 387
## 33 389
## 34 394
## 35 404
## 36 405
## 37 473
```

This single-column data frame contains the potential station IDs that are not contained in the station table. The questions are 1) are there any potential station IDs from this column that appear in any of the trips data tables from 2015, and 2) are there any station IDs greater than 511 that appear in the trips data tables from 2015?

We check each trips table from 2015 to determine if the maximum station ID is greater than 511, and if there are any station IDs in the 'from_station_id' or 'to_station_id' columns that are not contained in the stations table – or, equivalently, if there are any station IDs from the trips tables that are contained in the above-defined `missing_ids` data frame.

```r
df_trips_2015_Q1 <- read_csv("./data/Divvy_Trips_2015_Q1Q2/Divvy_Trips_2015-Q1.csv")
df_trips_2015_Q2 <- read_csv("./data/Divvy_Trips_2015_Q1Q2/Divvy_Trips_2015-Q2.csv")
```

```
df_trips_2015_07 <- read_csv("./data/Divvy_Trips_2015_Q3Q4/Divvy_Trips_2015_07.csv")
df_trips_2015_08 <- read_csv("./data/Divvy_Trips_2015_Q3Q4/Divvy_Trips_2015_08.csv")
df_trips_2015_09 <- read_csv("./data/Divvy_Trips_2015_Q3Q4/Divvy_Trips_2015_09.csv")
df_trips_2015_Q4 <- read_csv("./data/Divvy_Trips_2015_Q3Q4/Divvy_Trips_2015_Q4.csv")
```

We begin by constructing a data frame containing all unique station IDs contained in either the 'from_station_id' or 'to_station_id' columns in the trips tables from 2015:

```
unique_station_ids_2015 <-
                  data.frame(station_id =
                  unique(c(unique(df_trips_2015_Q1$from_station_id),
                           unique(df_trips_2015_Q1$to_station_id),
                           unique(df_trips_2015_Q2$from_station_id),
                           unique(df_trips_2015_Q2$to_station_id),
                           unique(df_trips_2015_07$from_station_id),
                           unique(df_trips_2015_07$to_station_id),
                           unique(df_trips_2015_08$from_station_id),
                           unique(df_trips_2015_08$to_station_id),
                           unique(df_trips_2015_09$from_station_id),
                           unique(df_trips_2015_09$to_station_id),
                           unique(df_trips_2015_Q4$from_station_id),
                           unique(df_trips_2015_Q4$to_station_id))))
unique_station_ids_2015 <- filter(unique_station_ids_2015,!is.na(unique_station_ids_2015))
```

```
unique_station_ids_2015 %>%
  summarize(minimum_station_id = min(station_id),
            maximum_station_id = max(station_id),
            number_of_unique_stations = length(unique(station_id)),
            number_of_entries = length(station_id),
            number_of_missing_ids = maximum_station_id - number_of_unique_stations)
```

```
##   minimum_station_id maximum_station_id number_of_unique_stations
## 1                  2                511                       475
##   number_of_entries number_of_missing_ids
## 1               475                    36
```

We see that the minimum and maximum station IDs are the same as they were for the stations table contained in the 'Divvy_Trips_2015_Q1Q2' folder – two and 511, respectively; however, there are 475 unique station IDs, indicating that there is at least one station that is not represented in the stations table. Let's determine which station ID(s) are contained in the trips tables but not the stations table:

```
missing_ids_from_trips <- unique_station_ids_2015 %>%
  filter(!(unique_station_ids_2015$station_id %in% df_stations_2015_Q1Q2$id)) %>%
  print()
```

```
##   station_id
## 1        394
```

There is a single station contained in the trips tables that is not recorded in the stations table from the first two quarters of 2015. We conclude that the station data for 2015 is incomplete. If we assume that station IDs are never reassigned, it is possible that the identity, location, and other attributes associated with station 394 are contained in station data from previous years.

Overall, we conclude that this data can be made sense of, but the data storage practices are non-standardized and *ad hoc*.

## Data Cleaning

For simplicity, we will focus on the data after standardization – i.e., from 2020-04 to 2023-10. This represents 3.5 years of data, which I anticipate will be sufficient to extract statistically significant inferences about the distinctions between casual riders and annual members. It is also the most recent, and therefore most relevant, data.

We begin by loading the trip data files from the above-mentioned period into a single data frame using a `for` loop:

```r
df_trips <- data.frame()
for (i in 2020:2023) {
  for (j in 1:12) {
    if (j < 10) {
      prefix <- paste(i,j,sep="0")
    } else {
      prefix <- paste(i,j,sep="")
    }
    if (i == 2020 & j < 4) {
      next
    } else if (i == 2023 & j > 10) {
      break
    } else {
      filename <- paste(prefix,"divvy-tripdata.csv",sep="-")
      df_trips <- rbind(df_trips,read_csv(paste("./data/",filename,sep="")))
    }
  }
}
```

```r
df_trips %>%
  summarize(length(rideable_type))
```

```
## # A tibble: 1 x 1
##   `length(rideable_type)`
##                     <int>
## 1                19510862
```

This produces a single data frame with over 19.5 million observations.

We will look for differences in the statistical distributions of starting station, ending station, trip duration, day of week of trip, time of day of trip, and rideable type between casual customers and annual members to determine if any of these metrics effectively discriminate between these groups.

Note that the attribute 'rideable_type' is not defined in any README, and it is unclear to what it refers.

```r
print(unique(df_trips$rideable_type))
```

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

It takes one of three values – 'docked_bike', 'electric_bike', or 'classic_bike' – though since all bikes are obtained from a dock, it is unclear what 'docked_bike' refers to in particular.

We start our cleaning process by checking the data types:

```r
str(df_trips)
```

```
## spc_tbl_ [19,510,862 x 13] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ ride_id           : chr [1:19510862] "A847FADBBC638E45" "5405B80E996FF60D" "5DD24A79A4E006F4" "2A
##  $ rideable_type     : chr [1:19510862] "docked_bike" "docked_bike" "docked_bike" "docked_bike" ...
##  $ started_at        : POSIXct[1:19510862], format: "2020-04-26 17:45:14" "2020-04-17 17:08:54" ...
##  $ ended_at          : POSIXct[1:19510862], format: "2020-04-26 18:12:03" "2020-04-17 17:17:03" ...
##  $ start_station_name: chr [1:19510862] "Eckhart Park" "Drake Ave & Fullerton Ave" "McClurg Ct & Eri
##  $ start_station_id  : chr [1:19510862] "86" "503" "142" "216" ...
##  $ end_station_name  : chr [1:19510862] "Lincoln Ave & Diversey Pkwy" "Kosciuszko Park" "Indiana Ave
##  $ end_station_id    : chr [1:19510862] "152" "499" "255" "657" ...
##  $ start_lat         : num [1:19510862] 41.9 41.9 41.9 41.9 41.9 ...
##  $ start_lng         : num [1:19510862] -87.7 -87.7 -87.6 -87.7 -87.6 ...
##  $ end_lat           : num [1:19510862] 41.9 41.9 41.9 41.9 42 ...
##  $ end_lng           : num [1:19510862] -87.7 -87.7 -87.6 -87.7 -87.7 ...
##  $ member_casual     : chr [1:19510862] "member" "member" "member" "member" ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   ride_id = col_character(),
##   ..   rideable_type = col_character(),
##   ..   started_at = col_datetime(format = ""),
##   ..   ended_at = col_datetime(format = ""),
##   ..   start_station_name = col_character(),
##   ..   start_station_id = col_double(),
##   ..   end_station_name = col_character(),
##   ..   end_station_id = col_double(),
##   ..   start_lat = col_double(),
##   ..   start_lng = col_double(),
##   ..   end_lat = col_double(),
##   ..   end_lng = col_double(),
##   ..   member_casual = col_character()
##   .. )
##  - attr(*, "problems")=<externalptr>
```

The starting and ending station IDs need to be converted from `chr` to `int` – first, let's check what happens when we try to make this conversion:

```r
# number of nulls in start_station_id
is_null_id_before <- sum(is.na(df_trips$start_station_id))
# number of nulls after conversion
is_null_id_after <- sum(is.na(as.integer(df_trips$start_station_id)))
# number of nulls introduced by conversion
becomes_null_id <- is_null_id_after - is_null_id_before
cat("There were initially",is_null_id_before/1e6,"million null values;
    conversion to int introduced",becomes_null_id/1e6,"million additional null values.")
```

```
## There were initially 2.404287 million null values;
##     conversion to int introduced 7.782181 million additional null values.
```

Over 2.4 million starting station IDs are null in our data; nearly another 7.8 million are improperly formatted and are cast to null when we try to convert the starting station IDs to integers. Similar results hold for the ending station IDs.

Let's look at the starting station IDs that cannot be converted to integers:

```r
df_trips %>%
  filter(!is.na(start_station_id) & is.na(as.integer(start_station_id))) %>%
  select(start_station_id,start_station_name) %>%
  as_tibble()
```

```
## # A tibble: 7,782,181 x 2
##    start_station_id start_station_name
##    <chr>            <chr>
##   1 TA1309000006     Larrabee St & Armitage Ave
##   2 TA1309000006     Larrabee St & Armitage Ave
##   3 KA1503000043     Kingsbury St & Kinzie St
##   4 TA1309000014     Clark St & Leland Ave
##   5 TA1309000014     Clark St & Leland Ave
##   6 TA1305000006     Dearborn St & Monroe St
##   7 TA1309000014     Clark St & Leland Ave
##   8 TA1309000014     Clark St & Leland Ave
##   9 TA1309000014     Clark St & Leland Ave
##  10 TA1309000014     Clark St & Leland Ave
## # i 7,782,171 more rows
```

It looks like many of these entries have sensible starting station names. We can use these to assign the proper station IDs. To do this, we will need to make a lookup table of station names and the corresponding IDs. But first we will have to significantly clean the station data.

**Starting Station IDs** We create two lists: starting stations with and without associated non-null IDs:

```
starting_unique_IDs <- df_trips %>%
  filter(!is.na(start_station_id) & !is.na(start_station_name)) %>%
  transform(start_station_name = trimws(start_station_name)) %>%
  select(start_station_name) %>%
  distinct()

starting_no_IDs <- df_trips %>%
  transform(start_station_name = trimws(start_station_name)) %>%
  filter(is.na(start_station_id) &
         !is.na(start_station_name) &
         !(start_station_name %in% starting_unique_IDs$start_station_name)) %>%
  select(start_station_name) %>%
  distinct()

cat("There are",length(starting_unique_IDs$start_station_name),
    "unique starting stations with non-null IDs and",
    length(starting_no_IDs$start_station_name),"without non-null IDs.")
```

```
## There are 2055 unique starting stations with non-null IDs and 1 without non-null IDs.
```

Let's see which starting station has no non-null ID:

```
print(starting_no_IDs)
```

```
##   start_station_name
## 1   hubbard_test_lws
```

– and where trips starting from this station end:

```
df_trips %>%
  filter(start_station_name == "hubbard_test_lws") %>%
  select(start_station_name, end_station_name, end_station_id)
```

```
## # A tibble: 3 x 3
##   start_station_name end_station_name                       end_station_id
##   <chr>              <chr>                                  <chr>
## 1 hubbard_test_lws   <NA>                                   <NA>
## 2 hubbard_test_lws   <NA>                                   <NA>
## 3 hubbard_test_lws   HUBBARD ST BIKE CHECKING (LBS-WH-TEST) 671
```

```
df_trips %>%
  filter(end_station_id == "671") %>%
  select(start_station_name, end_station_name)
```

```
## # A tibble: 721 x 2
##    start_station_name                      end_station_name
##    <chr>                                   <chr>
##  1 Oakley Ave & Irving Park Rd             HUBBARD ST BIKE CHECKING (LBS-WH-TEST)
##  2 Canal St & Madison St                   HUBBARD ST BIKE CHECKING (LBS-WH-TEST)
##  3 Oakley Ave & Irving Park Rd             HUBBARD ST BIKE CHECKING (LBS-WH-TEST)
##  4 HUBBARD ST BIKE CHECKING (LBS-WH-TEST) HUBBARD ST BIKE CHECKING (LBS-WH-TEST)
##  5 State St & Randolph St                  HUBBARD ST BIKE CHECKING (LBS-WH-TEST)
##  6 Oakley Ave & Irving Park Rd             HUBBARD ST BIKE CHECKING (LBS-WH-TEST)
##  7 Oakley Ave & Irving Park Rd             HUBBARD ST BIKE CHECKING (LBS-WH-TEST)
##  8 Wabash Ave & Wacker Pl                  HUBBARD ST BIKE CHECKING (LBS-WH-TEST)
##  9 Oakley Ave & Irving Park Rd             HUBBARD ST BIKE CHECKING (LBS-WH-TEST)
## 10 Oakley Ave & Irving Park Rd             HUBBARD ST BIKE CHECKING (LBS-WH-TEST)
## # i 711 more rows
```

```
ending_unique_IDs <- df_trips %>%
  filter(!is.na(end_station_id) & !is.na(end_station_name)) %>%
  transform(end_station_name = trimws(end_station_name)) %>%
  select(end_station_name) %>%
  distinct()

ending_no_IDs <- df_trips %>%
  transform(end_station_name = trimws(end_station_name)) %>%
  filter(is.na(end_station_id) &
         !is.na(end_station_name) &
         !(end_station_name %in% ending_unique_IDs$end_station_name)) %>%
  select(end_station_name) %>%
  distinct()

cat("There are",length(ending_unique_IDs$end_station_name),
    "unique ending stations with non-null IDs and",
    length(ending_no_IDs$end_station_name),"without non-null IDs.")
```

### Ending Station IDs

```
## There are 2066 unique ending stations with non-null IDs and 1 without non-null IDs.
```

The ending station without a non-null ID:

```
print(ending_no_IDs)
```

```
##   end_station_name
## 1 hubbard_test_lws
```

These stations appear to indicate test data that was not deleted. We will assume that this is the case and delete any records starting or ending at 'hubbard_test_lws', 'HUBBARD ST BIKE CHECKING (LBS-WH-TEST)', or station 671:

```
df_trips <- df_trips %>%
  filter(start_station_name != "hubbard_test_lws" &
         end_station_name != "hubbard_test_lws" &
         start_station_name != "HUBBARD ST BIKE CHECKING (LBS-WH-TEST)" &
```

```r
         end_station_name != "HUBBARD ST BIKE CHECKING (LBS-WH-TEST)" &
         start_station_id != "671" &
         start_station_id != "671")
```

This has reduced the number of observations from 19.5 million to 15.8 million:

```r
print(length(df_trips$ride_id))
```

```
## [1] 15809787
```

We now need to recreate the lists of unique starting and ending station IDs with non-null identifiers:

```r
starting_unique_IDs <- df_trips %>%
  filter(!is.na(start_station_id) & !is.na(start_station_name)) %>%
  transform(start_station_name = trimws(start_station_name)) %>%
  select(start_station_name) %>%
  distinct()
ending_unique_IDs <- df_trips %>%
  filter(!is.na(end_station_id) & !is.na(end_station_name)) %>%
  transform(end_station_name = trimws(end_station_name)) %>%
  select(end_station_name) %>%
  distinct()
```

**Mysterious Source/Sink Stations** From the number of observations in the above data frames, we find 1949 unique starting stations and 1985 unique ending stations with non-null IDs:

```r
c(length(starting_unique_IDs$start_station_name),length(ending_unique_IDs$end_station_name))
```

```
## [1] 1949 1985
```

This means that there are at least 36 ending stations that trips never originate from. Let's determine which starting stations are not ending stations (sources) and *vice versa* (sinks).

The starting stations that are not also ending stations (source stations) are:

```r
start_only_stations <- starting_unique_IDs %>%
  filter(!(starting_unique_IDs$start_station_name %in%
             ending_unique_IDs$end_station_name)) %>%
  arrange(start_station_name)
```

```r
as_tibble(start_only_stations)
```

```
## # A tibble: 44 x 1
##    start_station_name
##    <chr>
##  1 351
##  2 Chase Ave & Touhy Ave - NE
##  3 Greenwood Ave & 87th St
##  4 Hale Ave & 111th St
##  5 Halsted St & 102nd St
##  6 Jeffery Blvd & 83rd St
##  7 Karlov Ave & Milwaukee Ave
##  8 Lamon Ave & Archer Ave
##  9 Loomis Blvd & 83rd St
## 10 Lowe Ave & 98th St
## # i 34 more rows
```

One of these, station “351”, appears to be merely a station ID that has been mistakenly entered as a station name.

The ending stations that are not also starting stations (sink stations) are:

```
end_only_stations <- ending_unique_IDs %>%
  filter(!(ending_unique_IDs$end_station_name %in% starting_unique_IDs$start_station_name)) %>%
  arrange(end_station_name)
```

```
as_tibble(end_only_stations)
```

```
## # A tibble: 80 x 1
##    end_station_name
##    <chr>
##  1 Ada St & 119th St
##  2 Baltimore Ave & 132nd St
##  3 Canty Elementary School
##  4 Central Ave & Congress Pkwy
##  5 Champlain Ave & 134th St
##  6 Christiana Ave & 111th St
##  7 Cottage Grove & 84th St
##  8 East End Ave & 75th St
##  9 Eggleston Ave & 115th St
## 10 Halsted St & 94th St
## # i 70 more rows
```

It is quite strange that, in 3.5 years and 15.8 million trips, no trip has ever started at any of the 80 stations listed above. Let’s take a look at the trips ending at one of these stations to see if we can understand what’s going on:

```
df_trips %>%
  filter(end_station_name == "N Damen Ave & W Wabansia St")
```

```
## # A tibble: 2 x 13
##   ride_id          rideable_type started_at          ended_at
##   <chr>            <chr>         <dttm>              <dttm>
## 1 9D0686B8392D6464 electric_bike 2021-01-12 21:07:22 2021-01-12 21:21:08
## 2 9271624C76FCE6B3 electric_bike 2021-03-13 14:19:55 2021-03-13 14:48:32
## # i 9 more variables: start_station_name <chr>, start_station_id <chr>,
## #   end_station_name <chr>, end_station_id <chr>, start_lat <dbl>,
## #   start_lng <dbl>, end_lat <dbl>, end_lng <dbl>, member_casual <chr>
```

Nothing seems immediately unusual about these trips. Let’s look at a few more:

```
df_trips %>%
  filter(end_station_name == "Cottage Grove & 84th St")
```

```
## # A tibble: 2 x 13
##   ride_id          rideable_type started_at          ended_at
##   <chr>            <chr>         <dttm>              <dttm>
## 1 CE135955ABAEF3C2 electric_bike 2022-05-10 20:24:17 2022-05-10 20:24:48
## 2 EF388099CCAB94F4 electric_bike 2022-06-23 14:42:27 2022-06-23 14:45:13
## # i 9 more variables: start_station_name <chr>, start_station_id <chr>,
## #   end_station_name <chr>, end_station_id <chr>, start_lat <dbl>,
## #   start_lng <dbl>, end_lat <dbl>, end_lng <dbl>, member_casual <chr>
```

We do note that the first of these trips only lasted only 31 seconds; trips lasting less than one minute were meant to have been filtered out already, but it appears we will have to do this at a later step. Otherwise, nothing is unusual about these trips.

```r
df_trips %>%
  filter(end_station_name == "Public Rack - Christiana Ave & 111th St")
```

```
## # A tibble: 2 x 13
##   ride_id          rideable_type started_at          ended_at
##   <chr>            <chr>         <dttm>              <dttm>
## 1 71893F5C9E3FB4A1 electric_bike 2022-10-23 11:23:34 2022-10-23 11:30:40
## 2 00B2B548C69F77F3 electric_bike 2023-01-22 10:03:51 2023-01-22 10:12:51
## # i 9 more variables: start_station_name <chr>, start_station_id <chr>,
## #   end_station_name <chr>, end_station_id <chr>, start_lat <dbl>,
## #   start_lng <dbl>, end_lat <dbl>, end_lng <dbl>, member_casual <chr>
```

With this example, we do see a pattern start to emerge. All trips are relatively short, less than an hour, and they all involve electric bikes. Let's see if all trips to these mysterious 'sink stations' occur on electric bikes:

```r
df_trips %>%
  filter(end_station_name %in% end_only_stations$end_station_name) %>%
  select(rideable_type) %>%
  distinct()
```

```
## # A tibble: 1 x 1
##   rideable_type
##   <chr>
## 1 electric_bike
```

Indeed, these trips were all on electric bikes. Let's check trips emanating from one of the 'source stations':

```r
df_trips %>%
  filter(start_station_name == "Halsted St & 102nd St")
```

```
## # A tibble: 2 x 13
##   ride_id          rideable_type started_at          ended_at
##   <chr>            <chr>         <dttm>              <dttm>
## 1 926FD6FD52E1FBB4 electric_bike 2022-05-29 20:33:16 2022-05-29 20:43:46
## 2 E6638B071AF9568D electric_bike 2022-05-31 04:44:55 2022-05-31 05:18:40
## # i 9 more variables: start_station_name <chr>, start_station_id <chr>,
## #   end_station_name <chr>, end_station_id <chr>, start_lat <dbl>,
## #   start_lng <dbl>, end_lat <dbl>, end_lng <dbl>, member_casual <chr>
```

It appears as though there are exactly two trips corresponding to each source/sink station. Let's check if this is actually the case:

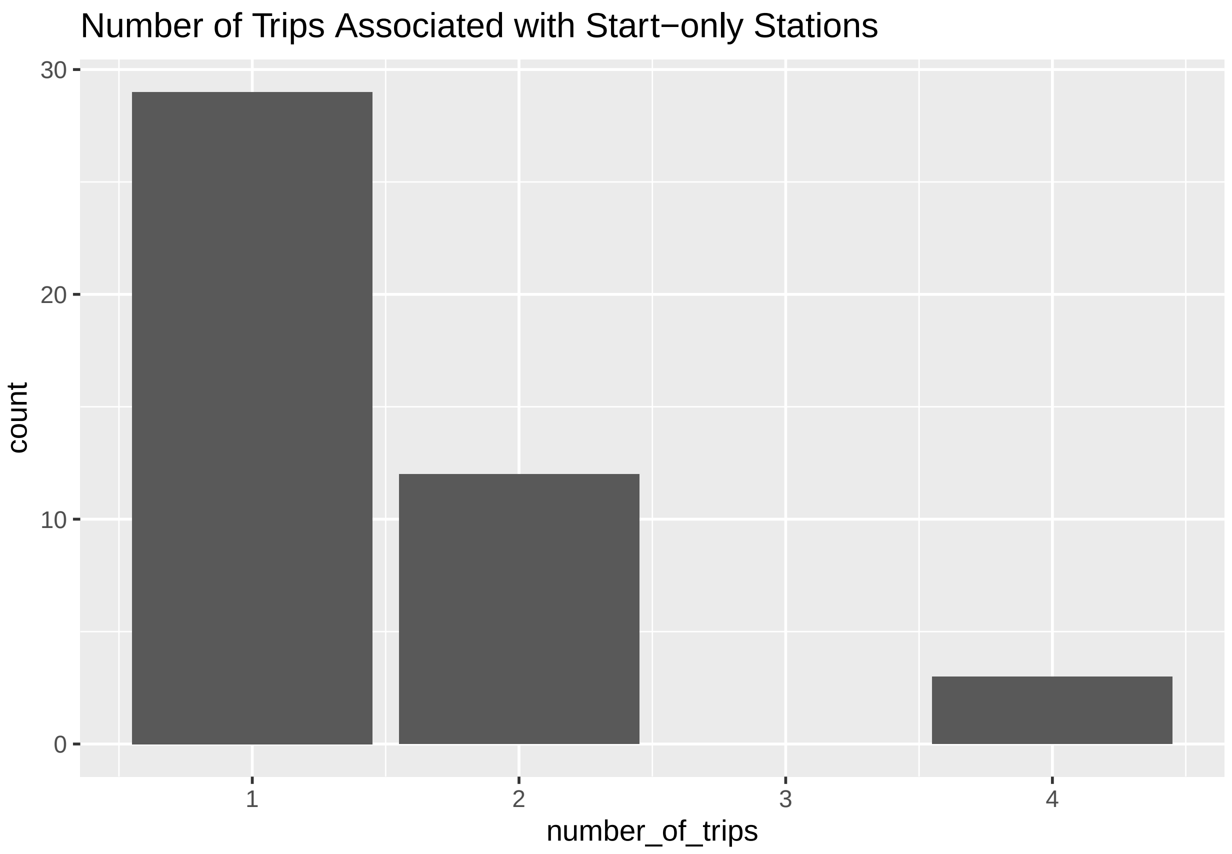
```r
df_trips %>%
  filter(start_station_name %in% start_only_stations$start_station_name) %>%
  select(start_station_name,rideable_type) %>%
  arrange(start_station_name)
```

```
## # A tibble: 65 x 2
##    start_station_name         rideable_type
##    <chr>                      <chr>
##  1 351                        electric_bike
##  2 Chase Ave & Touhy Ave - NE electric_bike
##  3 Chase Ave & Touhy Ave - NE electric_bike
##  4 Greenwood Ave & 87th St    electric_bike
##  5 Hale Ave & 111th St        electric_bike
##  6 Hale Ave & 111th St        electric_bike
##  7 Halsted St & 102nd St      electric_bike
##  8 Halsted St & 102nd St      electric_bike
##  9 Jeffery Blvd & 83rd St     electric_bike
## 10 Jeffery Blvd & 83rd St     electric_bike
## # i 55 more rows
```

We see immediately that in fact our hypothesis was wrong and there are start-only stations from which only one trip emanates. Let's look at the distribution of the number of associated trips from start-only or to end-only stations:

```
ghost_trips_start <- df_trips %>%
  filter(start_station_name %in% start_only_stations$start_station_name) %>%
  select(start_station_name,rideable_type) %>%
  arrange(start_station_name) %>%
  group_by(start_station_name) %>%
  count(start_station_name) %>%
  transform(number_of_trips = n) %>%
  select(start_station_name,number_of_trips)
```

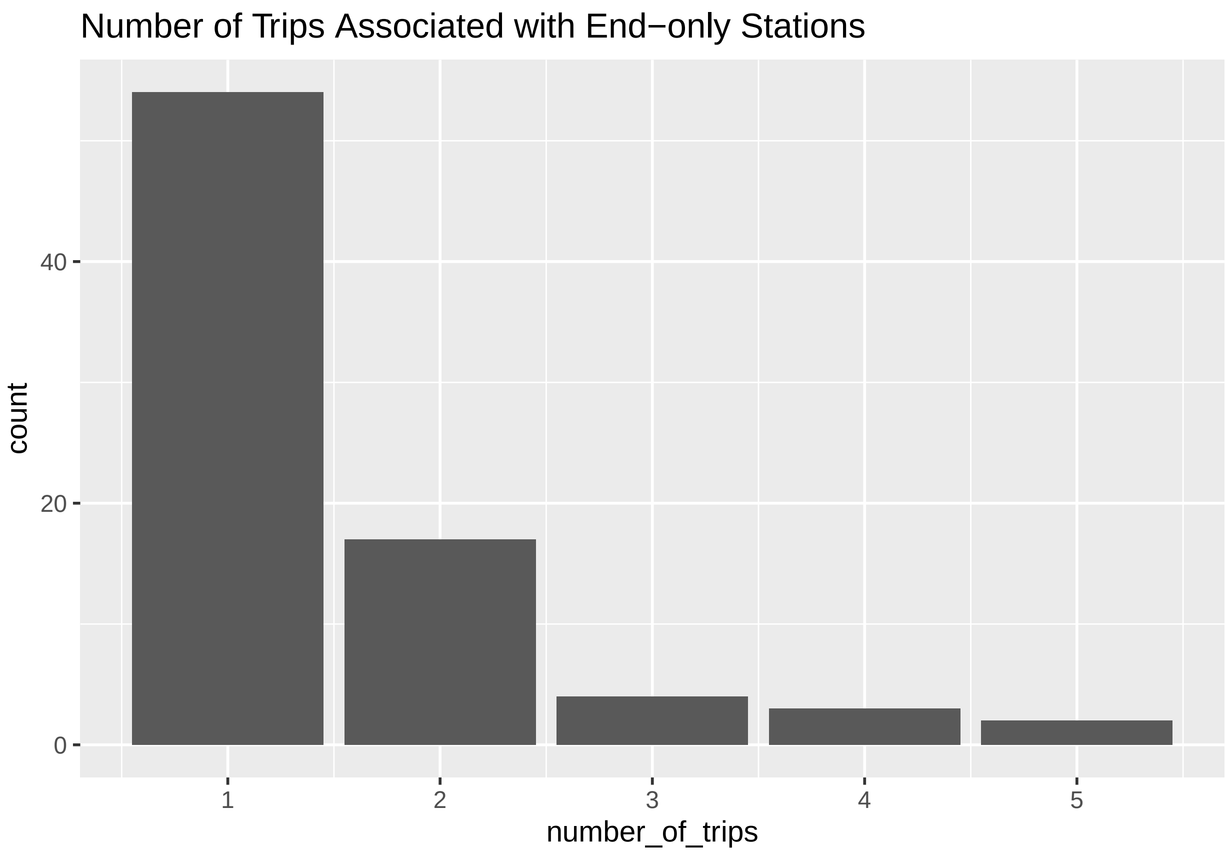
```
ggplot(data=ghost_trips_start) +
  geom_bar(aes(x=number_of_trips)) +
  labs(title = "Number of Trips Associated with Start-only Stations")
```



```
ghost_trips_end <- df_trips %>%
  filter(end_station_name %in% end_only_stations$end_station_name) %>%
  select(end_station_name,rideable_type) %>%
  arrange(end_station_name) %>%
  group_by(end_station_name) %>%
  count(end_station_name) %>%
```

```r
  transform(number_of_trips = n) %>%
  select(end_station_name,number_of_trips)
```

```r
ggplot(data=ghost_trips_end) +
  geom_bar(aes(x=number_of_trips)) +
  labs(title = "Number of Trips Associated with End-only Stations")
```



The number of trips associated with start-only (source) stations varies between 1 and 4, and with end-only (sink) stations varies between 1 and 5.

Nonetheless, these patterns indicate that there is something 'off' about this data; even if there are certain stations that only accept electric bikes, they should still be represented in both the starting and ending station data. The issue with the data is not clear without further clarification from the client. Because we cannot further clarify with the client, we will proceed with the analysis using the data as-is.

**Assign Unique Station IDs** Before proceeding, let's delete any data for which both the starting station name and ID – or both the ending station name and ID – are null. We do not have enough information for these trips to determine where they started/ended from, and hence cannot use them effectively in the subsequent analysis.

```r
df_trips <- df_trips %>%
  filter(!is.na(start_station_name) | !is.na(start_station_id))
```

```r
df_trips <- df_trips %>%
  filter(!is.na(start_station_name) | !is.na(start_station_id))
```

```r
print(length(df_trips$ride_id))
```

```
## [1] 15809787
```

We still have 15.8 million observations. It looks like the trips we had retained all had associated started and ending stations already.

Let's continue to make a table of all unique stations and their associated IDs. (As we'll find, there is frequently more than one ID associated with a given station.)

```
station_lookup <- rbind(
  df_trips %>%
    filter(!is.na(start_station_name) & !is.na(start_station_id)) %>%
    transform(name = start_station_name, id = start_station_id) %>%
    select(name,id),
  df_trips %>%
    filter(!is.na(end_station_name) & !is.na(start_station_name)) %>%
    transform(name = end_station_name, id = end_station_id) %>%
    select(name,id)) %>%
  distinct() %>%
  filter(!is.na(id)) %>%
  arrange(name)
```

```
as_tibble(station_lookup)
```

```
## # A tibble: 2,639 x 2
##    name                           id
##    <chr>                          <chr>
##  1 10101 S Stony Island Ave       922
##  2 111th St - Morgan Park Metra   682
##  3 2112 W Peterson Ave            456
##  4 2112 W Peterson Ave            KA1504000155
##  5 351                            351
##  6 410                            410
##  7 532 E 43rd St                  913
##  8 63rd & Western Ave - north corner 739
##  9 63rd & Western Ave - south corner 738
## 10 63rd St Beach                  101
## # i 2,629 more rows
```

```
length(station_lookup$id)
```

```
## [1] 2639
```

There are many duplicate entries for stations in this table, leading to a total of 2639 observations. However, a quick look at this data frame suggests that each station has exactly one station ID that may be converted to an integer. Let's try to convert the station IDs to integers, remove all nulls produced by this conversion, and check that the number of entries in this new lookup table matches the number of unique station names. The number of unique station names is:

```
unique_stations <- rbind(
  starting_unique_IDs %>%
    transform(name = start_station_name) %>%
    select(name),
  ending_unique_IDs %>%
    transform(name = end_station_name) %>%
    select(name)) %>%
  distinct() %>%
  arrange(name)
print(length(unique_stations$name))
```

```
## [1] 2029
```

Let's check if there are also 2029 'integer-like' station IDs:

```
nrow(station_lookup %>%
        transform(id = as.integer(id)) %>%
        filter(!is.na(id)) %>%
        distinct())
```

```
## [1] 2291
```

This produces 2291 integer-like station IDs, more than the 2029 unique stations. Clearly, there are still duplicates, though this has reduced the number of duplicates and all duplicated IDs are now integers. If there is at least one integer-like ID associated with each station, we can assign a unique ID to each station by taking the lowest integer associated with that station as the station ID:

```
nrow(station_lookup %>%
        transform(id = as.integer(id)) %>%
        filter(!is.na(id)) %>%
        distinct() %>%
        group_by(name) %>%
        summarize(id = min(id)))
```

```
## [1] 1985
```

Unfortunately, this produces 1985 entries, less than the number of stations. Clearly, some stations do not have integer values associated with them. We will take a new approach: we create new station IDs, defined by numbering the ordered list of unique station names sequentially:

```
station_locations <- rbind(
  df_trips %>%
    transform(name = start_station_name, lat = start_lat, lng = start_lng) %>%
    select(name, lat, lng) %>%
    distinct(),
  df_trips %>%
    transform(name = end_station_name, lat = end_lat, lng = end_lng) %>%
    select(name, lat, lng) %>%
    distinct()
) %>%
  distinct() %>%
  arrange(name) %>%
  group_by(name) %>%
  summarize(lat = mean(lat), lng = mean(lng))
```

```
unique_station_lookup <- data.frame(name = unique_stations$name,
                                    id = 1:length(unique_stations$name),
                                    lat = station_locations$lat,
                                    lng = station_locations$lng)
```

Because we have saved station locations in our 'unique_station_lookup' data frame, we can delete these columns from 'df_trips':

```
df_trips <- df_trips %>%
  select(ride_id,
         rideable_type,
         started_at,ended_at,
         start_station_name,
         start_station_id,
         end_station_name,
         end_station_id,
         member_casual)
```

We now need to replace all starting and ending station IDs with these new station IDs. We will cycle over each unique station, reassign all IDs associated with that station to its new unique ID, and assign the unique station ID to any other observations with the same station name and no ID.

```
# NB: I attempted a multiprocessing implementation using foreach and the %dopar%
# operator + parallel library to create a forking backend, but it did not use
# multiple cores efficiently, typically only using half of a single physical core;
# need to figure out a more efficient way to implement basic multiprocessing
for (station_name in unique_station_lookup$name) {
  # vector of IDs currently associated w/ station
  IDvec = as.vector(filter(station_lookup,name == station_name)[2])
  # new unique station ID
  uniqueID = as.character(filter(unique_station_lookup,name == station_name)[2])
  # make sure all IDs corresponding to given station are identical
  for (ID in IDvec) { # appears to be more efficient than using %in%
    df_trips <- within(df_trips, start_station_id[start_station_id == ID] <- uniqueID)
    df_trips <- within(df_trips, end_station_id[end_station_id == ID] <- uniqueID)
  }
  # make sure all listed stations have corresponding IDs
  # NB: did not perform logical test to exclude data that already has both the
  # station name and the unique ID
  # informal testing indicated that this significantly increased execution time
  # of this loop
  df_trips <- within(df_trips, start_station_id[start_station_name == station_name] <- uniqueID)
  df_trips <- within(df_trips, end_station_id[end_station_name == station_name] <- uniqueID)
}
```

Let's make sure we no longer have any null starting or ending station IDs:

```
c(sum(is.na(df_trips$start_station_id)),sum(is.na(df_trips$end_station_id)))
```

```
## [1] 0 0
```

And now let's convert the station IDs to integers and confirm there are no nulls introduced by the conversion:

```
df_trips <- df_trips %>%
  transform(start_station_id = as.integer(start_station_id),
            end_station_id = as.integer(end_station_id))
c(sum(is.na(df_trips$start_station_id)),sum(is.na(df_trips$end_station_id)))
```

```
## [1] 0 0
```

We now have unique identifiers for each station that we can lookup using the 'unique_station_lookup' data frame. We can now remove the station names from the df_trips data frame:

```
df_trips <- df_trips %>%
  select(ride_id,
         rideable_type,
         started_at,
         ended_at,
         start_station_id,
         end_station_id,
         member_casual)
```

Let’s next check if the data in the ‘started_at’ and ‘ended_at’ columns can all be converted to datetime format:

```
c(sum(is.na(ymd_hms(df_trips$started_at))),sum(is.na(ymd_hms(df_trips$ended_at))))
```

```
## [1] 64 69
```

There are 64 ‘started_at’ and 69 ‘ended_at’ entries that do not parse. Let’s take a look at these entries to see why:

```
df_trips %>%
  filter(is.na(ymd_hms(started_at)) | is.na(ymd_hms(ended_at))) %>%
  as_tibble()
```

```
## # A tibble: 133 x 7
##    ride_id          rideable_type started_at          ended_at
##    <chr>            <chr>         <dttm>              <dttm>
##  1 D077A98B1B27ADCE docked_bike   2020-05-24 23:35:44 2020-05-25 00:00:00
##  2 5ABDF8E657319A6C docked_bike   2020-05-25 22:05:48 2020-05-26 00:00:00
##  3 13F9954F4BA49088 docked_bike   2020-05-25 22:02:45 2020-05-26 00:00:00
##  4 8B44E58E918C1CE7 docked_bike   2020-06-12 00:00:00 2020-06-12 00:15:30
##  5 1FF15DBCD7921AF2 docked_bike   2020-06-28 00:00:00 2020-06-28 00:16:08
##  6 8278EBAFA368E887 docked_bike   2020-07-29 23:48:11 2020-07-30 00:00:00
##  7 3C5869977E33D16C docked_bike   2020-07-11 00:00:00 2020-07-11 01:13:38
##  8 A788FC24BD16BA51 docked_bike   2020-07-02 22:46:18 2020-07-03 00:00:00
##  9 DDEFB093955B346C docked_bike   2020-08-08 00:00:00 2020-08-08 00:21:23
## 10 57764F95A84C329D docked_bike   2020-08-09 22:58:03 2020-08-10 00:00:00
## # i 123 more rows
## # i 3 more variables: start_station_id <int>, end_station_id <int>,
## #   member_casual <chr>
```

It appears there is an issue parsing datetimes at exactly midnight (00:00:00). It is unclear why. For the sake of time, and because this represents only a vanishingly small fraction of the data, I will simply delete these entries:

```
df_trips <- df_trips %>%
  transform(started_at = ymd_hms(started_at), ended_at = ymd_hms(ended_at)) %>%
  filter(!is.na(started_at) & !is.na(ended_at))
```

Let’s confirm that there are no remaining null values in any of the columns:

```
colSums(is.na(df_trips))
```

```
##          ride_id    rideable_type       started_at         ended_at
##                0                0                0                0
## start_station_id   end_station_id    member_casual
##                0                0                0
```

**Derived Attributes** Let’s append a few useful derived attributes to our data frame that we will use in the subsequent analysis. In particular, we would like to look at the day of week, start time, duration, and month and year of each ride:

```
df_trips <- df_trips %>%
  mutate(
         # numeric weekday of trip with 1 = Monday
         weekday = wday(started_at, week_start = 1),
         # start time of trip
         start_time = as_hms(started_at),
         # duration of trip in minutes
         duration = as.double(difftime(ended_at,started_at,units = "mins")),
         # month of trip
         month = month(started_at,label=FALSE),
         # year of trip
         year = year(started_at)
         )
```

Filter out trips lasting less than one minute or longer than 24 hours = 1440:

```
df_trips <- df_trips %>%
  filter(duration > 1 & duration < 1440)
print(length(df_trips$ride_id))
```

```
## [1] 15538706
```

Though the client reported having already deleted trips that lasted less than one minute or more than 24 hours, the number of observations decreased from 15.8 to 15.5 million.

## The Analysis

We now conduct our main analysis, reminding ourselves that we want to find characteristics that discriminate casual customers from annual members. In all of the following analysis, we make the assumption of statistical stationarity – i.e., that the distributions of attributes have not evolved significantly over the 3.5 years covered by our analysis. In fact, a more careful analysis would check this assumption of stationarity by, for example, examining distributions by year/month/quarter or some other temporal sub-bin. I won't confirm the stationarity of the data here given constraints on time.

### Starting Station Distribution

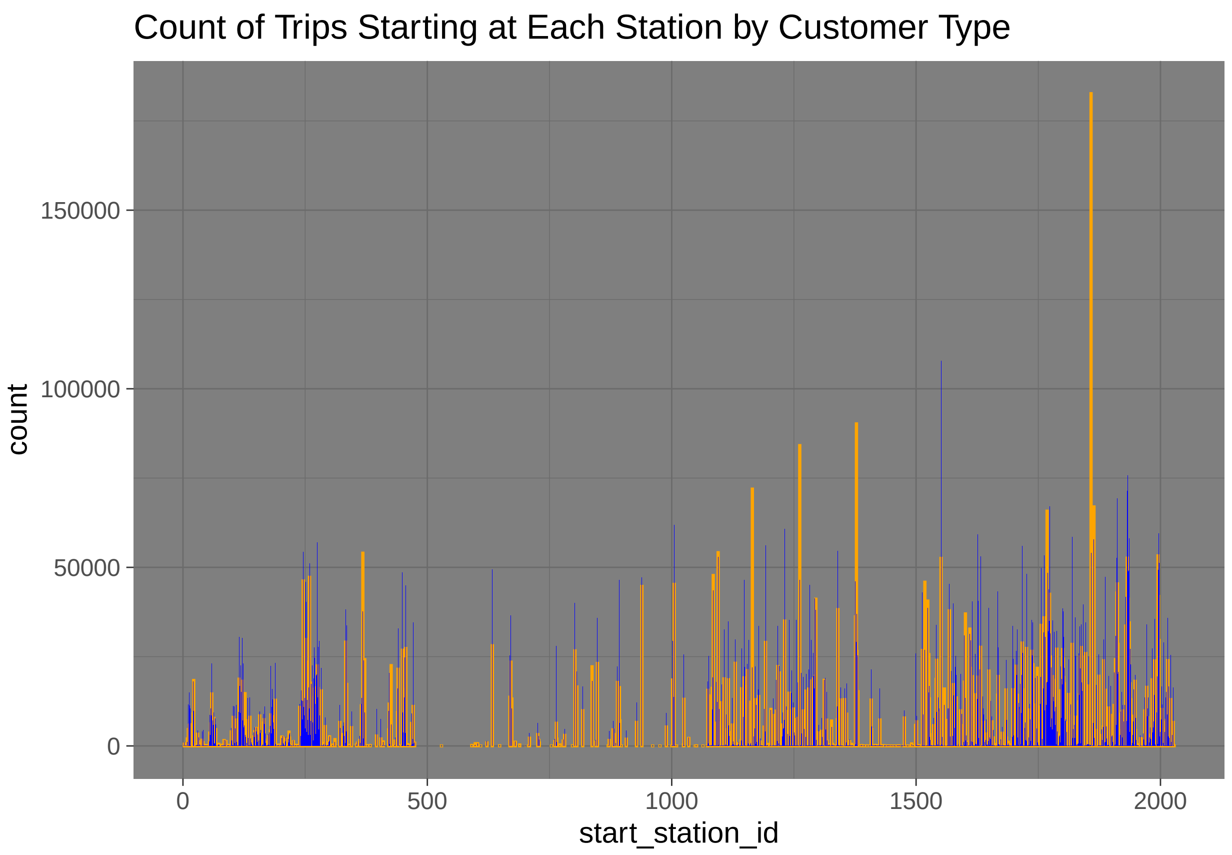
Our data is composed of over 9.1 million member trips and nearly 6.4 million casual trips:

```
c(nrow(filter(df_trips, member_casual == "member")),
  nrow(filter(df_trips, member_casual == "casual")))
```

```
## [1] 9149528 6389178
```

Let's take a look at the number of casual and member trips starting from each station:

```
ggplot(data = filter(df_trips, member_casual == "casual"),
       aes(x = start_station_id)) +
  geom_bar(fill = "orange", color = "orange") +
  geom_bar(data = filter(df_trips, member_casual == "member"),
           aes(x = start_station_id), fill = "blue") +
  labs(title = "Count of Trips Starting at Each Station by Customer Type") +
  theme_dark()
```

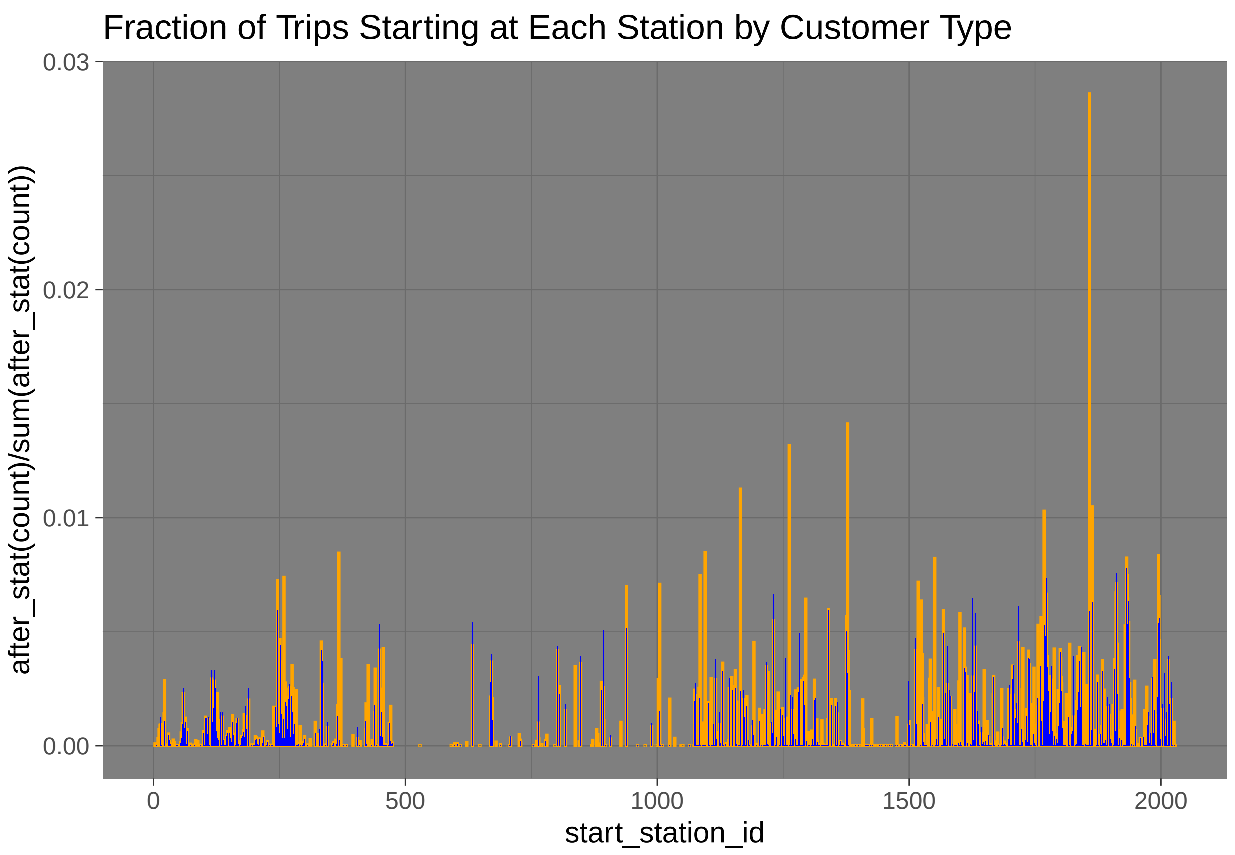


The most immediately notable feature of this plot is that the casual customer trips (orange) appear to be *heavily* dominated by just a few stations; in fact, even though the number of member trips exceeds the number of casual trips by approximately 50%, the distribution of casual trips shows the largest peaks in the plot. At this stage, we formulate three hypotheses:

- The distribution of the fraction of trips decays abruptly between a few stations from which most casual trips emanate and the rest. We will test this by plotting the fraction of casual trips emanating from a station versus its popularity rank to determine if there is an abrupt transition between more and less popular stations.
- There is no such abrupt transition for annual members.
- These ‘super stations’ for casual members are likely clustered in a downtown commercial region, indicating that these trips are primarily commutes and/or weekend excursions. We will test this by looking at the geographic distribution of super stations and the distribution of casual trips across time of day and day of the week.

Let’s look at the difference in the distribution of trips according to starting location by normalizing the trip counts:

```r
ggplot(data = filter(df_trips, member_casual == "casual"),
       aes(x = start_station_id, y = after_stat(count)/sum(after_stat(count)))) +
  geom_bar(fill = "orange", color = "orange") +
  geom_bar(data = filter(df_trips, member_casual == "member"),
           aes(x = start_station_id, y = after_stat(count)/sum(after_stat(count))),
           fill = "blue") +
  labs(title = "Fraction of Trips Starting at Each Station by Customer Type") +
  theme_dark()
```



When we normalize by the number of trips of a given customer type, it becomes even more apparent that casual trips are dominated by a relatively small subset of all stations.

Next, we want to plot the fraction of trips coming from a station versus the popularity rank of that station (with a rank of one indicating the most popular starting station, from which the largest fraction of trips emanate). This will allow us to test our hypothesis regarding the existence of super stations in the casual customer trips data. We construct two data frames containing the ranked starting stations and corresponding fraction of trips emanating from that station:

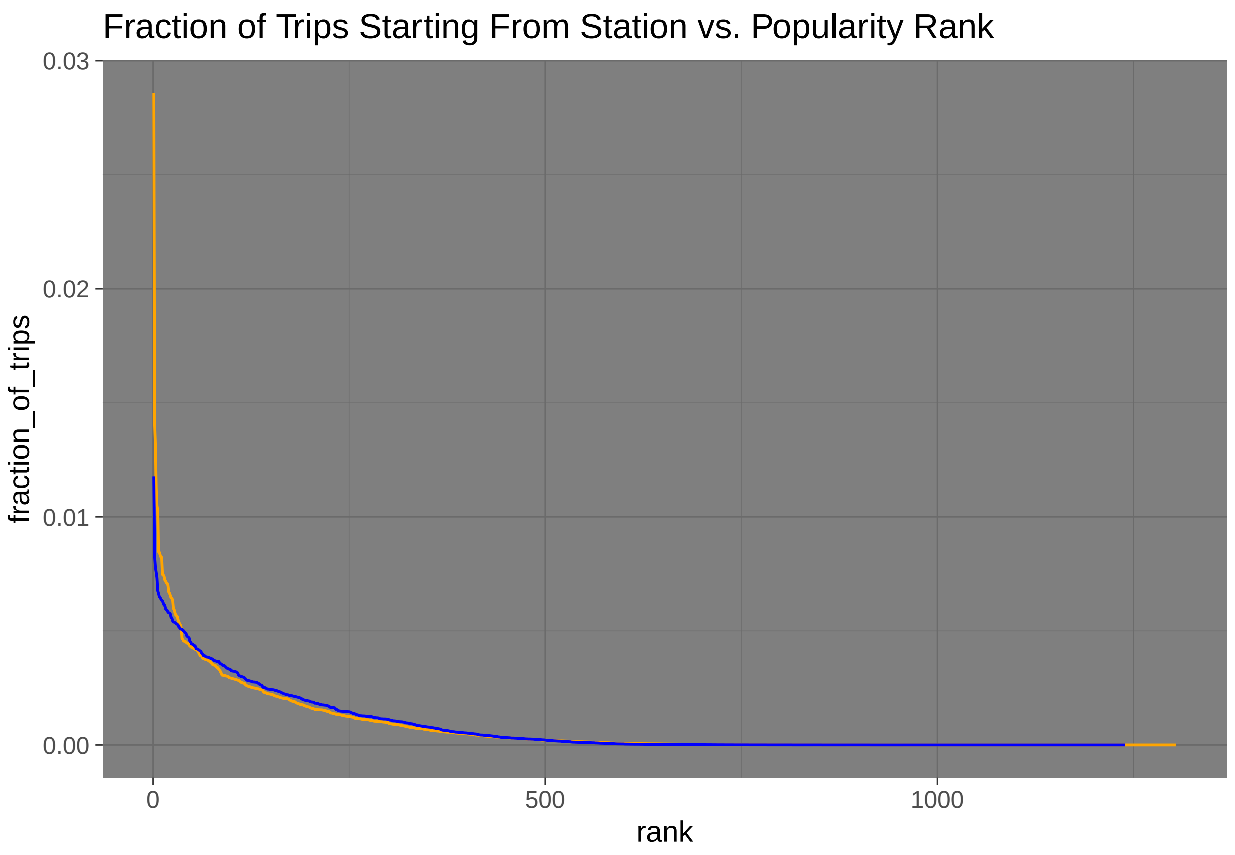
```r
number_of_casual_trips <- nrow(filter(df_trips,member_casual == "casual"))
number_of_member_trips <- nrow(filter(df_trips,member_casual == "member"))
```

```r
ranked_casual_starting_station <- df_trips %>%
  filter(member_casual == "casual") %>%
  group_by(start_station_id) %>%
  count(start_station_id) %>%
  transform(number_of_trips = n) %>%
  select(start_station_id, number_of_trips) %>%
  arrange(desc(number_of_trips))
ranked_casual_starting_station <- ranked_casual_starting_station %>%
  transform(rank = 1:nrow(ranked_casual_starting_station),
            fraction_of_trips = number_of_trips/number_of_casual_trips) %>%
  select(rank, start_station_id, number_of_trips, fraction_of_trips)
```

```r
ranked_member_starting_station <- df_trips %>%
  filter(member_casual == "member") %>%
  group_by(start_station_id) %>%
  count(start_station_id) %>%
  transform(number_of_trips = n) %>%
  select(start_station_id, number_of_trips) %>%
  arrange(desc(number_of_trips))
ranked_member_starting_station <- ranked_member_starting_station %>%
  transform(rank = 1:nrow(ranked_member_starting_station),
            fraction_of_trips = number_of_trips/number_of_member_trips) %>%
  select(rank, start_station_id, number_of_trips, fraction_of_trips)
```

We use these data frames to plot the fraction of trips emanating from a station versus the station popularity rank:

```
ggplot(data = ranked_casual_starting_station,
       aes(x = rank, y = fraction_of_trips)) +
  # geom_point(color = "orange") +
  geom_line(color = "orange") +
  # geom_point(data = ranked_member_starting_station, color = "blue") +
  geom_line(data = ranked_member_starting_station, color = "blue") +
  labs(title = "Fraction of Trips Starting From Station vs. Popularity Rank") +
  theme_dark()
```

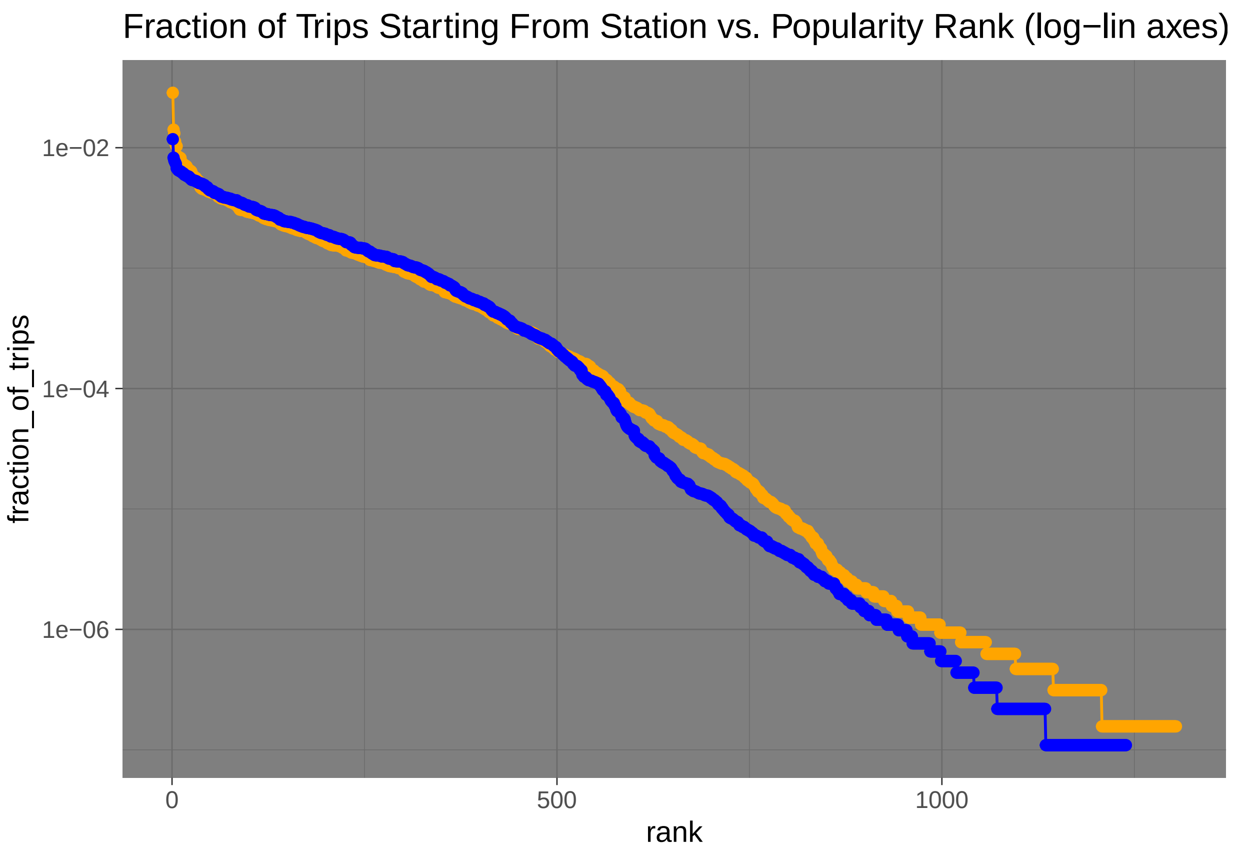


We see indeed that there is a precipitous, super-exponential decrease in the fraction of trips as rank decreases for casual trips. Actually, both curves appear to exhibit an abrupt slope change and deviation from exponential behavior for highly ranked stations, though it is much more pronounced – and the highest ranked stations are considerably more popular – for casual trips.

This redistribution of station popularity is indicated by the curve cross-over: Moving from left to right (high rank to lower rank), we see that initially the orange curve – corresponding to casual trips – is above the blue (annual member) curve; this indicates that there is a shift in popularity towards the highest ranked stations for casual trips relative to member trips. To the right of this cross-over – where the blue annual member curve is now slightly above the orange casual customer curve – there is a relative dearth of popularity of casual starting stations, indicated by the vertical reordering of the curves.

Let's take a look at this plot with the fraction of trips plotted logarithmically. Deviations from a linear relationship will indicate deviations from an exponential relationship between rank and fraction of trips:

```r
ggplot(data = ranked_casual_starting_station,
       aes(x = rank, y = fraction_of_trips)) +
  geom_point(color = "orange") +
  geom_line(color = "orange") +
  geom_point(data = ranked_member_starting_station, color = "blue") +
  geom_line(data = ranked_member_starting_station, color = "blue") +
  scale_y_continuous(trans = "log10") +
  labs(title = "Fraction of Trips Starting From Station vs. Popularity Rank (log-lin axes)") +
  theme_dark()
```



There is an apparent secondary cross-over – and deviations from linearity – for stations with rank 500 or greater. However, these stations account for only 2% of trips or less, and this behavior can safely be excluded from our analysis:

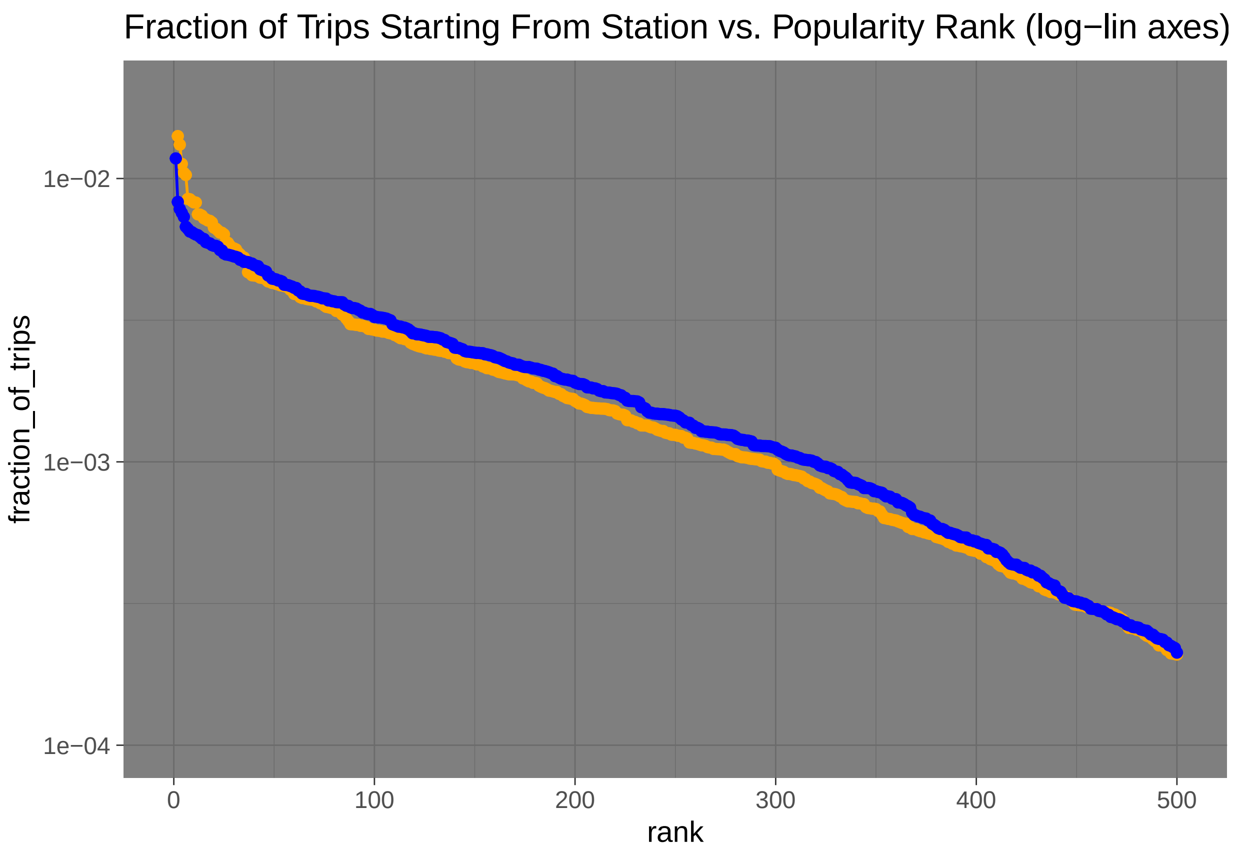
```r
sum(ranked_casual_starting_station %>%
      filter(rank > 500) %>%
      select(fraction_of_trips)) * 100
```

```
## [1] 2.066166
```

```r
sum(ranked_member_starting_station %>%
      filter(rank > 500) %>%
      select(fraction_of_trips)) * 100
```

```
## [1] 1.473169
```

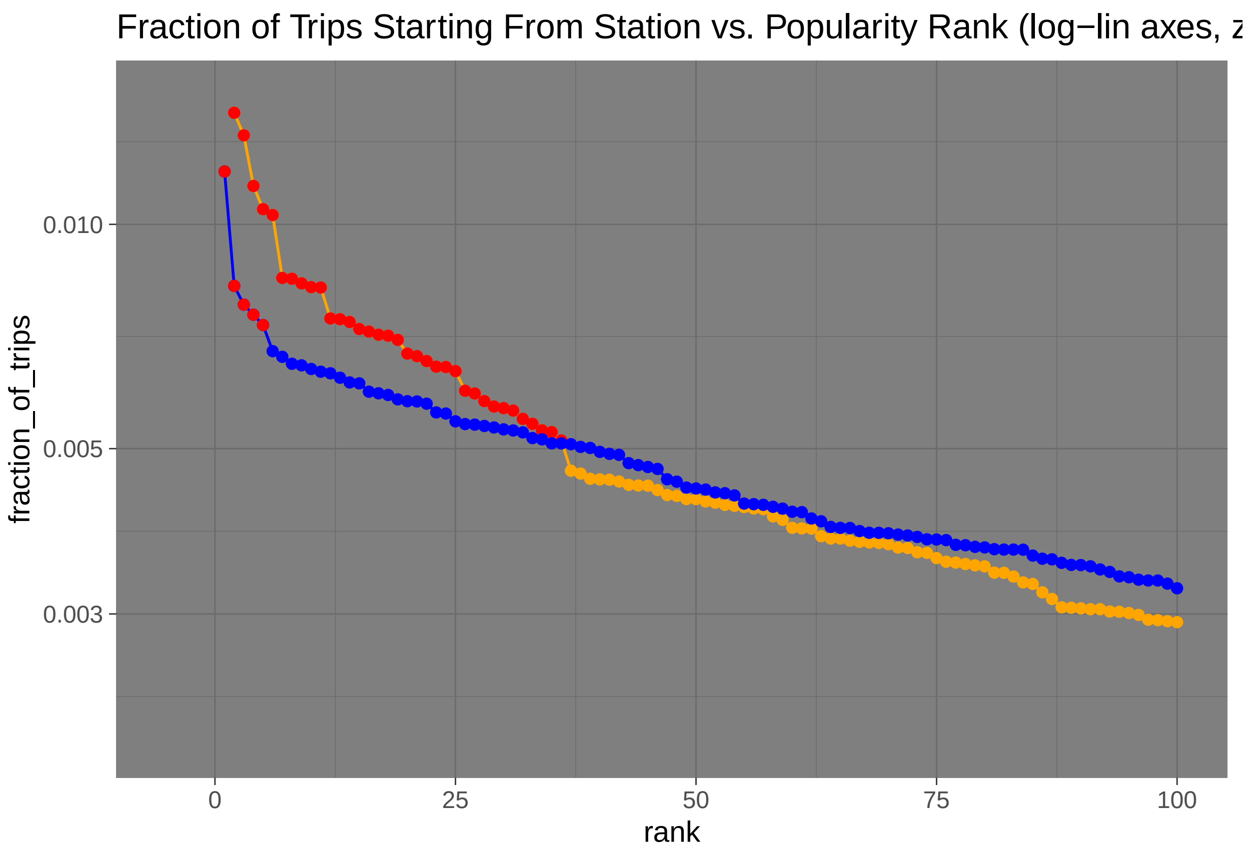
```
ggplot(data = ranked_casual_starting_station,
       aes(x = rank, y = fraction_of_trips)) +
  geom_point(color = "orange") +
  geom_line(color = "orange") +
  geom_point(data = ranked_member_starting_station, color = "blue") +
  geom_line(data = ranked_member_starting_station, color = "blue") +
  xlim(0,500) +
  scale_y_continuous(trans = "log10", limits = c(1e-4,2e-2)) +
  labs(
    title = "Fraction of Trips Starting From Station vs. Popularity Rank (log-lin axes)"
    ) +
  theme_dark()
```



We are interested in the upward tail at the beginning of both curves. These points indicate strong deviations from an exponential decay, with the popularity of highly ranked stations decaying super-exponentially. Accordingly, these stations are called super stations.

Let's take a closer look at the super stations:

```
ggplot(data = ranked_casual_starting_station,
       aes(x = rank, y = fraction_of_trips)) +
  geom_point(color = "orange") +
  geom_line(color = "orange") +
  geom_point(data = filter(ranked_casual_starting_station, rank <= 36),
             color = "red") +
  geom_point(data = ranked_member_starting_station, color = "blue") +
  geom_line(data = ranked_member_starting_station, color = "blue") +
  geom_point(data = filter(ranked_member_starting_station,rank <= 5),
  color = "red") +
  xlim(-5,100) +
  # ylim(0.002,0.015) +
  scale_y_continuous(trans = "log10", limits = c(0.002,0.015)) +
  labs(
    title = "Fraction of Trips Starting From Station vs. Popularity Rank (log-lin axes, zoom)"
    ) +
  theme_dark()
```



We've colored the super stations – the highest ranked stations where the exponential relationship between rank and fraction of trips breaks down – red; there are five member super stations and 36 casual super stations. Let's check the total fraction of trips starting from these super stations:

```
sum(
  ranked_member_starting_station %>%
    filter(rank <= 5) %>%
    select(fraction_of_trips)
) * 100
```

```
## [1] 4.273423
```

```
sum(
  ranked_casual_starting_station %>%
    filter(rank <= 36) %>%
    select(fraction_of_trips)
) * 100
```

```
## [1] 28.79586
```

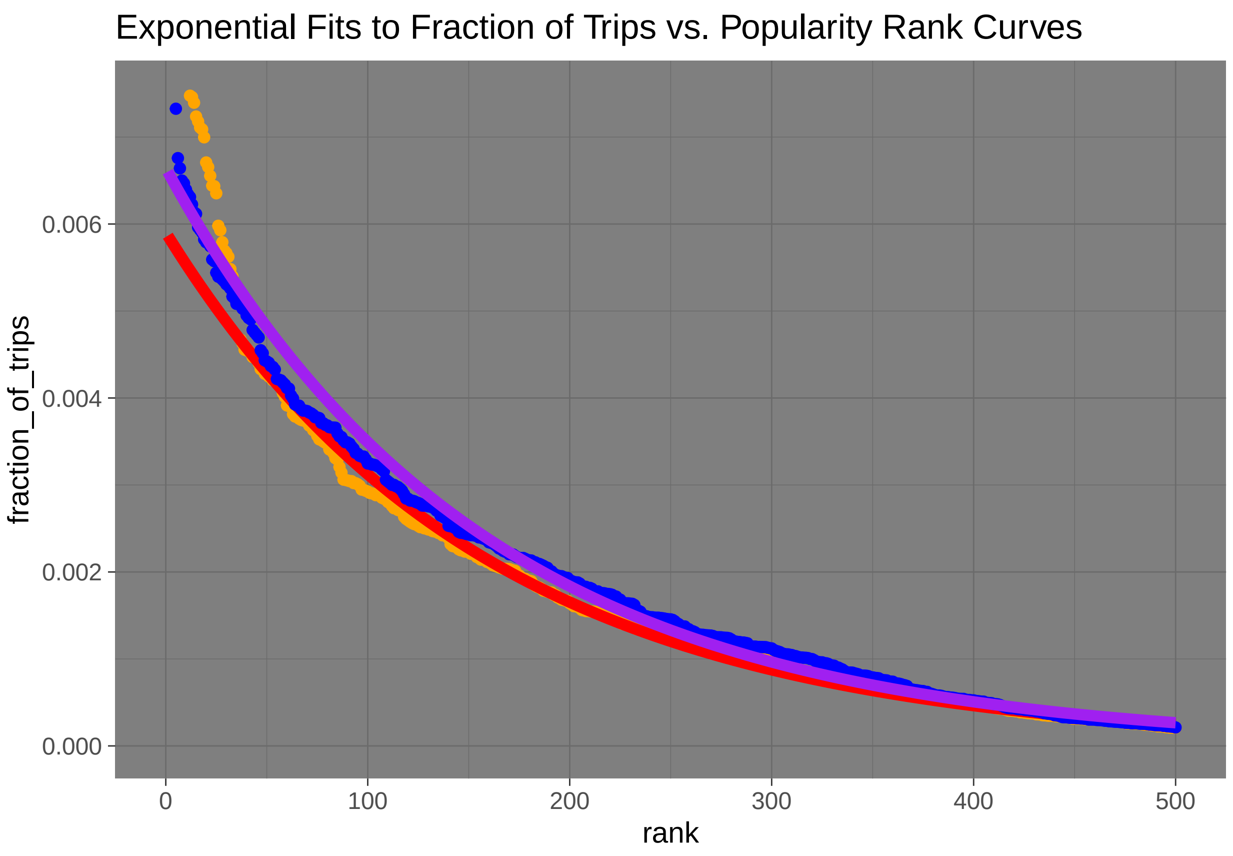
Out of 1239 distinct member trip starting stations, there are five member super stations – or about 0.4% – accounting for about 4% of member starting stations; out of 1304 distinct casual trip starting stations, there are 36 casual super stations – or about 2.7% – that account for nearly 30% of casual starting stations.

In order to determine how significant these results are, let's compare these numbers to what we would obtain if the exponential trend held for even the most popular stations:

```
# exponential fit to casual trip data
rank_casual <- 37:500
fraction_of_trips_casual <- unlist(ranked_casual_starting_station %>%
                                     filter(rank > 36 & rank <= 500) %>%
                                     select(fraction_of_trips))
casual_linexp_fit <- lm(log(fraction_of_trips_casual) ~ rank_casual)
rank_fit <- 1:max(nrow(ranked_casual_starting_station),
                  nrow(ranked_member_starting_station))
casual_fraction_of_trips_fit <- exp(coef(casual_linexp_fit)[1]) *
  exp(coef(casual_linexp_fit)[2]*rank_fit)
# exponential fit to member trip data
rank_member <- 6:500
fraction_of_trips_member <- unlist(ranked_member_starting_station %>%
                                     filter(rank > 5 & rank <= 500) %>%
                                     select(fraction_of_trips))
member_linexp_fit <- lm(log(fraction_of_trips_member) ~ rank_member)
member_fraction_of_trips_fit <- exp(coef(member_linexp_fit)[1]) *
  exp(coef(member_linexp_fit)[2]*rank_fit)
# construct data frame
exponential_fits_fraction_of_trips <- data.frame(rank = rank_fit,
                                                 casual_fit =
                                                   casual_fraction_of_trips_fit,
                                                 member_fit =
                                                   member_fraction_of_trips_fit)
```

Let's plot these exponential fits:

```
ggplot(data = ranked_casual_starting_station) +
  geom_point(aes(x = rank, y = fraction_of_trips), color = "orange") +
  geom_line(data = exponential_fits_fraction_of_trips,
            aes(x = rank, y = casual_fit),
            color = "red",
            size = 2) +
  geom_point(data = ranked_member_starting_station,
             aes(x = rank, y = fraction_of_trips), color = "blue") +
  geom_line(data = exponential_fits_fraction_of_trips,
            aes(x = rank, y = member_fit),
            color = "purple",
            size = 2) +
  # scale_x_continuous(limits = c(0,500)) +
  # scale_y_continuous(trans = "log10", limits = c(2e-4,0.0075))
  xlim(0,500) +
  ylim(0,0.0075) +
  theme_dark() +
  labs(title = "Exponential Fits to Fraction of Trips vs. Popularity Rank Curves")
```



Though they are not perfect, these fits do seem to describe the trend in popularity for changing rank well – except for the super stations that we have excluded from the fit. We can use these exponential fits to quantify the percent of trips we would anticipate to start from the super stations we have identified if they were not super stations – that is, if they obeyed the same exponential scaling between popularity and rank that all other stations obey:

```
sum(
  exponential_fits_fraction_of_trips %>%
    filter(rank <= 5) %>%
    select(member_fit)
) * 100
```

```
## [1] 3.258834
```

```
sum(
  exponential_fits_fraction_of_trips %>%
    filter(rank <= 36) %>%
    select(casual_fit)
) * 100
```

```
## [1] 18.9319
```

We learn that, if there were no super stations, the highest ranked member starting stations would account for 3.3% of trips, instead of 4.3% – and the highest ranked casual starting stations would account for 18.9% of trips instead of 28.8%. Therefore, the casual super stations account for a significantly enhanced percentage of trip starts. Note that this does not imply that a majority of trips start at these super stations, only a significantly enhanced percentage – about 50% more than we would anticipate from an exponential scaling.

Let's take a look at the most prominent stations by trip type:

```
list(ranked_member_starting_station,
     select(transform(unique_station_lookup, start_station_id = id),
                                            name, start_station_id)) %>%
  reduce(inner_join, by = 'start_station_id') %>%
  filter(rank <= 5) %>%
  transform(percent_of_trips = 100*fraction_of_trips) %>%
  select(name, percent_of_trips) %>%
  arrange() %>%
  print()
```

```
##                                name percent_of_trips
## 1 Public Rack - Rockwell & 111th St        1.1774487
## 2                  Wells St & Elm St        0.8268186
## 3              Wells St & Concord Ln        0.7802370
## 4          Wabash Ave & Roosevelt Rd        0.7563669
## 5    Sheffield Ave & Wellington Ave        0.7325514
```

```
list(ranked_casual_starting_station,
     select(transform(unique_station_lookup, start_station_id = id),
                                            name, start_station_id)) %>%
  reduce(inner_join, by = 'start_station_id') %>%
  filter(rank <= 36) %>%
  transform(percent_of_trips = 100*fraction_of_trips) %>%
  select(name, percent_of_trips) %>%
  arrange() %>%
  print()
```

```
##                                        name percent_of_trips
## 1                      Streeter Dr & Grand Ave        2.8587402
## 2          Public Rack - Marquette Rd & 67th St        1.4117309
## 3         Public Rack - Kedzie Ave & Archer Ave        1.3166482
## 4             Public Rack - Hale Ave & 111th St        1.1259977
## 5                          Theater on the Lake        1.0479752
## 6                               Shedd Aquarium        1.0293030
## 7     Public Rack - Cicero Ave & Wellington Ave        0.8473234
## 8             DuSable Lake Shore Dr & North Blvd        0.8452418
## 9             William Rainey Harper High School        0.8333466
## 10                        Wells St & Concord Ln        0.8236740
## 11            Public Rack - Rockwell & 111th St        0.8226254
## 12 Public Rack - Christiana Ave & Bryn Mawr Ave        0.7475610
## 13                            Wells St & Elm St        0.7458862
## 14                        Clark St & Lincoln Ave        0.7394691
## 15                       Clark St & Armitage Ave        0.7236299
## 16                   Public Rack - Pulaski & 52nd        0.7179640
## 17                     Wabash Ave & Roosevelt Rd        0.7109678
## 18           Public Rack - Ashland Ave & 83rd St        0.7088392
## 19                Paul Revere Elementary School        0.6996988
## 20                        Wabash Ave & Grand Ave        0.6707592
## 21               Sheffield Ave & Wellington Ave        0.6654847
## 22                      Wilton Ave & Belmont Ave        0.6554051
## 23                     Wells St & Evergreen Ave        0.6441204
## 24          Public Rack - King Dr & Oakwood Blvd        0.6435257
## 25            Public Rack - Pulaski Rd & 40th St        0.6353556
## 26          Public Rack - Lawndale Ave & 63rd St        0.5979955
## 27          Public Rack - Sangamon St & 79th St        0.5929088
## 28       Public Rack - Talman Ave & Pershing Rd        0.5791042
## 29                Sheffield Ave & Waveland Ave        0.5694942
## 30        Public Rack - Maplewood Ave & 63rd St        0.5666457
## 31                     Sedgwick St & North Ave        0.5623572
## 32                   Public Rack - Jensen Park        0.5481926
## 33                         Wells St & Huron St        0.5398817
## 34                       Sawyer Ave & 111th St        0.5290665
## 35                     Washtenaw Ave & Polk St        0.5261866
## 36                 Public Rack - Troy & 111th St        0.5127577
```

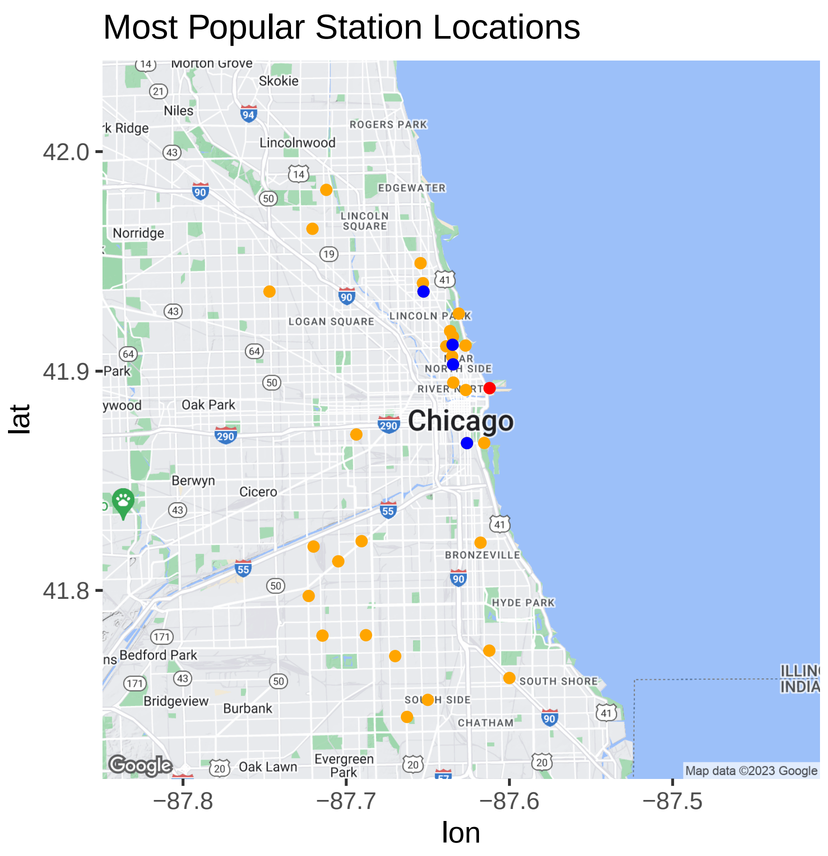
Now, let's view the super stations on a map to get a sense of their geographic distribution. First, we'll make a data station with the locations of the ranked casual and member starting stations:

```r
ranked_casual_locations <- list(ranked_casual_starting_station %>%
               transform(id = start_station_id) %>%
               select(rank, id, fraction_of_trips),
             unique_station_lookup %>%
               select(id, lat, lng)) %>%
  reduce(inner_join, by = 'id')

ranked_member_locations <- list(ranked_member_starting_station %>%
               transform(id = start_station_id) %>%
               select(rank, id, fraction_of_trips),
             unique_station_lookup %>%
               select(id, lat, lng)) %>%
  reduce(inner_join, by = 'id')
```

Now, we use the ggmap package to plot the location of the super stations on the map:

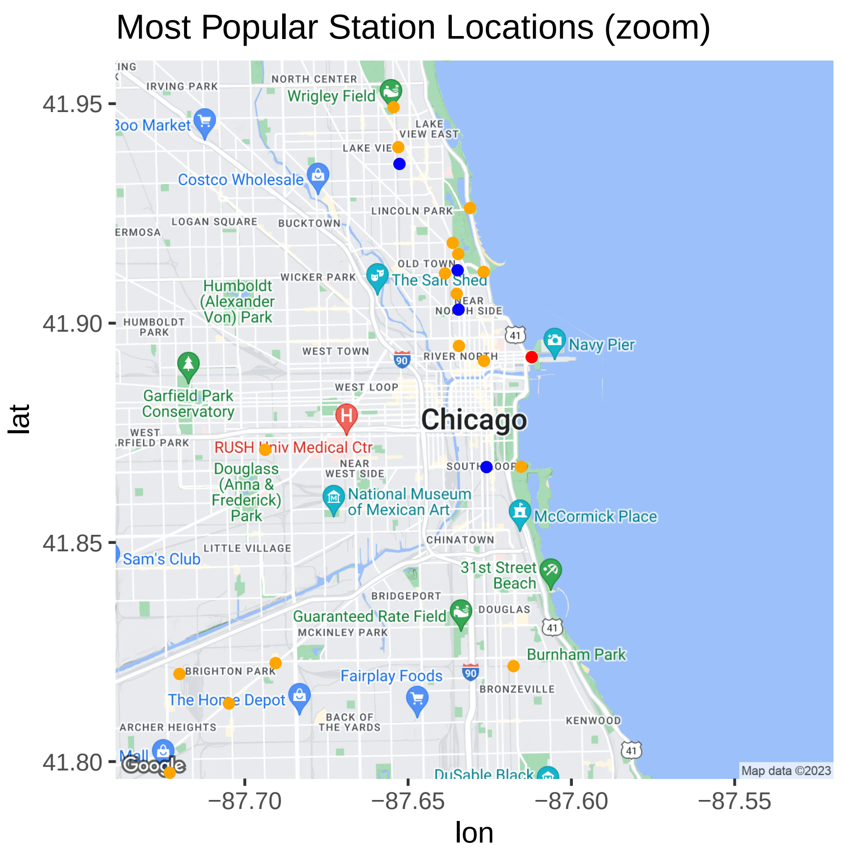
```r
# NB: Use of ggmap requires obtention and input of an API key from Google
# NB: I know that the aesthetics (in the more general, non-ggplot sense) of this
# map need work, but time constraints require me to move on
# get map of Chicago area from Google Maps
chicago_map <- get_googlemap("Chicago", zoom = 11)
ggmap(chicago_map) +
  geom_point(data = ranked_casual_locations %>%
               filter(rank <= 36),
             aes(x = lng, y = lat), color = "orange") +
  geom_point(data = ranked_member_locations %>%
               filter(rank <= 5),
             aes(x = lng, y = lat), color = "blue") +
  geom_point(data = ranked_casual_locations %>%
               filter(rank == 1),
             aes(x = lng, y = lat), color = "red") +
  geom_point(data = ranked_member_locations %>%
               filter(rank == 1),
             aes(x = lng, y = lat), color = "purple") +
  labs(title = "Most Popular Station Locations")
```



Four of the member super stations (blue) are clustered in the north end of downtown; however, the highest ranked member super station (purple) is on the far southwest outskirts of the city – a residential area. On the other hand, the highest ranked casual station (red) is on Navy Pier, which is a major tourist attraction.

Let's zoom in a bit more on the downtown and waterfront areas:

```
# get zoomed-in map of Chicago area from Google Maps
chicago_map <- get_googlemap("Chicago", zoom = 12)
ggmap(chicago_map) +
  geom_point(data = ranked_casual_locations %>%
               filter(rank <= 36),
             aes(x = lng, y = lat), color = "orange") +
  geom_point(data = ranked_member_locations %>%
               filter(rank <= 5),
             aes(x = lng, y = lat), color = "blue") +
  geom_point(data = ranked_casual_locations %>%
               filter(rank == 1),
             aes(x = lng, y = lat), color = "red") +
  geom_point(data = ranked_member_locations %>%
               filter(rank == 1),
             aes(x = lng, y = lat), color = "purple") +
  labs(title = "Most Popular Station Locations (zoom)")
```



We find that casual super stations are more concentrated around the waterfront and waterfront parks, and many are located near tourist attractions like Navy Pier, Millennium Park, Wrigley Field, and the Miracle Mile. This is further evidence that a large fraction of casual users are tourists or locals engaged in leisure/entertainment activities.

### Ending Station Distribution

This analysis is incomplete without looking at the distribution of ending stations as well. Such an analysis might reveal, for example, that the primary ending stations for member trips are in residential areas and downtown business districts, while the primary ending stations for casual trips are at the waterfront or in the vicinity of downtown shopping/entertainment areas. This would further corroborate that a substantial portion of casual trips correspond to entertainment and tourism, while a substantial fraction of member trips correspond to weekday commutes.

However, in the interest of time, I will forgo this analysis.

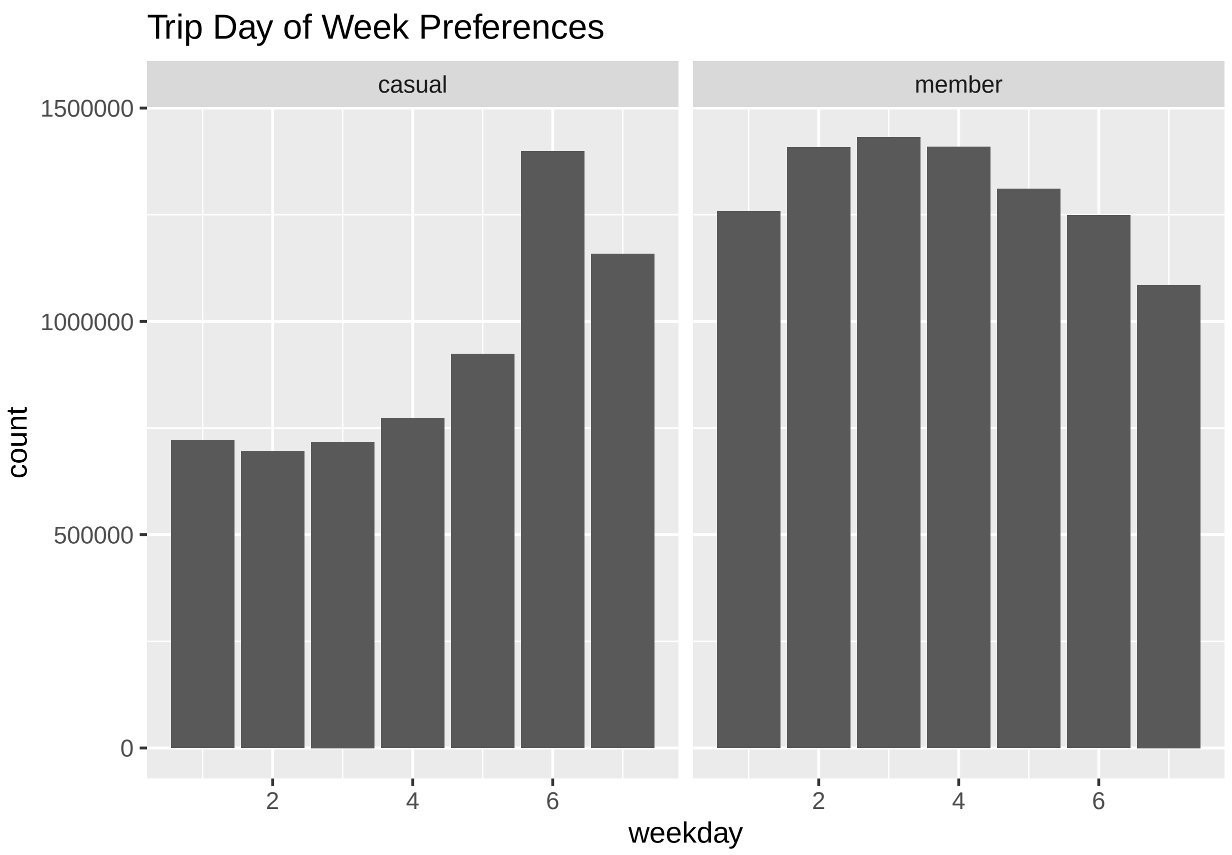
### Trip Distribution

We could also look at the unique pairs of starting and ending stations to perform a similar analysis on trips, instead of just starting and ending locations, but again I will forgo such an analysis in the interest of time.

### Ranking of Day of Week of Trip

Let's take a look at the distribution of weekdays that casual and member trips start on:

```r
ggplot(data = df_trips) +
  geom_bar(aes(x = weekday)) +
  facet_wrap(~member_casual) +
  labs(title = "Trip Day of Week Preferences")
```



There is an extremely clear difference in the distribution of weekdays between casual customers and annual members: Casual customers are weekend riders. Trips begin to increase on Friday (fifth day of the week according to our numbering) and increase more precipitously over the weekend, with Saturday the most common day for casual trips. Indeed, about 2.6 out of 6.4 million – or about 40% of – casual trips are weekend trips:

```r
nrow(
  df_trips %>%
    filter(member_casual == "casual" & (weekday == 6 | weekday == 7))
)
```

```
## [1] 2556647
```

If we include Friday, this increases to 3.5 million trips – or about 55%:

```r
nrow(
  df_trips %>%
    filter(member_casual == "casual" & (weekday ==5 | weekday == 6 | weekday == 7))
)
```

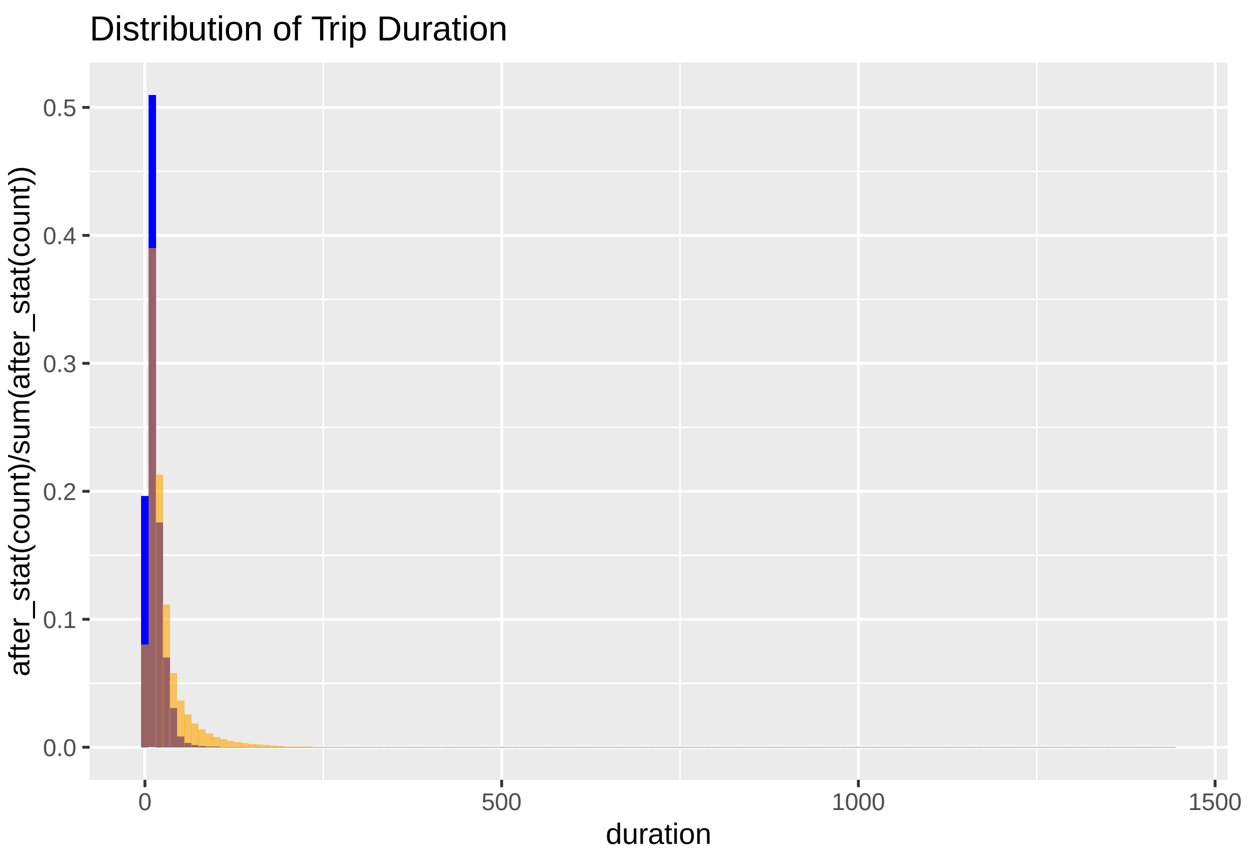
```
## [1] 3480451
```

On the other hand, the distribution of weekdays is relatively flat for annual members, with enhanced values for Tuesday through Thursday – the heart of the workweek – and somewhat diminished ridership on Sundays. This suggests that annual members are daily riders and many of them use Cyclistic bikes for their weekday commutes.

**Trip Duration Distribution**

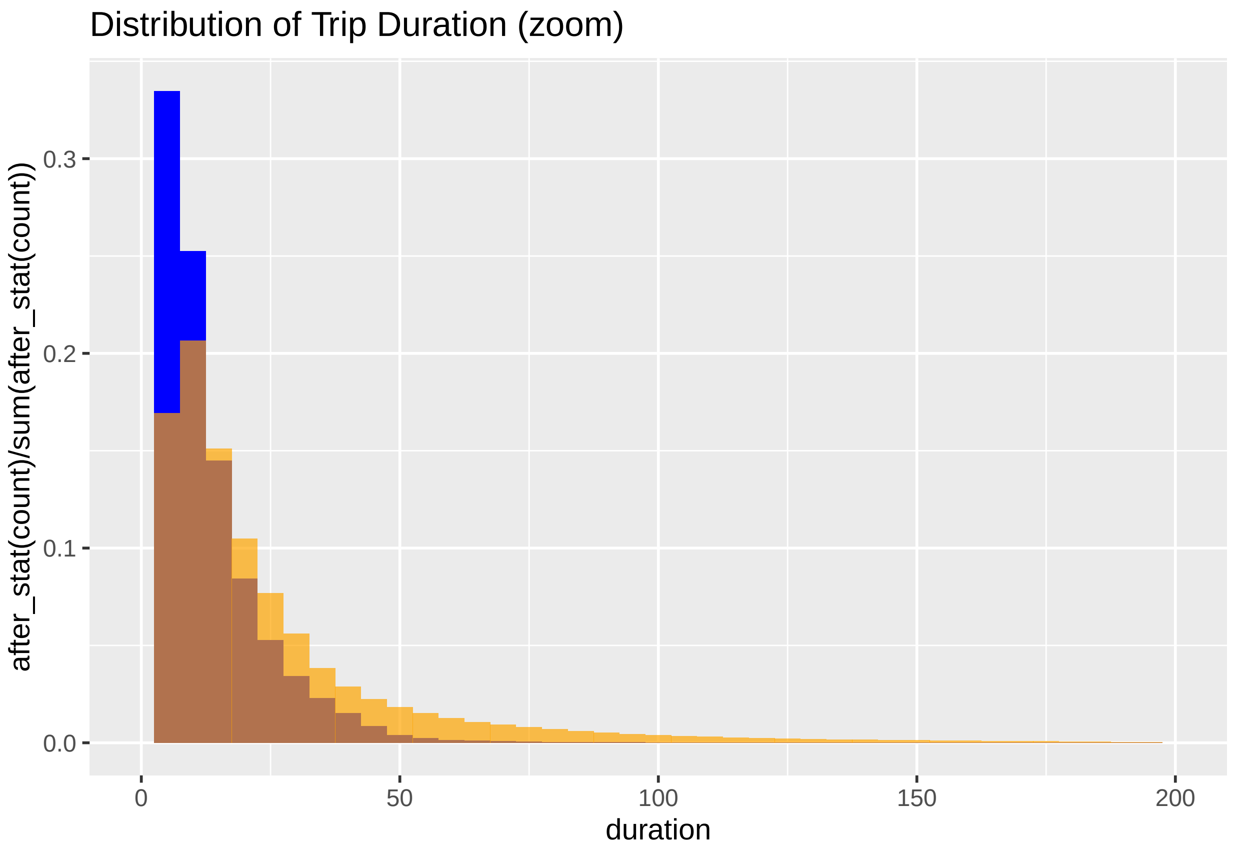
Let's use a histogram to look at the distribution of trip durations for member and casual trips. We want to normalize by the total number of trips of the appropriate customer type, so we will have to plot them separately, rather than using `facet_wrap`:

```r
ggplot() +
  geom_histogram(data = df_trips %>%
                   filter(member_casual == "member"),
                 aes(x = duration, y = after_stat(count)/sum(after_stat(count))),
                 binwidth = 10,
                 fill = "blue") +
  geom_histogram(data = df_trips %>%
                   filter(member_casual == "casual"),
                 aes(x = duration, y = after_stat(count)/sum(after_stat(count))),
                 binwidth = 10,
                 fill = "orange",
                 alpha = 0.6) +
  labs(title = "Distribution of Trip Duration")
```



Clearly, the distribution of member trips is more tightly clustered around short trips. Let's zoom in:

```r
ggplot() +
  geom_histogram(data = df_trips %>%
                   filter(member_casual == "member"),
                 aes(x = duration, y = after_stat(count)/sum(after_stat(count))),
                 binwidth = 5,
                 fill = "blue") +
  geom_histogram(data = df_trips %>%
                   filter(member_casual == "casual"),
                 aes(x = duration, y = after_stat(count)/sum(after_stat(count))),
                 binwidth = 5,
                 fill = "orange",
                 alpha = 0.7) +
  xlim(0,200) +
  labs(title = "Distribution of Trip Duration (zoom)")
```



It seems the most probable duration for member trips is quite short, only around five minutes. The most probable casual trip duration is a bit longer, between five and ten minutes, but still relatively short. Let's look at the average trip durations:

```
df_trips %>%
  filter(member_casual == "casual") %>%
  summarize(mean(duration))
```

```
##   mean(duration)
## 1       28.03591
```

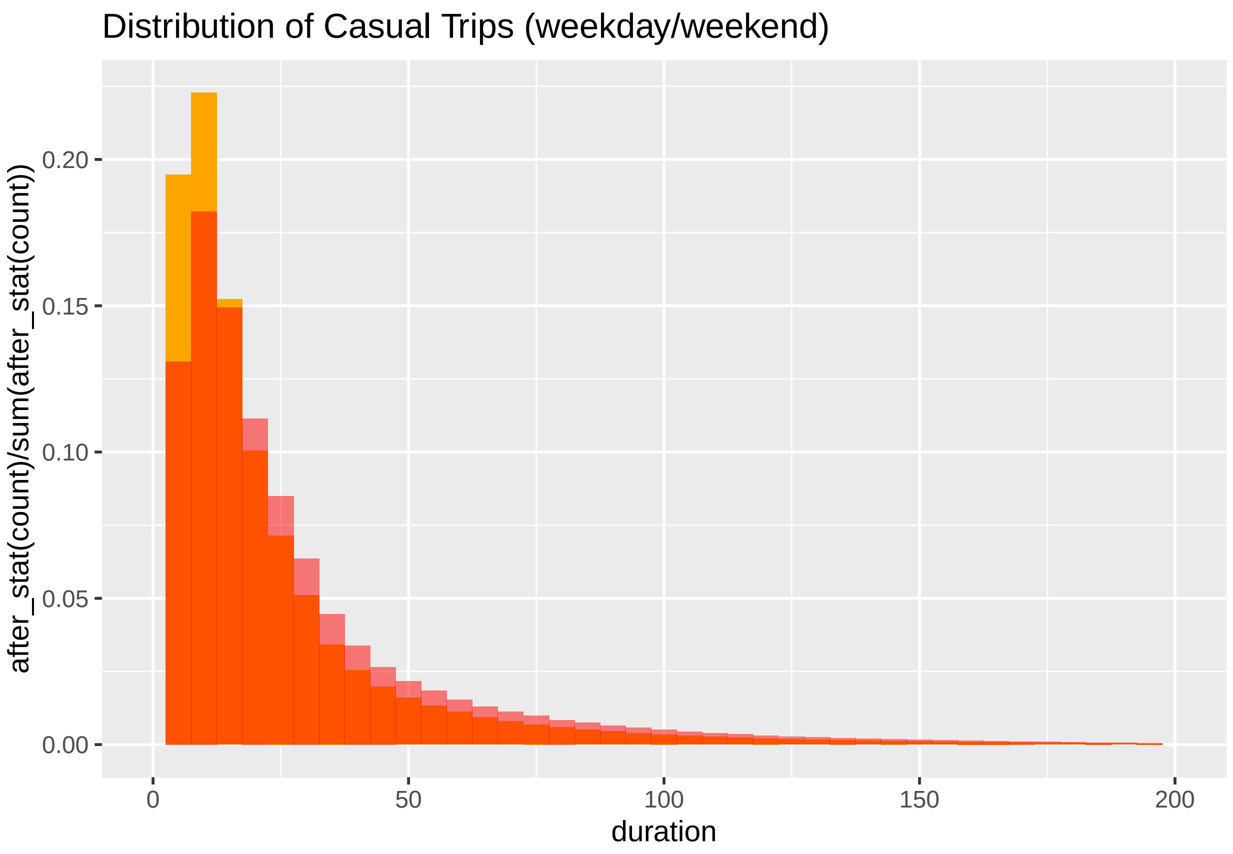
```
df_trips %>%
  filter(member_casual == "member") %>%
  summarize(mean(duration))
```

```
##   mean(duration)
## 1       13.45169
```

This difference is also borne out by the difference in average durations. The average casual trip lasts 28 minutes, while the average member trip lasts only 13 minutes. This further suggests that many members use the bikes for quick commuter trips, while casual riders use the bikes for longer leisure trips.

Let's look at how these distributions change if we further segregate by weekday/weekend:

```
ggplot() +
  geom_histogram(data = df_trips %>%
                   filter(member_casual == "casual" &
                            (weekday >= 1 & weekday <= 5)), # casual weekday trips
                 aes(x = duration, y = after_stat(count)/sum(after_stat(count))),
                 binwidth = 5,
                 fill = "orange") +
  geom_histogram(data = df_trips %>%
                   filter(member_casual == "casual" &
                            (weekday == 6 | weekday == 7)), # casual weekend trips
                 aes(x = duration, y = after_stat(count)/sum(after_stat(count))),
                 binwidth = 5,
                 fill = "red",
                 alpha = 0.5) +
  xlim(0,200) +
  labs(title = "Distribution of Casual Trips (weekday/weekend)")
```



Though it is not dramatic, the distribution of casual weekend trips is spread out towards longer trips than the corresponding distribution of casual weekday trips. Let's see how this is reflected in the mean trip durations:

```
df_trips %>%
  filter(member_casual == "casual" & (weekday >= 1 & weekday <= 5)) %>%
  summarize(mean(duration))
```

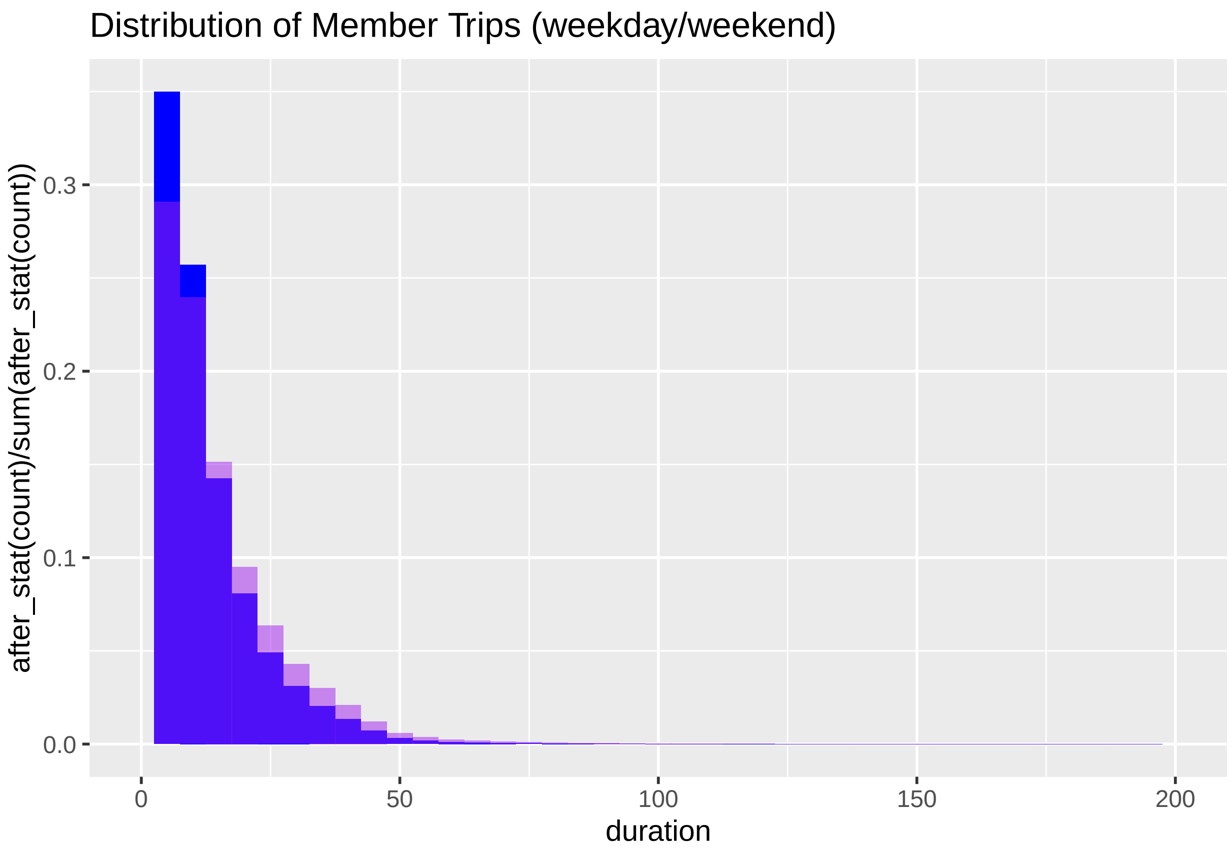
```
##   mean(duration)
## 1       25.76437
```

```r
df_trips %>%
  filter(member_casual == "casual" & (weekday == 6 | weekday == 7)) %>%
  summarize(mean(duration))
```

```
##   mean(duration)
## 1       31.44104
```

Indeed, we see that weekend casual riders tend to take longer trips, lasting more than 30 minutes on average. Let's repeat the analysis for member trips:

```r
ggplot() +
  geom_histogram(data = df_trips %>%
                   filter(member_casual == "member" &
                            (weekday >= 1 & weekday <= 5)), # casual weekday trips
                 aes(x = duration, y = after_stat(count)/sum(after_stat(count))),
                 binwidth = 5,
                 fill = "blue") +
  geom_histogram(data = df_trips %>%
                   filter(member_casual == "member" &
                            (weekday == 6 | weekday == 7)), # casual weekend trips
                 aes(x = duration, y = after_stat(count)/sum(after_stat(count))),
                 binwidth = 5,
                 fill = "purple",
                 alpha = 0.5) +
  xlim(0,200) +
  labs(title = "Distribution of Member Trips (weekday/weekend)")
```



Unsurprisingly, weekend member trips also tend to be longer:

```r
df_trips %>%
  filter(member_casual == "member" & (weekday >= 1 & weekday <= 5)) %>%
  summarize(mean(duration))
```

```
##   mean(duration)
## 1       12.84211
```

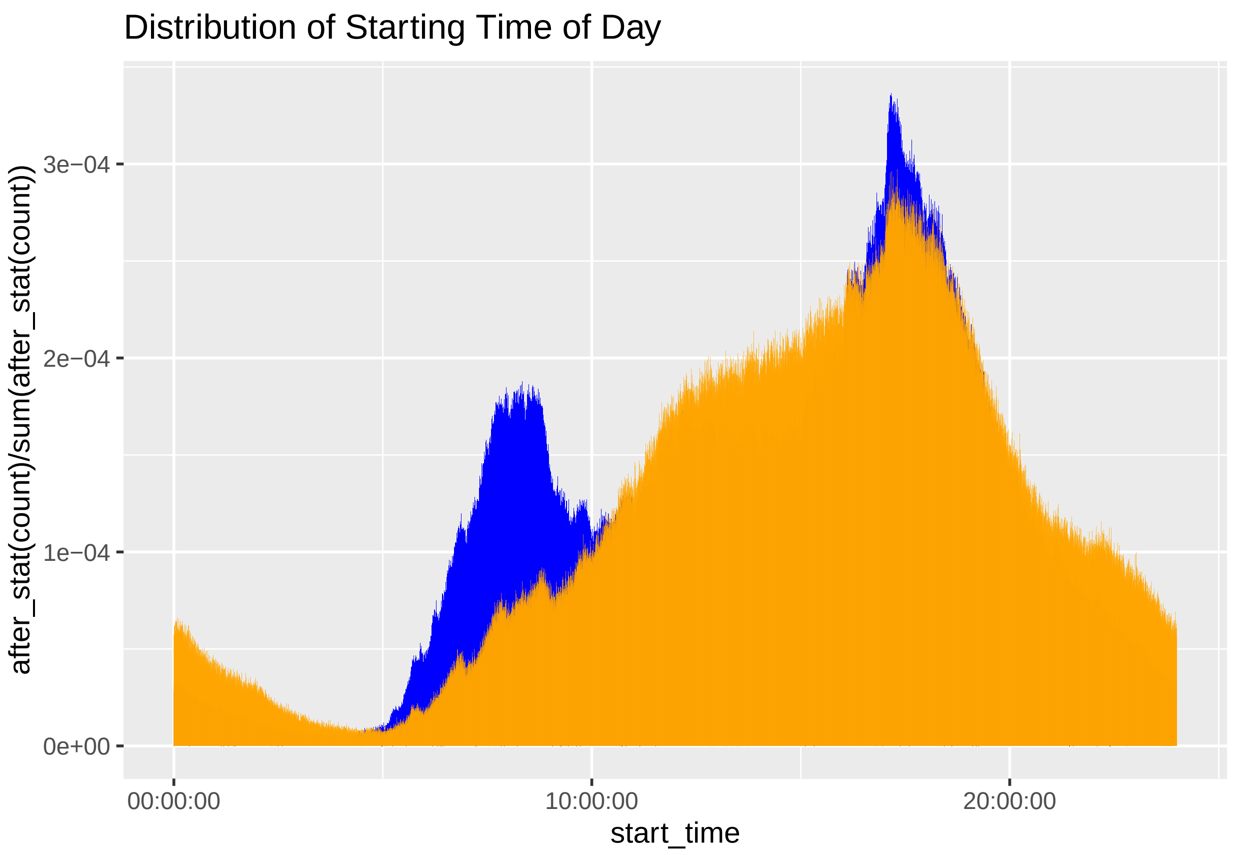
```r
df_trips %>%
  filter(member_casual == "member" & (weekday == 6 | weekday == 7)) %>%
  summarize(mean(duration))
```

```
##   mean(duration)
## 1       15.23211
```

This is a somewhat puzzling observation. Even weekend trips by annual members tend to be rather short, lasting only 15 minutes on average. Perhaps this reflects commutes to weekend jobs. However, we might in this case expect to see a bimodal distribution reflecting two distinct trip means for commuters and members taking leisure trips. This could also reflect a tendency of annual members to use bikes on the weekend for errands, which are pragmatic trips that likely tend to be shorter than leisure trips. Nonetheless, we conclude from these pronounced differences in the distributions of trip durations that annual members tend to take shorter trips, averaging 13 to 15 minutes in length, while casual riders tend to take more leisurely trips, lasting 26 to 31 minutes on average; this is further evidence that many members tend to rely on Cyclistic bikes for pragmatic trips such as commutes and errands, while casual customers tend to use Cyclistic bikes for leisure.

### Time of Day Distribution

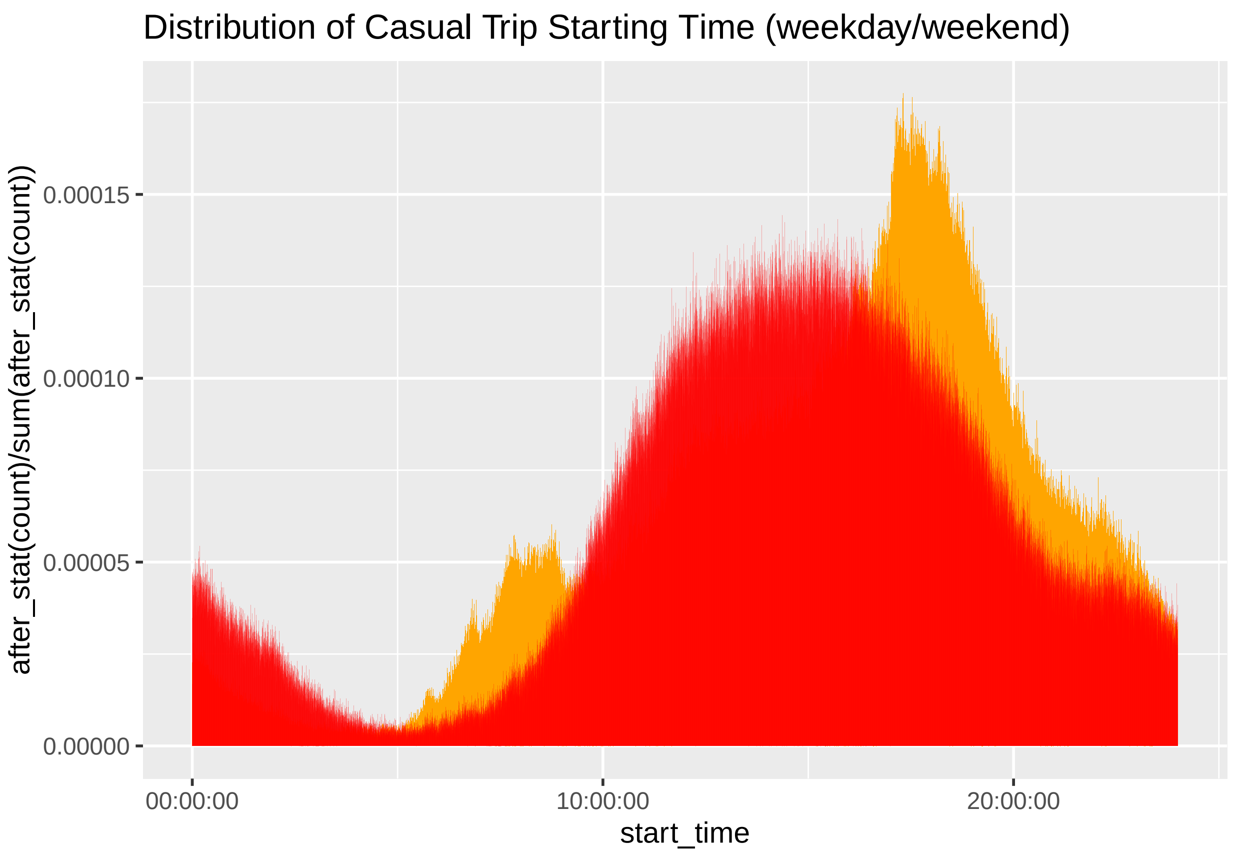
```r
ggplot() +
  geom_histogram(data = df_trips %>%
                   filter(member_casual == "member"),
                 aes(x = start_time, y = after_stat(count)/sum(after_stat(count))),
                 binwidth = 10,
                 fill = "blue") +
  geom_histogram(data = df_trips %>%
                   filter(member_casual == "casual"),
                 aes(x = start_time, y = after_stat(count)/sum(after_stat(count))),
                 binwidth = 10,
                 fill = "orange",
                 alpha = 0.7) +
  labs(title = "Distribution of Starting Time of Day")
```



This is an incredibly rich pair of distributions. The member start time distribution is trimodal, with the characteristic workday peaks at about 8 am, noon, and 5 pm. This is very strong evidence that commuting is the primary use of Cyclistic bikes by annual members.

The casual customer distribution shows no peak in the morning, a shoulder at noon, and a peak at about 5 pm, coincident with the evening peak in the annual member distribution. This suggests that there may still be the signal of commuter trips present in the casual distribution, but it is a sub-dominant contribution. Let’s split these distributions up into weekday and weekend distributions as we did for trip duration above:

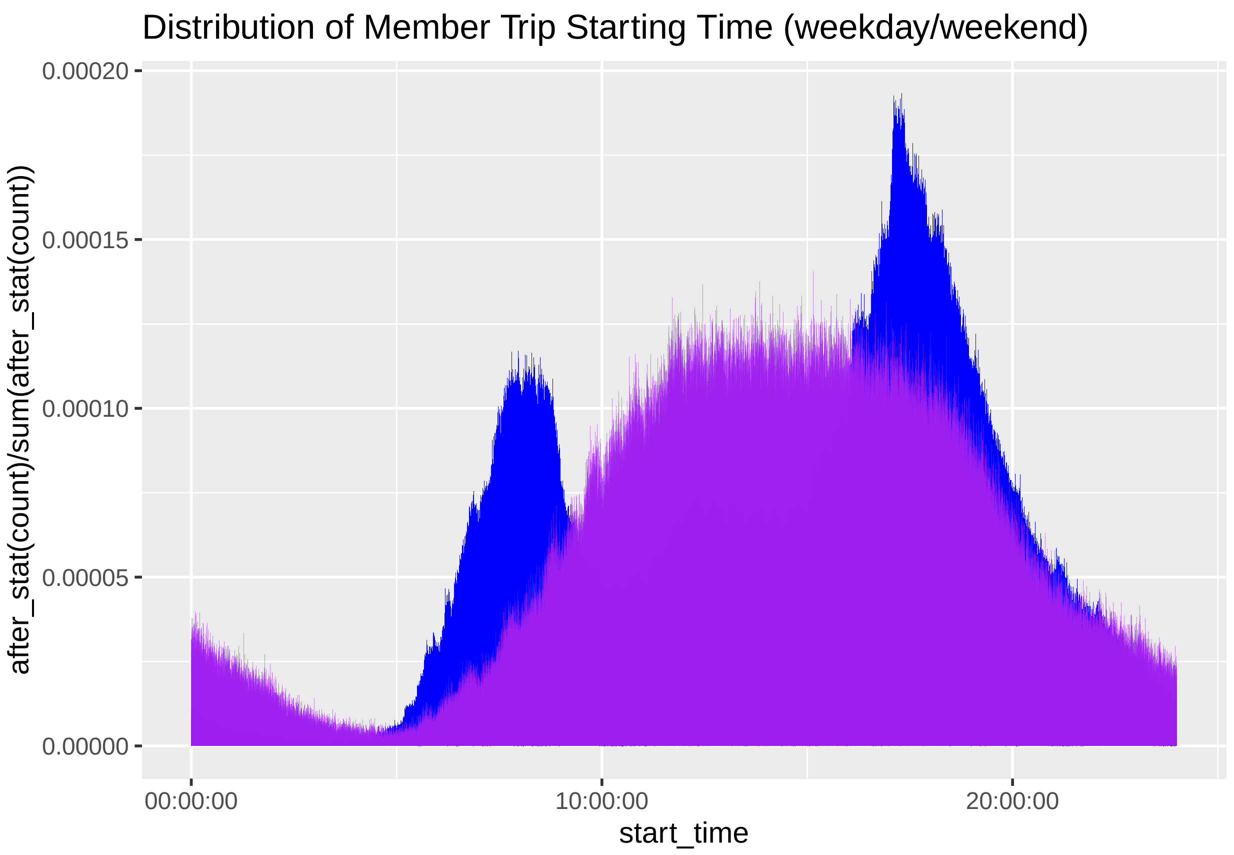
```
ggplot() +
  geom_histogram(data = df_trips %>%
                   filter(member_casual == "casual" &
                            (weekday >= 1 & weekday <= 5)), # casual weekday trips
                 aes(x = start_time, y = after_stat(count)/sum(after_stat(count))),
                 binwidth = 5,
                 fill = "orange") +
  geom_histogram(data = df_trips %>%
                   filter(member_casual == "casual" &
                            (weekday == 6 | weekday == 7)), # casual weekend trips
                 aes(x = start_time, y = after_stat(count)/sum(after_stat(count))),
                 binwidth = 5,
                 fill = "red",
                 alpha = 0.3) +
  labs(title = "Distribution of Casual Trip Starting Time (weekday/weekend)")
```



The weekday trips show the classic commuter pattern with peaks in the morning, at noon, and in the early evening. The weekend trips show only a broad peak centered around late morning / noon – indicative of daytime leisure trips. This is a clear signal that 1) casual weekday trips tend to be commutes, while casual weekend trips tend to be leisure trips, and 2) a majority of casual trips are weekend leisure trips.

Let's look at the breakdown for member trips:

```
ggplot() +
  geom_histogram(data = df_trips %>%
                   filter(member_casual == "member" &
                            (weekday >= 1 & weekday <= 5)), # casual weekday trips
                 aes(x = start_time, y = after_stat(count)/sum(after_stat(count))),
                 binwidth = 5,
                 fill = "blue") +
  geom_histogram(data = df_trips %>%
                   filter(member_casual == "member" &
                            (weekday == 6 | weekday == 7)), # casual weekend trips
                 aes(x = start_time, y = after_stat(count)/sum(after_stat(count))),
                 binwidth = 5,
                 fill = "purple",
                 alpha = 0.5) +
  labs(title = "Distribution of Member Trip Starting Time (weekday/weekend)")
```

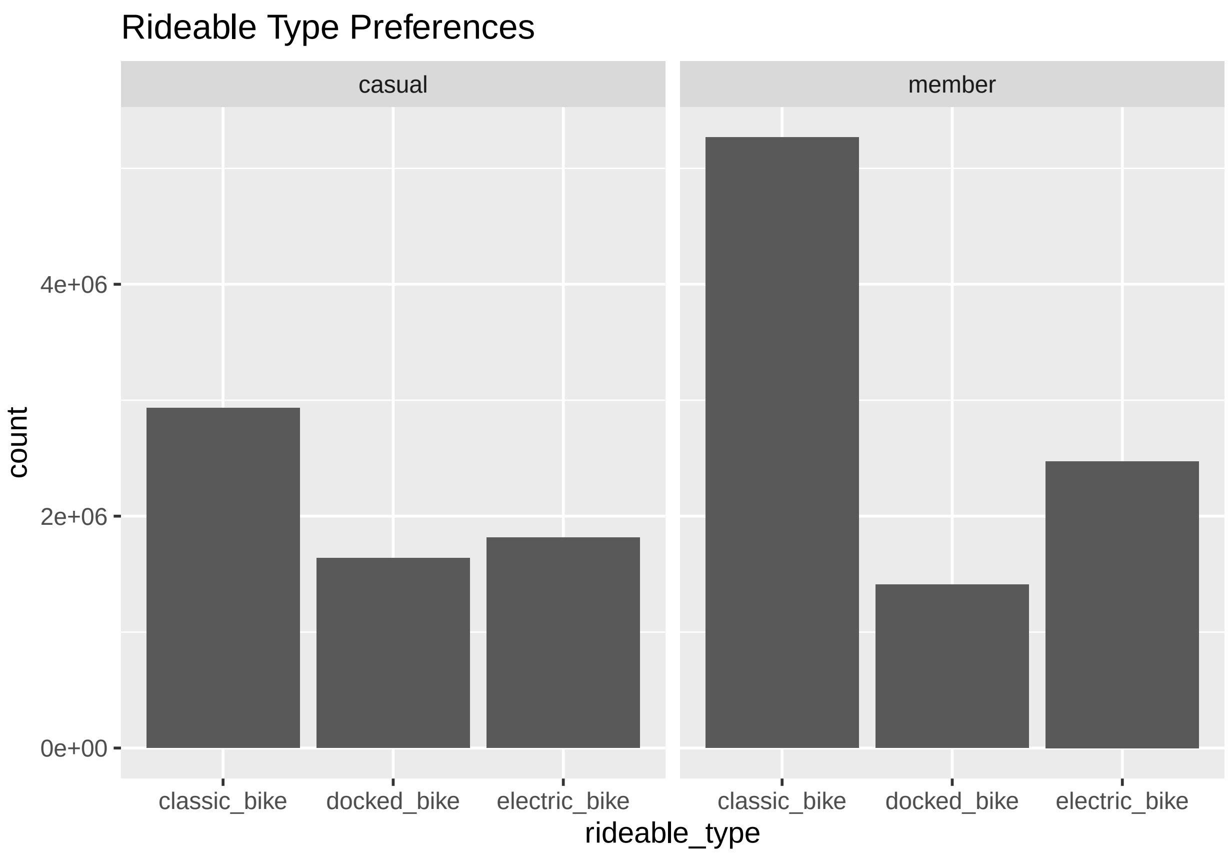


Weekend member trips also show a single broad peak centered around midday – though we recall that weekend member trips are only about half as long as weekend casual trips. Previously, we hypothesized that this is because members tend to use bikes to run errands on the weekend. An alternative hypothesis is that weekend casual customers tend to be tourists using bicycles for sight seeing as well as travel, while members are locals who use bikes on the weekend for both leisure and personal errands but tend to take short, direct routes.

### Ranking of Rideable Types

Let's look at casual customer and annual member rideable type preferences:

```r
ggplot(data = df_trips) +
  geom_bar(aes(x = rideable_type)) +
  facet_wrap(~member_casual) +
  labs(title = "Rideable Type Preferences")
```



We see that, whereas casual customers show a slight preference for classic bikes and little preference between electric and docked bikes (with perhaps a slight edge for electric bikes), members show a strong preference for classic bikes overall and a significant preference for electric bikes over docked bikes. The interpretation of this data is rendered impossible by the fact that we do not know what the 'docked_bike' type refers to, but we can certainly speculate freely. Perhaps, the strong preference of annual members for classic bikes indicates that they value cycling as a form of exercise, whereas the substantial fraction of casual customers who prefer electric bikes indicates that they often use the bikes as a more pragmatic means of leisurely intra-city transportation.

### Time Series of Total Monthly Ridership

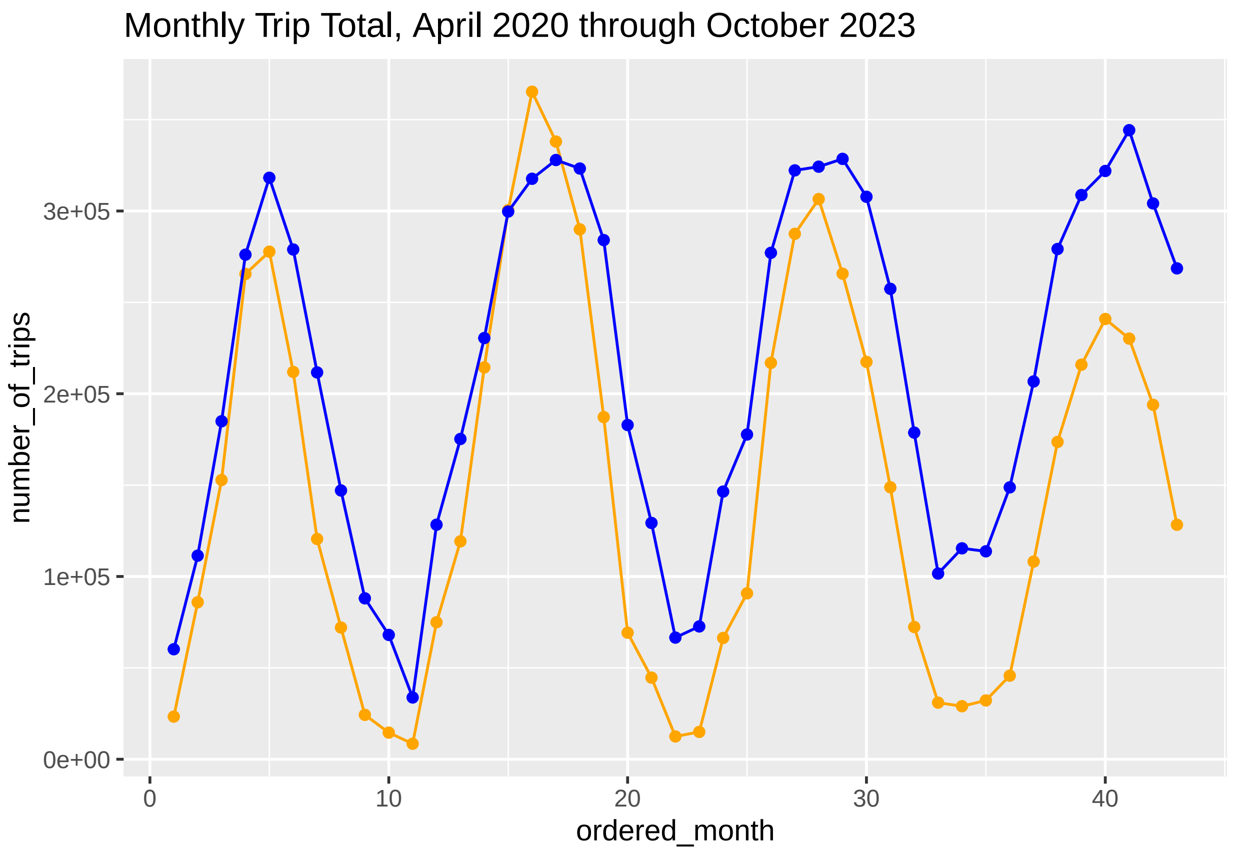
Finally, let's look at how the monthly total casual and member rides evolve in time over the period of our analysis (2020-04 through 2023-10):

```
total_casual_trips_by_month <- df_trips %>%
  filter(member_casual == "casual") %>%
  group_by(year, month) %>%
  count(month) %>%
  transform(number_of_trips = n) %>%
  select(year, month, number_of_trips)
total_casual_trips_by_month <- total_casual_trips_by_month %>%
  transform(ordered_month = 1:nrow(total_casual_trips_by_month)) %>%
  select(ordered_month, number_of_trips, year, month)
```

```
total_member_trips_by_month <- df_trips %>%
  filter(member_casual == "member") %>%
  group_by(year, month) %>%
  count(month) %>%
```

```r
  transform(number_of_trips = n) %>%
  select(year, month, number_of_trips)
total_member_trips_by_month <- total_member_trips_by_month %>%
  transform(ordered_month = 1:nrow(total_member_trips_by_month)) %>%
  select(ordered_month, number_of_trips, year, month)
```

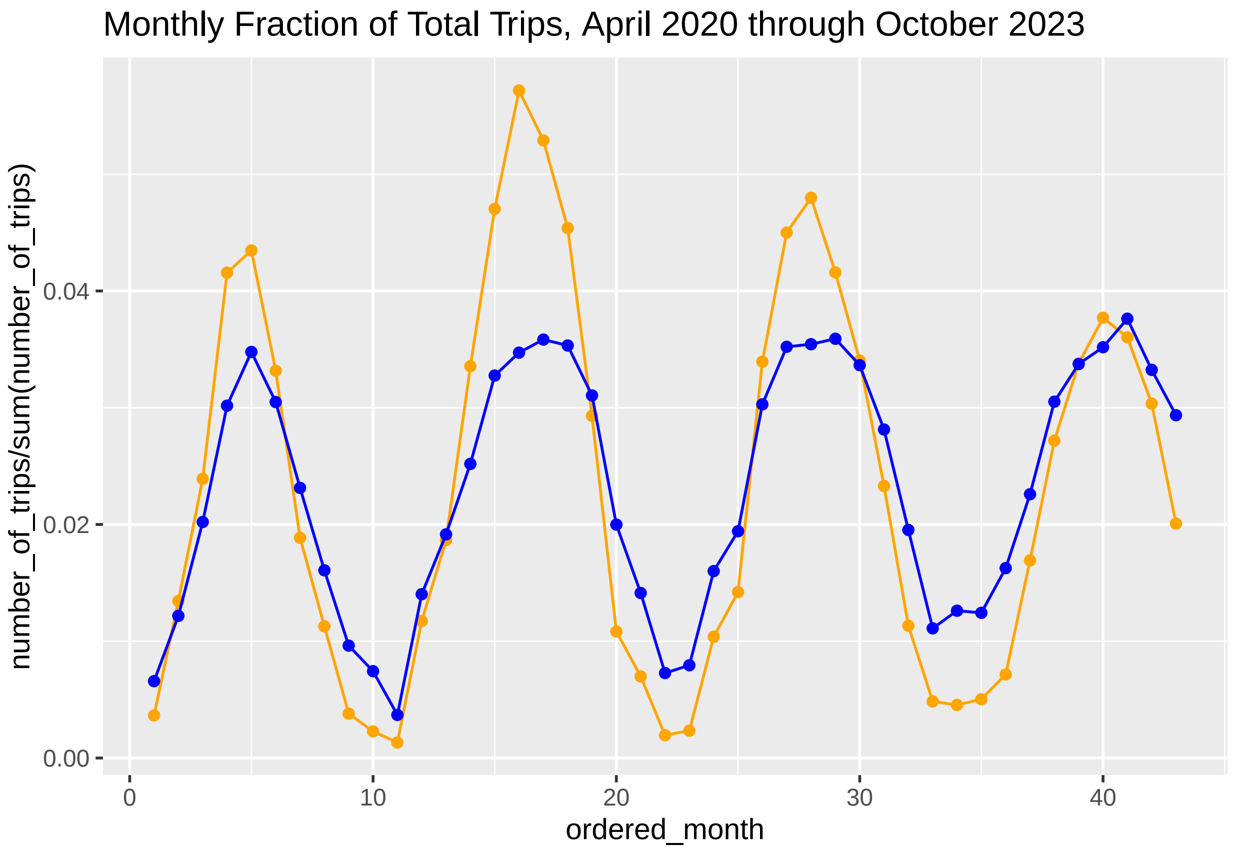
```r
ggplot() +
  geom_point(data = total_casual_trips_by_month,
             aes(x = ordered_month, y = number_of_trips),
             color = "orange") +
  geom_line(data = total_casual_trips_by_month,
             aes(x = ordered_month, y = number_of_trips),
             color = "orange") +
  geom_point(data = total_member_trips_by_month,
             aes(x = ordered_month, y = number_of_trips),
             color = "blue") +
  geom_line(data = total_member_trips_by_month,
             aes(x = ordered_month, y = number_of_trips),
             color = "blue") +
  labs(title = "Monthly Trip Total, April 2020 through October 2023")
```



Both casual and member trips show the anticipated summer/winter seasonal variation. The amplitude of the oscillation is larger for casual trips. Though annual member trips drop down to a minimum of 50 to 100 thousand monthly trips in the winter, casual trips drop down to only a few thousand to about 30 thousand a month in winter. This is a reflection of the significant leisure / tourist component of casual customer trips.

Let's check that the actual proportion of casual trips drops more significantly in winter months than the corresponding proportion of member trips. We do this by normalizing each curve by the total number of corresponding (member or casual) trips over the entire time period. This neglects the overall upper trend in the data indicative of the growth of the customer base, but this is acceptable because the signal of seasonal variation is much stronger than this upward trend.

```r
ggplot() +
  geom_point(data = total_casual_trips_by_month,
             aes(x = ordered_month, y = number_of_trips/sum(number_of_trips)),
             color = "orange") +
  geom_line(data = total_casual_trips_by_month,
             aes(x = ordered_month, y = number_of_trips/sum(number_of_trips)),
             color = "orange") +
  geom_point(data = total_member_trips_by_month,
             aes(x = ordered_month, y = number_of_trips/sum(number_of_trips)),
             color = "blue") +
  geom_line(data = total_member_trips_by_month,
             aes(x = ordered_month, y = number_of_trips/sum(number_of_trips)),
             color = "blue") +
  labs(title = "Monthly Fraction of Total Trips, April 2020 through October 2023")
```



Indeed, the fraction of casual trips has historically been much higher in summer and much lower in winter, indicative of the significant contribution of leisure and tourism to casual ridership. It is interesting to note that this was not true in the summer of 2023; in fact, the data suggests a pronounced downward trend in casual summer ridership over the last three years, and a less pronounced upward trend in winter ridership. The monthly distribution of casual trips appears to be becoming more similar to the monthly distribution of member trips – perhaps indicating a growing base of casual customers who use Cyclistic bikes for commuting.

## Summary and Recommendations

### A Summary of Key Observations

We list in this section the key observations of this analysis:

- About 30% of casual trips start at just 36 stations. These stations are clustered around downtown and the waterfront, and in particular around tourist and leisure attractions such as Navy Pier, Millennium Park, Wrigley Field, and the Miracle Mile.
- Casual ridership jumps on the weekends – 40% of trips start on Saturday or Sunday, and this number jumps to 55% if we include Friday.
- Members ridership remains relatively flat over the weekend, with a slight increase between Tuesday and Thursday indicative of weekday commutes and a slight drop on Sunday.
- Member trips last 13 (weekday) to 15 (weekend) minutes on average, while casual trips are significantly longer, lasting 26 to 31 minutes on average.
- The distribution of starting time of day for member trips clearly shows the telltale signs of workday commuters – with peaks around 8 am, noon, and 5 pm. This is true even without filtering out weekend rides, though weekend rides by themselves reflect unstructured daytime trips with a single broad peak centered around midday.
- The distribution of starting time of day casual weekday trips also shows evidence of workday commutes, though it is weaker than the corresponding signal in the member trip data. This indicates that a nonnegligible fraction of casual riders are weekday commuters.
- Seasonal patterns in casual use appear to be becoming more similar to those of annual members, with the proportion of summer trips decreasing and that of winter trips increasing. This possibly reflects an increase in the relative number of commuters among casual customers.
- Members show a strong preference for classic bikes, while casual customers show only a modest preference for such bikes.

### A Tale of Two Customers

- Annual members tend to take short, direct, pragmatic trips, and many of them are daily commuters, though they tend to use their bikes on the weekends as well for unstructured day trips
- There are two types of casual customers: 1) those who use Cyclistic bikes on summer weekends to travel to and between entertainment and tourist attractions – likely primarily tourists, and 2) those who use Cyclistic bikes for their daily commutes, and perhaps weekend excursions as well.

### Recommendations

Based on this analysis, my primary recommendation is to focus on converting that significant subpopulation of casual customers who use Cyclistic bikes for daily commutes into annual customers. This can likely be done by pointing out the associated savings (an annual membership is cheaper than paying for a day-pass for X days of the week) and convenience (no initial investment in a bicycle, no time or money associated with upkeep).

I have two secondary recommendations: there is likely another subpopulation of local casual customers who use Cyclistic bikes for weekend excursions around downtown and waterfront entertainment areas. These customers may be converted to annual members by selling a *lifestyle* associated with the brand. I hypothesize that annual members value the fitness and "green-ness" associated with regular bicycling while also appreciating the convenience of not having to store or maintain a bicycle themselves. There is likely a significant subpopulation of local casual customers that may be sold on this lifestyle.

The final secondary recommendation is to focus on the promotion of Cyclistic bicycles for summer tourism. Casual summer ridership has been dropping in absolute numbers (number of riders per month), but historically it seems this has been a prime use of casual passes. In addition to increased promotion, a possibility might be the creation of a special week or month pass design specifically for tourists.

**Remaining Questions and Future Analysis**

The key remaining questions relate to the reliable identification and quantification of user subpopulations with both casual and annual rider pools. In particular, it is crucial to understand – though not possible to determine with the current data – how many casual riders are 1) daily commuters, 2) local weekend/leisure riders, or 3) tourists. Further, it would be extremely useful to construct typical use patterns based on these customer types.

Unfortunately, it will be difficult or impossible to obtain this kind of data without violating customer privacy. Tourists could be identified either by voluntary self-identification as a tourist or by voluntary provision of a zip code. However, this still leaves the problem of associating all trips taken by a tourist purchasing a day pass with the tourism customer type.

A survey may be sent out to past casual customers, but this is only possible if casual customers willingly provide contact information such as an email address. Potentially, if a temporary card provided by the company is scanned, for example, each time a bike is retrieved by a casual customer, this could be a way to associate multiple trips with a single, randomly generated and fully anonymous customer ID.

The diminished number of casual summer trips is potentially concerning. The above monthly ridership time series plots indicate that it is not explained by a corresponding growth in member rides. Further context is needed – is this due to increased competition and hence a diminishing market share, overall tourism trends, or something else. One hypothesis: the increase between 2020 and 2021 in casual summer ridership indicates a greater use of bicycles as people are less willing to use public transport during the Covid pandemic; the decrease in casual summer rides over the last three years are a return to a more normal condition.

Some more minor points regarding analysis of the available data: * It turns out the number of distinct casual starting stations (1304) exceeds the number of distinct member starting stations (1239); it might be interesting to look at the stations from which member trips never emanate and try to determine why this is. * Differences in use patterns between casual customers and annual members can only be completely understood if, in addition to the distribution of starting station popularity and location, ending stations and trips (unique pairs of starting and ending stations) are also similary analysed.