Cyclistic Case Study

Sawandi Kirby

2024-12-03

Contents

Cyclistic - A bike ride share company

Ask:

What are the differences between Annual Members vs Casual Riders and what strategies can be used to convert Casual Riders into Annual Members?

With these insights we can create business campaigns based on Time, Day, Month, Distance, Duration and Frequency.

Prepare:

This is a public data set provided by Google Coursera in order to complete the capstone project. A case study based on a fictional business call Cyclistic, a bike ride share company that stores data about their rideshares here.

The data is formatted as .csv (comma separated) and can be downloaded based on month or operating quarter. The data consist of 13 columns. Column Names(ride_id, rideable_type, started_at, ended_at, start_station_id, start_station_name, end_station_id, end_station_name, start_lat, start_lng, end_lat, end_lng and member_casual).

Being that this data is provided to complete the case study by Google, we can assume that this data is credible with no major issues or biases.

Getting RStudio prepared

Setting up my environment.

Note: installing packages tidyverse, here, janitor, dplyr and others.

Note: Load Packages into the library every time you restart session.

```
library(tidyverse)
```

```
## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
## ✔ dplyr     1.1.4     ✔ readr     2.1.5
## ✔ forcats   1.0.0     ✔ stringr   1.5.1
## ✔ ggplot2   3.5.1     ✔ tibble    3.2.1
## ✔ lubridate 1.9.3     ✔ tidyr     1.3.1
## ✔ purrr     1.0.2
## ── Conflicts ────────────────────────────────────────────────────────────────
tidyverse_conflicts() ──
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
## ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all
conflicts to become errors
```

```
library(tidyr)
library(janitor)

##
## Attaching package: 'janitor'
##
## The following objects are masked from 'package:stats':
##
##     chisq.test, fisher.test

library(dplyr)
library(lubridate)
library(leaflet)
library(geosphere)
library(ggplot2)
library(ggmap)

## ℹ Google's Terms of Service: <https://mapsplatform.google.com>
##   Stadia Maps' Terms of Service:
<https://stadiamaps.com/terms-of-service/>
##   OpenStreetMap's Tile Usage Policy:
<https://operations.osmfoundation.org/policies/tiles/>
## ℹ Please cite ggmap if you use it! Use `citation("ggmap")` for details.
```

Create function to apply trimws to character columns of a data frame

```
trimws_df <- function(df) {
  char_cols <- sapply(df, is.character)
  df[char_cols] <- lapply(df[char_cols], trimws)
  return(df)
}
```

Load System environment file

Note: System environment file can be used to hide passwords and auth keys for privacy.

Use `Sys.getenv` to load Stadiamaps and Google Auth Keys

```
register_stadiamaps(key = Sys.getenv("STADIA_MAPS_KEY"))
register_google(key = Sys.getenv("GOOGLE_KEY"))
```

Installing Data downloaded from database.

Files were downloaded as a .csv, so we will use the read_csv function.

Note: Getting to know the data, getting a view of the op rows.

Note: Finding out what columns name are and what the rows consist of.

```
X202301_divvy_tripdata <- read_csv("Cyclistic Case Study
files/202301-divvy-tripdata.csv")

## Rows: 190301 Columns: 13
## ── Column specification
```

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

```
X202302_divvy_tripdata <- read_csv("Cyclistic Case Study
files/202302-divvy-tripdata.csv")
```

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

```
X202303_divvy_tripdata <- read_csv("Cyclistic Case Study
files/202303-divvy-tripdata.csv")
```

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

```
X202304_divvy_tripdata <- read_csv("Cyclistic Case Study
files/202304-divvy-tripdata.csv")
```

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

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

```
X202305_divvy_tripdata <- read_csv("Cyclistic Case Study
files/202305-divvy-tripdata.csv")
```

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

```
X202306_divvy_tripdata <- read_csv("Cyclistic Case Study
files/202306-divvy-tripdata.csv")
```

```
## Rows: 719618 Columns: 13
## ── Column specification
──────────────────────────────────────────────────────────────────────────────
────
## Delimiter: ","
## chr  (7): ride_id, rideable_type, start_station_name, start_station_id,
end_station...
## dbl  (4): start_lat, start_lng, end_lat, end_lng
## dttm (2): started_at, ended_at
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this
message.
```

```
X202307_divvy_tripdata <- read_csv("Cyclistic Case Study
files/202307-divvy-tripdata.csv")
```

```
## Rows: 767650 Columns: 13
## ── Column specification
```

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

```
X202308_divvy_tripdata <- read_csv("Cyclistic Case Study
files/202308-divvy-tripdata.csv")
```

```
## Rows: 771693 Columns: 13
## ── Column specification
──────────────────────────────────────────────────────────────────────────────
────
## Delimiter: ","
## chr  (7): ride_id, rideable_type, start_station_name, start_station_id,
end_station...
## dbl  (4): start_lat, start_lng, end_lat, end_lng
## dttm (2): started_at, ended_at
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this
message.
```

```
X202309_divvy_tripdata <- read_csv("Cyclistic Case Study
files/202309-divvy-tripdata.csv")
```

```
## Rows: 666371 Columns: 13
## ── Column specification
──────────────────────────────────────────────────────────────────────────────
────
## Delimiter: ","
## chr  (7): ride_id, rideable_type, start_station_name, start_station_id,
end_station...
## dbl  (4): start_lat, start_lng, end_lat, end_lng
## dttm (2): started_at, ended_at
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this
message.
```

```
X202310_divvy_tripdata <- read_csv("Cyclistic Case Study
files/202310-divvy-tripdata.csv")
```

```
## Rows: 537113 Columns: 13
## ── Column specification
```

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

```
X202311_divvy_tripdata <- read_csv("Cyclistic Case Study
files/202311-divvy-tripdata.csv")
```

```
## Rows: 362518 Columns: 13
## ── Column specification
──────────────────────────────────────────────────────────────────────
────
## Delimiter: ","
## chr  (7): ride_id, rideable_type, start_station_name, start_station_id,
end_station...
## dbl  (4): start_lat, start_lng, end_lat, end_lng
## dttm (2): started_at, ended_at
##
## ℹ Use `spec()` to retrieve the full column specification for this data.
## ℹ Specify the column types or set `show_col_types = FALSE` to quiet this
message.
```

```
Cyclistic_2023 <- bind_rows(list(X202301_divvy_tripdata,
X202302_divvy_tripdata, X202303_divvy_tripdata, X202304_divvy_tripdata,
X202305_divvy_tripdata, X202306_divvy_tripdata, X202307_divvy_tripdata,
X202308_divvy_tripdata, X202309_divvy_tripdata, X202310_divvy_tripdata,
X202311_divvy_tripdata), .id = "month")
```

Get to know the data.

Note: Use head() to get a view of the top rows

```
head(Cyclistic_2023)
```

```
## # A tibble: 6 × 14
##   month ride_id          rideable_type started_at          ended_at
##   <chr> <chr>            <chr>         <dttm>              <dttm>
## 1 1     F96D5A74A3E41399 electric_bike 2023-01-21 20:05:42 2023-01-21
20:16:33
## 2 1     13CB7EB698CEDB88 classic_bike  2023-01-10 15:37:36 2023-01-10
15:46:05
## 3 1     BD88A2E670661CE5 electric_bike 2023-01-02 07:51:57 2023-01-02
08:05:11
## 4 1     C90792D034FED968 classic_bike  2023-01-22 10:52:58 2023-01-22
```

```
11:01:44
## 5 1     3397017529188E8A classic_bike  2023-01-12 13:58:01 2023-01-12
14:13:20
## 6 1     58E68156DAE3E311 electric_bike 2023-01-31 07:18:03 2023-01-31
07:21:16
## # 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>
```

Note 14 rows returned, month added. 5,096,804 rows of data Find out what columns name are and what types of data are in them. Note: Use str function to accomplish this

```
str(Cyclistic_2023)
```

```
## spc_tbl_ [5,495,804 × 14] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ month             : chr [1:5495804] "1" "1" "1" "1" ...
##  $ ride_id           : chr [1:5495804] "F96D5A74A3E41399"
"13CB7EB698CEDB88" "BD88A2E670661CE5" "C90792D034FED968" ...
##  $ rideable_type     : chr [1:5495804] "electric_bike" "classic_bike"
"electric_bike" "classic_bike" ...
##  $ started_at        : POSIXct[1:5495804], format: "2023-01-21 20:05:42"
"2023-01-10 15:37:36" ...
##  $ ended_at          : POSIXct[1:5495804], format: "2023-01-21 20:16:33"
"2023-01-10 15:46:05" ...
##  $ start_station_name: chr [1:5495804] "Lincoln Ave & Fullerton Ave"
"Kimbark Ave & 53rd St" "Western Ave & Lunt Ave" "Kimbark Ave & 53rd St" ...
##  $ start_station_id  : chr [1:5495804] "TA1309000058" "TA1309000037"
"RP-005" "TA1309000037" ...
##  $ end_station_name  : chr [1:5495804] "Hampden Ct & Diversey Ave"
"Greenwood Ave & 47th St" "Valli Produce - Evanston Plaza" "Greenwood Ave &
47th St" ...
##  $ end_station_id    : chr [1:5495804] "202480.0" "TA1308000002" "599"
"TA1308000002" ...
##  $ start_lat         : num [1:5495804] 41.9 41.8 42 41.8 41.8 ...
##  $ start_lng         : num [1:5495804] -87.6 -87.6 -87.7 -87.6 -87.6 ...
##  $ end_lat           : num [1:5495804] 41.9 41.8 42 41.8 41.8 ...
##  $ end_lng           : num [1:5495804] -87.6 -87.6 -87.7 -87.6 -87.6 ...
##  $ member_casual     : chr [1:5495804] "member" "member" "casual" "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_character(),
##   ..   end_station_name = col_character(),
##   ..   end_station_id = col_character(),
```

```
##   ..   start_lat = col_double(),
##   ..   start_lng = col_double(),
##   ..   end_lat = col_double(),
##   ..   end_lng = col_double(),
##   ..   member_casual = col_character()
##   .. )
##  - attr(*, "problems")=<externalptr>
```

Use the `sum` and `is.na` functions combined to find the amount of **NA** values.*

*Note: The `sapply` funtion helps us to find which columns have the **NA** values in them.*

```
sapply(Cyclistic_2023, function(x) sum(is.na(x)))
```

```
##              month            ride_id      rideable_type
started_at
##                  0                  0                  0
0
##           ended_at start_station_name   start_station_id
end_station_name
##                  0             840006             840138
891278
##     end_station_id          start_lat          start_lng
end_lat
##             891419                  0                  0
6751
##            end_lng      member_casual
##               6751                  0
```

Step 1: Use the function we have created to trim rows of our data frame.

Note: The function will go through the data frame, find the character columns and trim any whitespace around the values off.

```
Cyclistic_2023 <- trimws_df(Cyclistic_2023)
```

Step 2: Use the `drop_na` function to remove any *NA* valuess from our **Cyclistic_2023** data frame.

Note: Using the `sapply`, `sum` and `is.na` functions agian we can verify the NA values have been removed.

```
Cyclistic_2023 <- drop_na(Cyclistic_2023)

sapply(Cyclistic_2023, function(x) sum(is.na(x)))
```

```
##              month            ride_id      rideable_type
started_at
##                  0                  0                  0
0
##           ended_at start_station_name   start_station_id
end_station_name
##                  0                  0                  0
0
```

```
##     end_station_id          start_lat          start_lng
end_lat
##                  0                  0                  0
0
##            end_lng      member_casual
##                  0                  0
```

*Note: 931,214 row lost due to `drop_na` Count of **NA** values returned as 0. We can assume there are no **NA** values in this data frame.*

4,164,564 rows returned ~18% loss

We verify the integrity of this data using the `sapply` + `is.na` functions combined to locate the NA values. Using the `str` function, we can see what data types are being stored in each column. Then we convert them as needed.

Also the head or glimpse functions give us a preview of the data to confirm changes.

Changing the data type of an entire column can result in some data loss returning NA values. We can infer that the data lost would not be accurate or usable. So we use drop.na again to remove those rows.

So we can expect there to be data loss from the (`ride_id`, `start_station_name`, `start_station_id`, `end_station_name` and `end_station_id`) because these columns will need to either be changed to numeric from character. Or changed to lower case letters to be read.

Lets review our data one more time

Note: Use `str` function

```
str(Cyclistic_2023)
```

```
## tibble [4,164,564 × 14] (S3: tbl_df/tbl/data.frame)
##  $ month             : chr [1:4164564] "1" "1" "1" "1" ...
##  $ ride_id           : chr [1:4164564] "F96D5A74A3E41399"
"13CB7EB698CEDB88" "BD88A2E670661CE5" "C90792D034FED968" ...
##  $ rideable_type     : chr [1:4164564] "electric_bike" "classic_bike"
"electric_bike" "classic_bike" ...
##  $ started_at        : POSIXct[1:4164564], format: "2023-01-21 20:05:42"
"2023-01-10 15:37:36" ...
##  $ ended_at          : POSIXct[1:4164564], format: "2023-01-21 20:16:33"
"2023-01-10 15:46:05" ...
##  $ start_station_name: chr [1:4164564] "Lincoln Ave & Fullerton Ave"
"Kimbark Ave & 53rd St" "Western Ave & Lunt Ave" "Kimbark Ave & 53rd St" ...
##  $ start_station_id  : chr [1:4164564] "TA1309000058" "TA1309000037"
"RP-005" "TA1309000037" ...
##  $ end_station_name  : chr [1:4164564] "Hampden Ct & Diversey Ave"
```

```
"Greenwood Ave & 47th St" "Valli Produce - Evanston Plaza" "Greenwood Ave &
47th St" ...
##  $ end_station_id    : chr [1:4164564] "202480.0" "TA1308000002" "599"
"TA1308000002" ...
##  $ start_lat         : num [1:4164564] 41.9 41.8 42 41.8 41.8 ...
##  $ start_lng         : num [1:4164564] -87.6 -87.6 -87.7 -87.6 -87.6 ...
##  $ end_lat           : num [1:4164564] 41.9 41.8 42 41.8 41.8 ...
##  $ end_lng           : num [1:4164564] -87.6 -87.6 -87.7 -87.6 -87.6 ...
##  $ member_casual     : chr [1:4164564] "member" "member" "casual" "member"
...
```

Process

Step 1: Lets handle the `chr` values in the `station_id` columns

Note: Use `as.numeric` on `start_station_id` and `end_station_id`.

```
Cyclistic_2023$start_station_id <-
as.numeric(Cyclistic_2023$start_station_id)

## Warning: NAs introduced by coercion

Cyclistic_2023$end_station_id <- as.numeric(Cyclistic_2023$end_station_id)

## Warning: NAs introduced by coercion
```

Step 2: Run the `drop_na` function for the *NAs* in the start_station_id and the end_station_id columns

Note: Record the amount of data lost.

```
Cyclistic_2023 <- drop_na(Cyclistic_2023)
```

Step 3: Next we are going to convert the month column and relpace the values with the corresponding month

Note: Weare going to use `as.numeric` on the month column.

```
Cyclistic_2023$month <- as.numeric(Cyclistic_2023$month)
```

Note: We received no warning message for the `month column`

Step 4: Next set the month names

Note: Make a list of the months and call it `month_names`

```
month_names <- c("January", "February", "March", "April", "May", "June",
"July", "August", "September", "October", "November")
```

Step 5: Then *mutate* the column with the month name list

Note: Use the `mutate` function to do this.

```
Cyclistic_2023 <- Cyclistic_2023%>%
  mutate(month= month_names[month])
```

Step 6: Format our columns to make them usable for graphs and correlations.

*Note: Make the date and time format **Y-M-D H:M:S***

```
Cyclistic_2023$started_at <- as.POSIXct(Cyclistic_2023$started_at, format =
"%Y-%m-%d %H:%M:%S")

Cyclistic_2023$ended_at <- as.POSIXct(Cyclistic_2023$ended_at, format =
"%Y-%m-%d %H:%M:%S")
```

Step 8: Now separate the date from the time

Note: Add them to a new column.

```
Cyclistic_2023$date <- as.Date(Cyclistic_2023$started_at)

Cyclistic_2023$end_date <- as.Date(Cyclistic_2023$ended_at)
```

Step 9: Find out the day of the week

Note: Use `wday` to find out what day of the week the dates correspond to.

```
Cyclistic_2023$day_of_week_start <- wday(Cyclistic_2023$date, label = TRUE)
Cyclistic_2023$day_of_week_end <-wday(Cyclistic_2023$end_date, label = TRUE)
```

Step 10: Then we are going to go through the `star_station_id` and `end_station_id` for *NA* values that may have been left over and filter them out.

Note: At the same time we are going to lower the characters in the `start_station_name` and `end_station_name` so that they are readable.

```
Cyclistic_2023 <- Cyclistic_2023 %>%
  mutate(start_station_name = tolower(start_station_name)) %>%
  filter(!is.na(start_station_name) & start_station_name != "") %>%
  mutate(end_station_name = tolower(end_station_name)) %>%
  filter(!is.na(end_station_name) & end_station_name != "")
```

Step 11: Format the **date** and **time** columns into Date/Time using `as.POSIXct` function.

Change the time column into integers with units *hours* and *mins* so that they can be usable for calculation.

*Note: Create a new column and call it **duration***

```
Cyclistic_2023$start_time <- as.POSIXct(Cyclistic_2023$started_at,
                                format = "%H:%M:%S", tz = "UTC", na.rm = TRUE)

Cyclistic_2023$end_time <- as.POSIXct(Cyclistic_2023$ended_at,
                               format = "%H:%M:%S",tz = "UTC", na.rm = TRUE)

Cyclistic_2023$duration_hrs <- as.integer(difftime(Cyclistic_2023$end_time,
```

```
Cyclistic_2023$start_time,
                                          units = "hours"))
Cyclistic_2023$duration_mins <- as.integer(difftime(Cyclistic_2023$end_time,
Cyclistic_2023$start_time,
                                           units = "min")) %% 60

Cyclistic_2023$duration <- Cyclistic_2023$duration_hrs +
Cyclistic_2023$duration_mins / 60
```

Step 12: Add a new column and call it `time_frames`

Note: Create a `function` to determine time frame. Then apply the `function` to create the `time_frame` column.

```
get_time_frame <- function(datetime) {
  hour <- hour(datetime)
  time_frame <- case_when(
    hour >= 0 & hour < 4 ~ "Twilight",
    hour >= 4 & hour < 7 ~ "Early Morning",
    hour >= 7 & hour < 10 ~ "Morning rush",
    hour >= 10 & hour < 14 ~ "Mid day",
    hour >= 14 & hour < 18 ~ "Afternoon",
    hour >= 18 & hour < 22 ~ "Evening",
    TRUE ~ "Night"
  )
  return(time_frame)
}
```

Ensure the time zone is correct for the started_at and ended_at columns.

Recalculate start_time and end_time based on the correct time zone.

```
Cyclistic_2023 <- Cyclistic_2023 %>%
  mutate(
    started_at = with_tz(started_at, tzone = "America/Chicago"),
    ended_at = with_tz(ended_at, tzone = "America/Chicago")
  )

Cyclistic_2023 <- Cyclistic_2023 %>%
  mutate(
    start_time = started_at,
    end_time = ended_at
  )
```

*Note: The time frames we are going to use are **Morning**, **Morning_rush**, **Mid_day**, **Evening_rush** and **Evening***

Step 13: Apply the function and create a new column.

Note: We only need one time frame column so we will only use it on the `start_time`

```
Cyclistic_2023 <- Cyclistic_2023 %>%
  mutate(time_frame = get_time_frame(start_time))
```

Step 14:Chicago = Central Daylight Timezone

Now that we have our time and dates separated, change time columns to reflect a 12hr, AM, PM, Central Daylight Timezone format.

```
# Extract only the time part in 12-hour format with AM/PM
Cyclistic_2023$start_time <- format(Cyclistic_2023$start_time, format =
"%I:%M:%S %p")
Cyclistic_2023$end_time <- format(Cyclistic_2023$end_time, format = "%I:%M:%S
%p")
```

Step 15: Now we are going to find the distance traveled between rides.

Note: First we want to create a `function` to normalize the longitude.

```
normalize_longitude <- function(lon) {
  while (any(lon > 180)) {
    lon[lon > 180] <- lon[lon > 180] - 360
  }
  while (any(lon < -180)) {
    lon[lon < -180] <- lon[lon < -180] + 360
  }
  return(lon)
}
```

Step 16: Normalize Longitude

Use the `normalize_longitude` function on the `Cyclistic_2023` data frame

Note: We only need to use it for the start longitude, `start_lng`

```
Cyclistic_2023$start_lng <- normalize_longitude(Cyclistic_2023$start_lng)
```

Step 17: Calculate Distance

Use `disHarversine` function to calculate distance between stations

Note: Call new column `distance_miles`

```
Cyclistic_2023$distance <- distHaversine(
  cbind(Cyclistic_2023$start_lng, Cyclistic_2023$start_lat),
  cbind(Cyclistic_2023$end_lng, Cyclistic_2023$end_lat))

Cyclistic_2023$distance_miles <- Cyclistic_2023$distance * 0.000621371
```

*Note: Multiply * the column distance by 0.000621371 to calculate distance in miles*

We can verify that our data is cleaned and ready to process based on the steps taken above. As a final step in verification we will use sapply(is.na()), str() and head() to view our cleaned data set.

Lets give our data one last clean check

```
sapply(Cyclistic_2023, function(x) sum(is.na(x)))
```

```
##              month            ride_id      rideable_type
started_at
##                  0                  0                  0
0
##           ended_at start_station_name   start_station_id
end_station_name
##                  0                  0                  0
0
##     end_station_id          start_lat          start_lng
end_lat
##                  0                  0                  0
0
##            end_lng      member_casual               date
end_date
##                  0                  0                  0
0
##  day_of_week_start    day_of_week_end         start_time
end_time
##                  0                  0                  0
0
##       duration_hrs      duration_mins           duration
time_frame
##                  0                  0                  0
0
##           distance     distance_miles
##                  0                  0
```

We can see that there are no NA values in any of the columns.

Now we verify that the columns have the correct data types.

```
str(Cyclistic_2023)
```

```
## tibble [977,373 × 26] (S3: tbl_df/tbl/data.frame)
##  $ month             : chr [1:977373] "January" "January" "January"
"January" ...
##  $ ride_id           : chr [1:977373] "F3344545150C3222"
"9DC70E5EE9D6A93F" "2B8E0781ED90C27C" "325B1DEA4641D815" ...
##  $ rideable_type     : chr [1:977373] "electric_bike" "electric_bike"
"electric_bike" "electric_bike" ...
##  $ started_at        : POSIXct[1:977373], format: "2023-01-09 13:11:35"
"2023-01-03 14:25:53" ...
##  $ ended_at          : POSIXct[1:977373], format: "2023-01-09 13:19:15"
"2023-01-03 14:35:50" ...
##  $ start_station_name: chr [1:977373] "broadway & waveland ave" "broadway
& waveland ave" "avondale ave & irving park rd" "komensky ave & 55th st" ...
##  $ start_station_id  : num [1:977373] 13325 13325 15624 347 632 ...
##  $ end_station_name  : chr [1:977373] "hampden ct & diversey ave" "hampden
ct & diversey ave" "campbell ave & irving park rd" "pulaski rd & 51st st" ...
##  $ end_station_id    : num [1:977373] 202480 202480 439 343 202480 ...
##  $ start_lat         : num [1:977373] 41.9 41.9 42 41.8 41.9 ...
```

```
##  $ start_lng         : num [1:977373] -87.6 -87.6 -87.7 -87.7 -87.7 ...
##  $ end_lat           : num [1:977373] 41.9 41.9 42 41.8 41.9 ...
##  $ end_lng           : num [1:977373] -87.6 -87.6 -87.7 -87.7 -87.6 ...
##  $ member_casual     : chr [1:977373] "member" "casual" "casual" "member"
...
##  $ date              : Date[1:977373], format: "2023-01-09" "2023-01-03"
...
##  $ end_date          : Date[1:977373], format: "2023-01-09" "2023-01-03"
...
##  $ day_of_week_start : Ord.factor w/ 7 levels "Sun"<"Mon"<"Tue"<..: 2 3 4
5 3 4 5 6 7 7 ...
##  $ day_of_week_end   : Ord.factor w/ 7 levels "Sun"<"Mon"<"Tue"<..: 2 3 4
5 3 4 5 6 7 7 ...
##  $ start_time        : chr [1:977373] "01:11:35 PM" "02:25:53 PM"
"11:03:25 AM" "01:36:18 AM" ...
##  $ end_time          : chr [1:977373] "01:19:15 PM" "02:35:50 PM"
"11:13:34 AM" "01:44:11 AM" ...
##  $ duration_hrs      : int [1:977373] 0 0 0 0 0 0 0 0 0 0 ...
##  $ duration_mins     : num [1:977373] 7 9 10 7 8 3 3 9 20 6 ...
##  $ duration          : num [1:977373] 0.117 0.15 0.167 0.117 0.133 ...
##  $ time_frame        : chr [1:977373] "Mid day" "Afternoon" "Mid day"
"Twilight" ...
##  $ distance          : num [1:977373] 2240 2244 3491 1113 2024 ...
##  $ distance_miles    : num [1:977373] 1.392 1.394 2.169 0.692 1.258 ...
```

And view the frame.

```
head(Cyclistic_2023)
```

```
## # A tibble: 6 × 26
##   month   ride_id          rideable_type started_at          ended_at
##   <chr>   <chr>            <chr>         <dttm>              <dttm>
## 1 January F3344545150C3222 electric_bike 2023-01-09 13:11:35 2023-01-09
13:19:15
## 2 January 9DC70E5EE9D6A93F electric_bike 2023-01-03 14:25:53 2023-01-03
14:35:50
## 3 January 2B8E0781ED90C27C electric_bike 2023-01-11 11:03:25 2023-01-11
11:13:34
## 4 January 325B1DEA4641D815 electric_bike 2023-01-12 01:36:18 2023-01-12
01:44:11
## 5 January A552D9830672F73A electric_bike 2023-01-17 06:46:15 2023-01-17
06:54:55
## 6 January 688620F6C9C8835B electric_bike 2023-01-18 01:51:13 2023-01-18
01:54:13
## # ℹ 21 more variables: start_station_name <chr>, start_station_id <dbl>,
## #   end_station_name <chr>, end_station_id <dbl>, start_lat <dbl>,
start_lng <dbl>,
## #   end_lat <dbl>, end_lng <dbl>, member_casual <chr>, date <date>,
end_date <date>,
## #   day_of_week_start <ord>, day_of_week_end <ord>, start_time <chr>,
end_time <chr>,
```

```
## #   duration_hrs <int>, duration_mins <dbl>, duration <dbl>, time_frame
<chr>,
## #   distance <dbl>, distance_miles <dbl>
```

Analyze:

We are going to count the amounts of various factors in this data set. Mainly the riders, stations, dates, days and time periods. This will give us hard numbers to analyze, calculate, and visualize.

At the end of our analysis we will go back and confirm that we have the appropriate data formats. Then we can see how the Annual Members and Casual Riders compare.

How do annual members and casual riders use Cyclistic bikes differently?

Step 1: Find out which users are members/casual and what there percentages make up?

Note: Use the division function to find out the decimal calculation and make it a Value we can use for the percentage.

```
table(Cyclistic_2023$member_casual)
```

```
##
## casual member
## 402024 575349
```

```
casual_count <- 402024
member_count <- 575349
total_count <- casual_count + member_count

casual_percentage <- casual_count / total_count * 100
member_percentage <- member_count / total_count * 100
```

Note: The Casual Rider consists of around 41% while the Annual Members consist of around 59% of the data set.

We can easily View this data using a **Pie Chart**

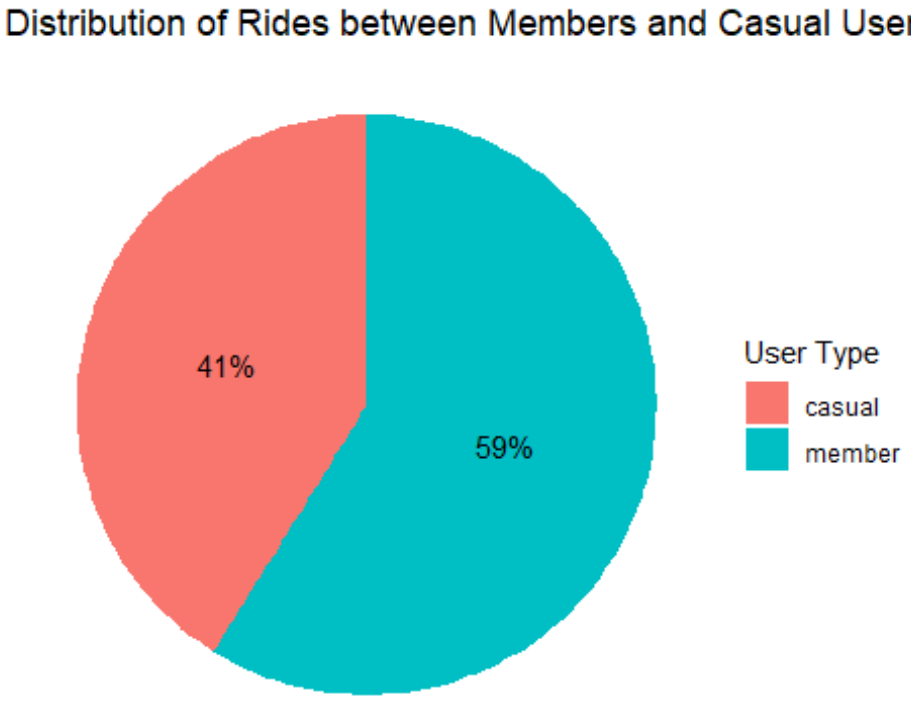
Note: Create a data frame with our total counts and percentages.

```
pie_chart_data <- data.frame(
  member_casual = c("casual", "member"),
  total_count = c(casual_count, member_count),
  percent = c(casual_percentage, member_percentage)
)
```

Note: Use `ggplot` and `geom_bar` functions for this chart.

```
ggplot(pie_chart_data, aes(x = "", y = total_count, fill = member_casual)) +
  geom_bar(stat = "identity", width = 1) +
```

```
  coord_polar("y", start = 0) +
  labs(fill = "User Type") +
  ggtitle("Distribution of Rides between Members and Casual Users") +
  geom_text(aes(label = paste0(round(percent), "%")), position =
position_stack(vjust = 0.5)) +
  theme_void()
```



Get info about Station location

Note: We are going to build a map to visualize the location we are working with.

```
Startstaion <-distinct(Cyclistic_2023, start_station_id, start_station_name,
.keep_all = TRUE) %>%
  select(start_station_id, start_station_name, start_lng, start_lat) %>%
  rename(station_id=start_station_id, station_name = start_station_name,
         station_lng =start_lng, station_lat = start_lat)
```

Note: Build data frame for Start stations.

```
Startstation_Counts <- Cyclistic_2023 %>%
  group_by(start_station_name) %>%
  summarise(startcount = n(), .groups = "drop") %>%
  rename(station_name = start_station_name)
```

Note: `group_by` and `summarize` the amount per station.

```
Startstaions <- full_join(Startstaion, Startstation_Counts) %>%
  distinct(station_id, station_name, .keep_all = TRUE)

## Joining with `by = join_by(station_name)`
```

Note: Next build a frame for the end stations.

```
Endstation <-distinct(Cyclistic_2023,end_station_id, end_station_name,
.keep_all = TRUE) %>%
  select(end_station_id, end_station_name, end_lng, end_lat) %>%
  rename(station_id=end_station_id, station_name = end_station_name,
         station_lng = end_lng, station_lat = end_lat)
```

Note: Complete same steps.

```
Endstation_Counts <- Cyclistic_2023 %>%
  group_by(end_station_name) %>%
  summarise(endcount = n(), .groups = "drop") %>%
  rename(station_name = end_station_name)
```

Note: Use `full_join` to combine the counts with the station information.

```
Endstaions <- full_join(Endstation, Endstation_Counts) %>%
  distinct(station_id, station_name, .keep_all = TRUE)

## Joining with `by = join_by(station_name)`
```

Note: `full_join the Startstations and Endstations data frames.

```
Station_lookup <- full_join(Startstaions, Endstaions) %>%
  distinct(station_id, station_name, .keep_all = TRUE)

## Joining with `by = join_by(station_id, station_name, station_lng,
station_lat)`
```

Finally use `drop_na` to remove any created NA values.

```
Station_lookup_cleaned <- drop_na(Station_lookup)
```

Note: Then Finally `bind_rows` to create the Station_lookup data frame.

Now let's remove the not needed data frames `Endstation` and `Startstation`

Also lets check out the `Station_lookup` data frame we created.

```
rm(Endstation, Startstaion, Endstation_Counts, Startstation_Counts)
```

Note: Use`head()`

```
head(Station_lookup_cleaned)

## # A tibble: 6 × 6
##   station_id station_name                  station_lng station_lat
startcount endcount
##        <dbl> <chr>                              <dbl>       <dbl>
<int>    <int>
## 1        347 komensky ave & 55th st              -87.7        41.8
84       63
## 2     202480 hampden ct & diversey ave           -87.6        41.9
1189     1240
## 3        661 evanston civic center               -87.7        42.1
878      874
## 4        343 pulaski rd & 51st st                -87.7        41.8
```

```
352      335
## 5        599 valli produce - evanston plaza       -87.7        42.0
370      307
## 6        431 western ave & grace st               -87.7        42.0
497      477
```

Lets calculate the station data into usable counts to find top performing stations.

Step 1: Get the total start and end counts call it totalcount.

```
Station_Count <- Station_lookup_cleaned %>%
  mutate(totalcount = startcount + endcount)
```

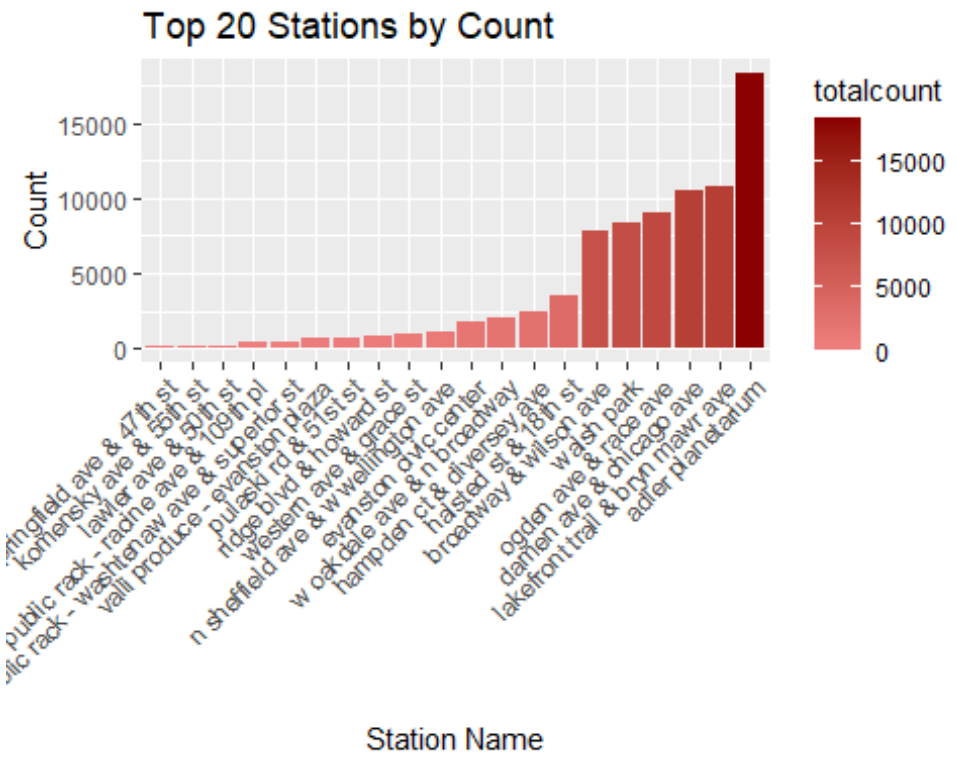
Step 2: Sort them with `arrange` and `desc` so we get the highest count to the top.

```
top_20_stations <- head(Station_Count, 20)
```

Let's view the Top 20 Stations

Note: Let's first view a bar chart going from lowset to highest.

```
ggplot(top_20_stations, aes(x = reorder(station_name, totalcount), y =
totalcount, fill = totalcount)) +
  geom_bar(stat = "identity") +
  labs(x = "Station Name", y = "Count") +
  ggtitle("Top 20 Stations by Count") +
  scale_fill_gradientn(colors = c("lightcoral", "darkred"), limits = c(0,
max(top_20_stations$totalcount))) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



Note: Lets see if the most traveled bike routes correspond with the top 20 most used stations.

```
Trip_routes <- Cyclistic_2023 %>%
  filter(start_lng != end_lng & start_lat != end_lat) %>%
  group_by(start_lng, start_lat, end_lng, end_lat, member_casual,
rideable_type) %>%
  summarise(total = n(), .groups = "drop") %>%
  filter(total >250)
```

Note: First filter out the trips that dont start and end at the same location

Note: Then filter casual riders and member riders.

```
Casual_trips <- Trip_routes %>%
  filter(member_casual == "casual")
head(Casual_trips)
```

```
## # A tibble: 6 × 7
##   start_lng start_lat end_lng end_lat member_casual rideable_type total
##       <dbl>     <dbl>   <dbl>   <dbl> <chr>         <chr>         <int>
## 1     -87.7      42.1   -87.7    42.0 casual        classic_bike    273
## 2     -87.6      41.9   -87.6    41.9 casual        classic_bike    344
## 3     -87.6      41.9   -87.6    41.9 casual        classic_bike    427
## 4     -87.6      41.9   -87.6    41.9 casual        classic_bike    459
## 5     -87.6      41.9   -87.6    41.9 casual        classic_bike    986
## 6     -87.6      41.9   -87.6    41.9 casual        classic_bike    270
```

```
Member_trips <- Trip_routes %>%
  filter(member_casual == "member")
head(Member_trips)
```

```
## # A tibble: 6 × 7
##   start_lng start_lat end_lng end_lat member_casual rideable_type total
##       <dbl>     <dbl>   <dbl>   <dbl> <chr>         <chr>         <int>
## 1     -87.7      41.9   -87.7    41.9 member        classic_bike    323
## 2     -87.7      41.9   -87.7    41.9 member        classic_bike    284
## 3     -87.7      41.9   -87.7    41.9 member        classic_bike    342
## 4     -87.7      42.0   -87.7    42.1 member        classic_bike    551
## 5     -87.7      42.1   -87.7    42.0 member        classic_bike    643
## 6     -87.7      41.9   -87.7    41.9 member        classic_bike    257
```

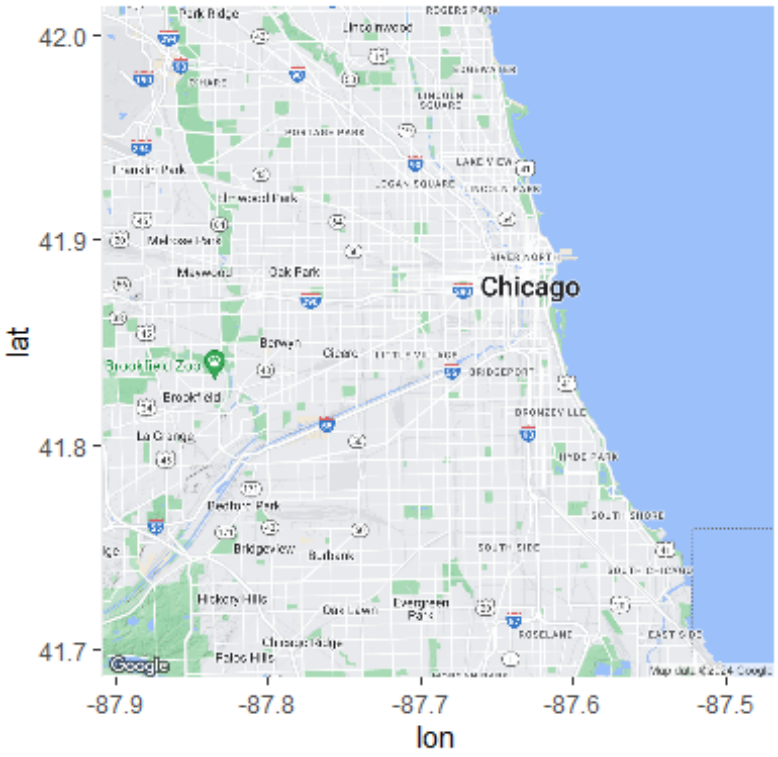
First a base map will allow us to have a better idea of the area we are working with.

*Note: Now that we have the **Station_lookup** with geo locations we can build a map to plot out the points.*

```
map <- get_googlemap(center = c(lon = mean(Station_lookup$station_lng),
                                lat = mean(Station_lookup$station_lat)),
                     width =20, height = 16, zoom = 11)
```

```
## ℹ
<https://maps.googleapis.com/maps/api/staticmap?center=41.850277,-87.689293&z
oom=11&size=640x640&scale=2&maptype=terrain&key=xxx>
```

```
ggmap(map)
```



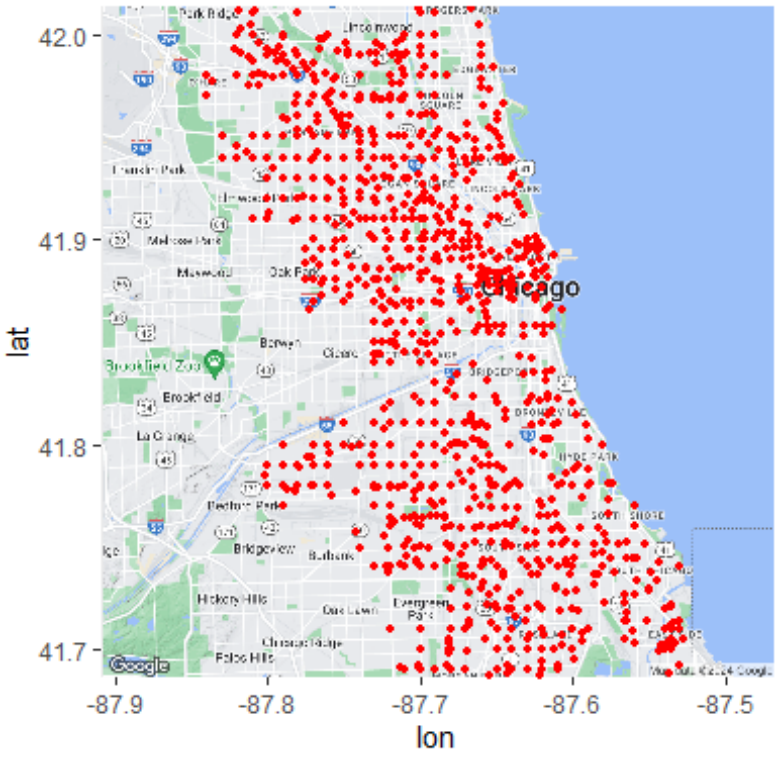
Note: Map Credit Google Maps

Let's get a general idea of all the station locations and how their scattered

Note: This map should give us a clear picture of the area we are working with and the cluster of bike stations in the `Station_Lookup` data frame.

```
ggmap(map) +
  geom_point(data = Station_lookup,
             aes(x = station_lng, y = station_lat),
             shape = "circle", color = "red", size = 1)
```

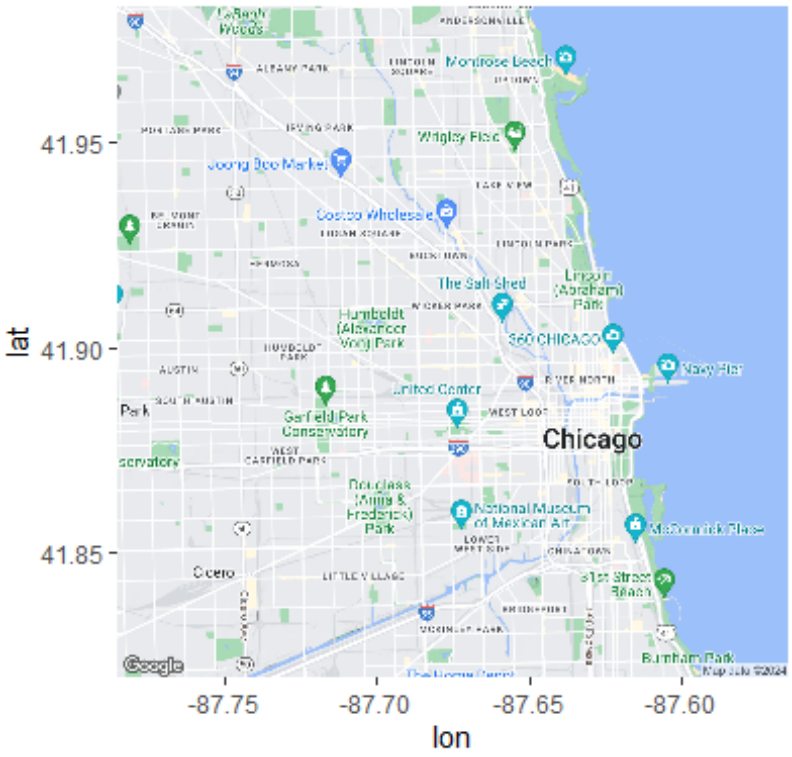
```
## Warning: Removed 50 rows containing missing values or values outside the
scale range
## (`geom_point()`).
```



First let's get a zoomed in base map of our Top 20 Stations

Call this new map `top_20_map`

```
top_20_map <- get_googlemap(center = c(lon =
mean(top_20_stations$station_lng),
                                       lat = mean(top_20_stations$station_lat)),
                        width =20, height = 16, zoom = 12)

## i
<https://maps.googleapis.com/maps/api/staticmap?center=41.901555,-87.675405&z
oom=12&size=640x640&scale=2&maptype=terrain&key=xxx>

ggmap(top_20_map)
```

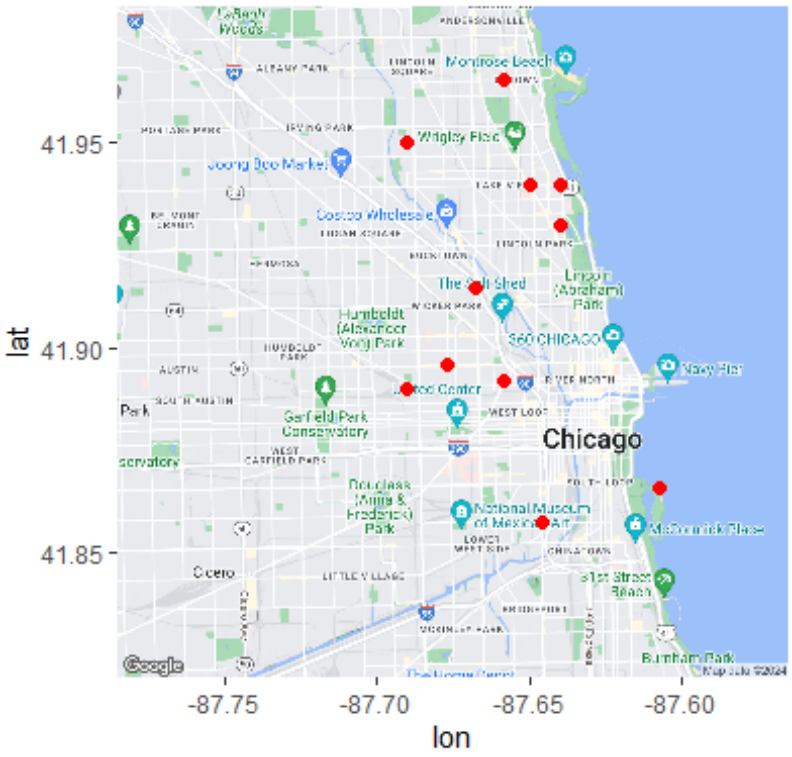


Now lets filter out our top 20 locations

Note: Use the top 20 stations data frame we created.

```
ggmap(top_20_map) +
  geom_point(data = top_20_stations,
             aes(x = station_lng, y = station_lat),
             shape = "circle", color = "red", size = 2)
```

```
## Warning: Removed 9 rows containing missing values or values outside the
scale range
## (`geom_point()`).
```



Now lets visualize the *Heatmap* of the top stations in relation to each other.

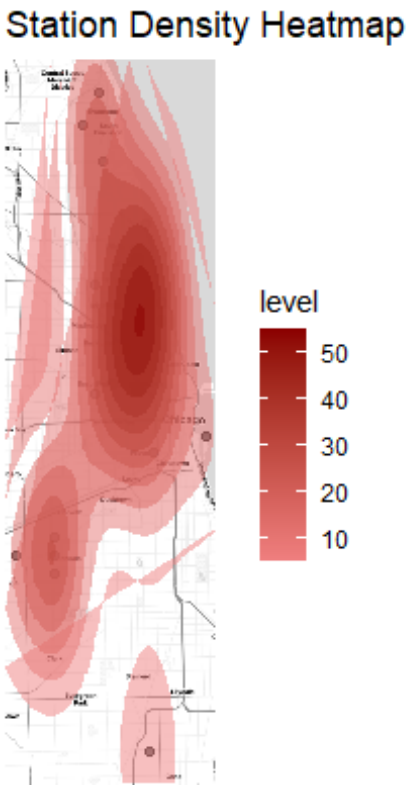
Note: We are going to use Stadia Maps for this visualization.

```
station_heatmap <- ggmap::qmplot(station_lng, station_lat, data =
top_20_stations, geom = "point", alpha = I(0.6), width = 20, zoom = 12) +
  stat_density_2d(aes(fill = ..level..), geom = "polygon", alpha = 0.5) +
  scale_fill_gradient(low = "lightcoral", high = "darkred") +
  labs(title = "Station Density Heatmap")
```

```
## ℹ © Stadia Maps © Stamen Design © OpenMapTiles © OpenStreetMap
contributors.
```

```
print(station_heatmap)
```

```
## Warning: The dot-dot notation (`..level..`) was deprecated in ggplot2
3.4.0.
## ℹ Please use `after_stat(level)` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
generated.
```



Let's see if the top 50 Casual Trip Routes correlate to the top 20 stations by count

Top 50 Casual Trip Routes.

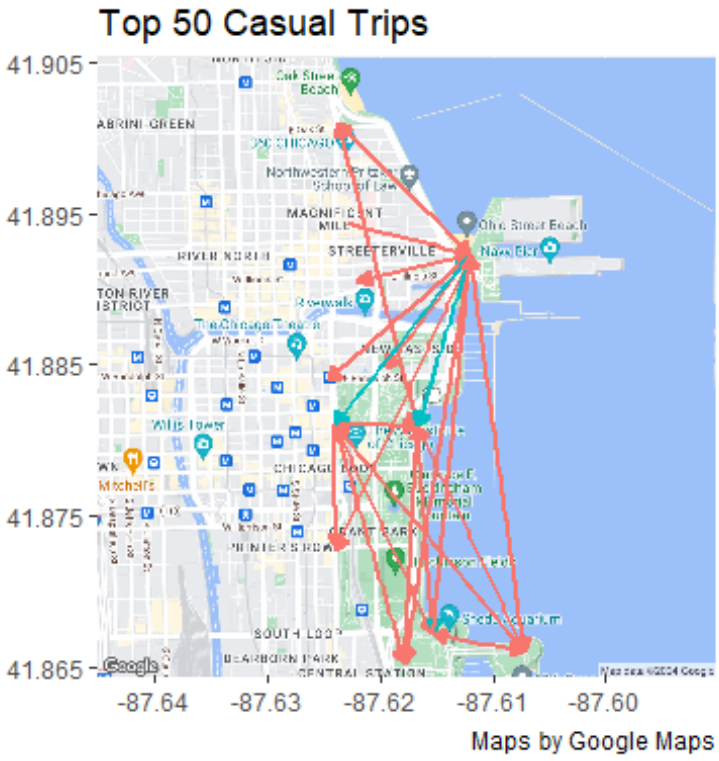
```
casualtripmap <- get_googlemap(center = c(lon = mean(Casual_trips$start_lng),
                                          lat = mean(Casual_trips$start_lat)),
                               width =26, height = 20, zoom = 14)
```

```
## i
<https://maps.googleapis.com/maps/api/staticmap?center=41.884924,-87.617723&z
oom=14&size=640x640&scale=2&maptype=terrain&key=xxx>
```

```
ggmap(casualtripmap) +
  geom_segment(
    data = Casual_trips,
    aes(x = start_lng, y = start_lat, xend = end_lng, yend = end_lat, color =
rideable_type),
    size = 0.8,
    arrow = arrow(length = unit(0.2, "cm"), ends = "first", type = "closed")
  ) +
  labs(title = "Top 50 Casual Trips", x = NULL, y = NULL, color = "User
Type", caption = "Maps by Google Maps") +
  theme(legend.position = "none")
```

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
```

```
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
generated.

## Warning: Removed 1 row containing missing values or values outside the
scale range
## (`geom_segment()`).

## Warning: Removed 3 rows containing missing values or values outside the
scale range
## (`geom_segment()`).
```

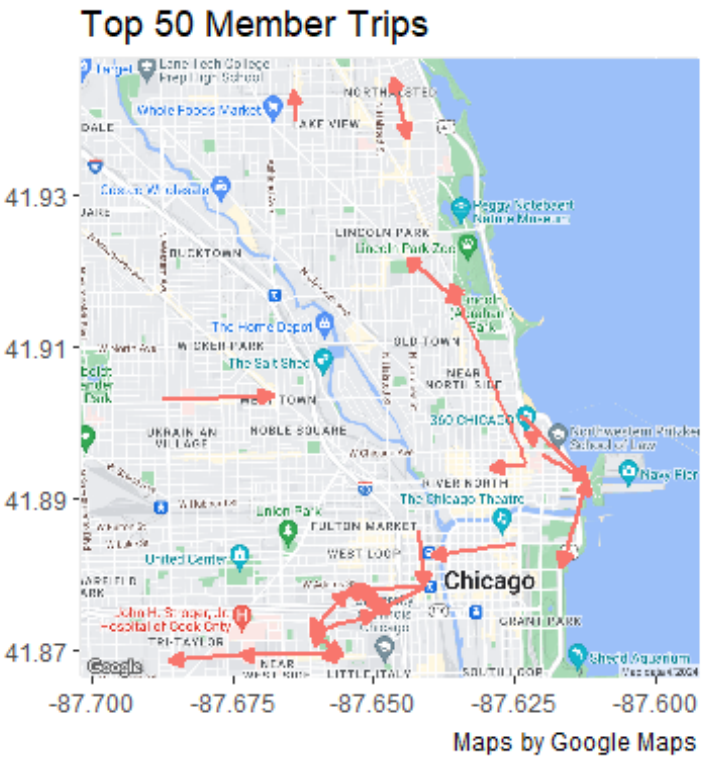


Now lets see if the top 50 Member Trip Routes correlate to the top 20 stations by count

Top 50 Member Trip Routes.

```
membertripmap <- get_googlemap(center = c(lon = mean(Member_trips$start_lng),
                                          lat = mean(Member_trips$start_lat)),
                               width =26, height = 20, zoom = 13)

## i
<https://maps.googleapis.com/maps/api/staticmap?center=41.907304,-87.647225&z
oom=13&size=640x640&scale=2&maptype=terrain&key=xxx>

ggmap(membertripmap) +
  geom_segment(
    data = Member_trips,
    aes(x = start_lng, y = start_lat, xend = end_lng, yend = end_lat, color =
```

```
rideable_type),
    size = 0.8,
    arrow = arrow(length = unit(0.2, "cm"), ends = "first", type = "closed")
  ) +
  labs(title = "Top 50 Member Trips", x = NULL, y = NULL, color = "User
Type", caption = "Maps by Google Maps") +
  theme(legend.position = "none")

## Warning: Removed 30 rows containing missing values or values outside the
scale range
## (`geom_segment()`).

## Warning: Removed 6 rows containing missing values or values outside the
scale range
## (`geom_segment()`).
```



Initial insights prove to be surprising that there are a significant amount more rides by Annual Members than there are Casual Riders. Also that the majority of rides start and end around the Navy Pier and the Aquarium, whereas Annual Members start and end points are a little more scattered. Stakeholders can offer discounted price promotions to sign up for membership to Casual riders that frequent these stations or ride these routes over a certain amount of times. This visualization is using routes taken over 250 times. I would suggest targeting riders after 5 matching route trips.

Also stakeholders could use information gathered by riders about why these routes are taken as marketing and promotions on the joys of Cyclistic's services.

Now lets create data frames that give us counts based on the days and we will call it `Daily_rides`

Note: Start by creating data frame for the daily starts and the daily ends.

```
Daily_starts <- Cyclistic_2023 %>%
  group_by(day_of_week_start, date, month, time_frame, member_casual,
rideable_type) %>%
  summarize(startcount = n(), .groups = "drop") %>%
  rename(day_of_week = day_of_week_start)
```

Note: Call the data frames `Daily_starts` and `Daily_end`

```
Daily_end <- Cyclistic_2023 %>%
  group_by(day_of_week_end, date, month, time_frame, member_casual,
rideable_type) %>%
  summarise(endcount = n(), .groups = "drop") %>%
  rename(day_of_week = day_of_week_end)
```

Note: Use bind_rows to combine the data frames while grouping the data according to the station name and counting the amount of rides into a new column called ridecount.

```
Daily_rides <- full_join(Daily_starts, Daily_end)

## Joining with `by = join_by(day_of_week, date, month, time_frame,
member_casual,
## rideable_type)`

Daily_rides <- drop_na(Daily_rides)
```

Note: Add a new column calculating the totalcount

```
Daily_rides <- Daily_rides %>%
  mutate(totalcount = startcount + endcount)
```

Now to get info specific about Riders

Note: Combine data into one data frame called `Rider_lookup.

```
Ridestart <- Cyclistic_2023 %>%
  distinct(ride_id, .keep_all = TRUE) %>%
  select(ride_id, member_casual, rideable_type,
         start_station_id, start_station_name,
         start_lng, start_lat, date, start_time,
         day_of_week_start, time_frame) %>%
  rename(weekday = day_of_week_start)
```

Note: start with a `ride_start` frame then a `ride_end` frame using the `distinct` function focusing on th `ride_id` column.

```
Ride_end <- Cyclistic_2023 %>%
  distinct(ride_id, .keep_all = TRUE) %>%
  select(ride_id, member_casual, rideable_type,
         end_station_id, end_station_name,
         end_lng, end_lat, end_date, end_time,
         day_of_week_end, time_frame) %>%
  rename(weekday = day_of_week_end, date = end_date)
```

Combine the data frames using `full_join`, call the new data frame Rider_lookup

```
Rider_lookup <- full_join(Ridestart, Ride_end) %>%
  distinct(ride_id, .keep_all = TRUE) %>%
  group_by(ride_id, date)

## Joining with `by = join_by(ride_id, member_casual, rideable_type, date,
weekday,
## time_frame)`
```

*Lastly, we will combine this data set and make a new data frame that will count the amount of rides corrosponding to the **days of the week** and **time frames** based on the `member_casual` column.*

```
Members_per_day <- Rider_lookup %>%
  group_by(member_casual, start_station_name, weekday, rideable_type) %>%
  summarize(member_per_day__count = n(), .groups = "drop") %>%
  rename(station_name = start_station_name)

Member_per_time_frame <- Rider_lookup %>%
  group_by(member_casual, start_station_name, time_frame, rideable_type) %>%
  summarize(time_frame_count = n(), .groups = "drop") %>%
  rename(station_name = start_station_name)
```

Note: For the sake of analyzation lets combine that data into one data frame.

```
member_per_day_counts <- full_join(Members_per_day, Member_per_time_frame,
                                   relationship = "many-to-many")

## Joining with `by = join_by(member_casual, station_name, rideable_type

)`
```

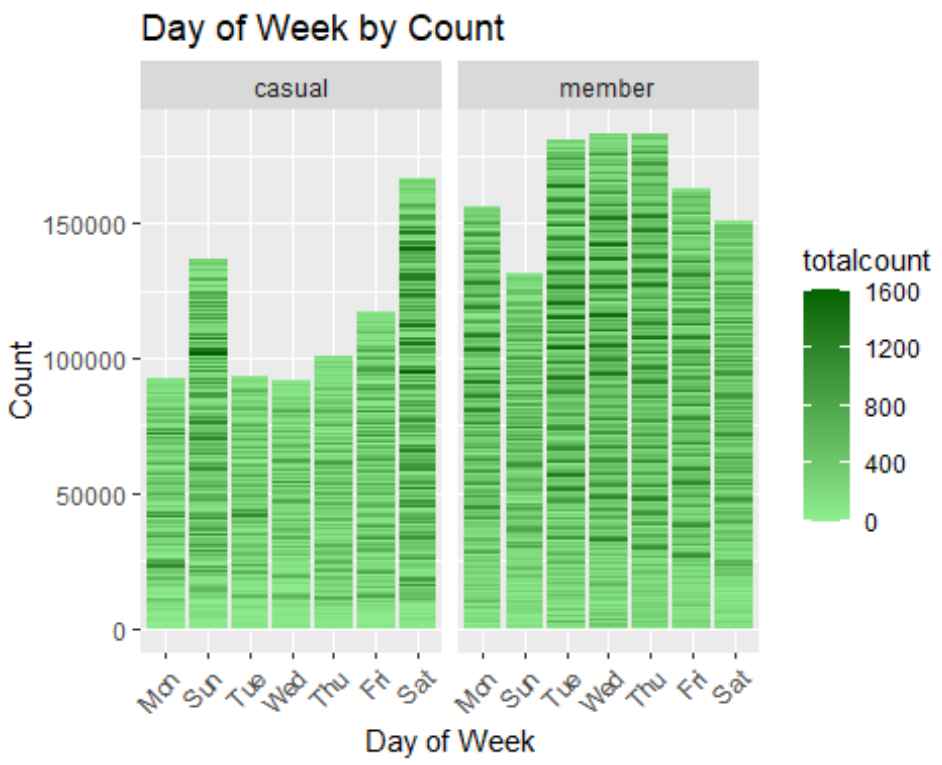
Note: Also let seperate and summarise a `ride_count` for the `rideable_type`, `date` and `weekday` based on our `member_casual` column.

```
rideable_counts <- Rider_lookup %>%
  group_by(rideable_type, member_casual) %>%
  summarise(ride_count = n(), .groups = "drop")

ride_counts_time <- Rider_lookup %>%
  group_by(date, member_casual) %>%
  summarise(ride_count = n(), .groups = "drop")

weekday_counts <- Rider_lookup %>%
  group_by(weekday, member_casual) %>%
  summarise(ride_count = n(), .groups = "drop") %>%
  filter(!is.na(weekday))
```

Note: This just gives us more points of data to analize but is not necessary

Lets visualize a **Bar chart** showing the differences in ride counts throughout the week.

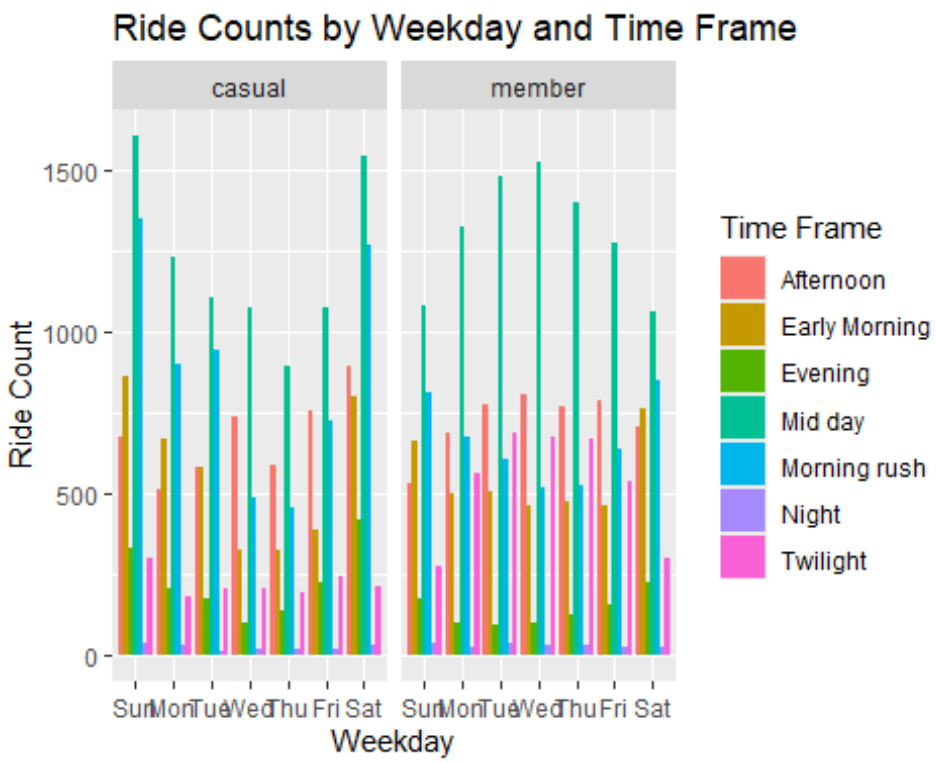
```
ggplot(Daily_rides, aes(x = reorder(day_of_week, totalcount), y = totalcount,
fill = totalcount)) +
  geom_bar(stat = "identity") +
  labs(x = "Day of Week", y = "Count") +
  ggtitle("Day of Week by Count") +
  scale_fill_gradientn(colors = c("lightgreen", "darkgreen"), limits = c(0,
max(Daily_rides$totalcount))) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  facet_wrap(~member_casual)
```



Note: This visualization seems to be giving us more underlying information lets visua;ize more of the data values in the data frame

We also have time frames in this data set so lets compare day of week and time frame.

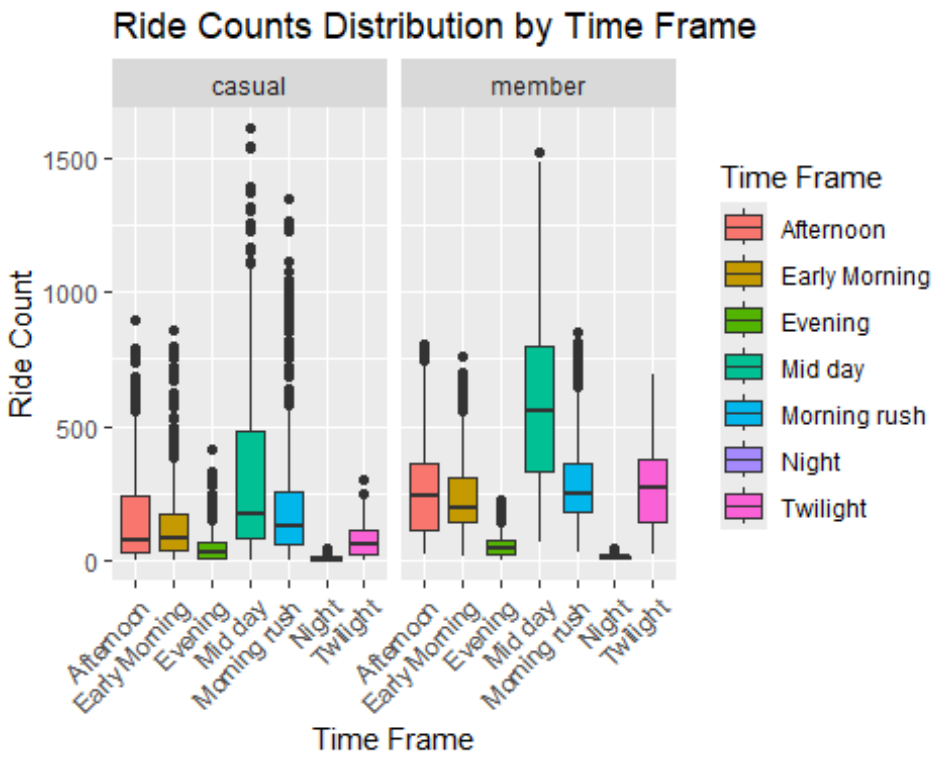
```
ggplot(Daily_rides, aes(x = day_of_week, y = totalcount, fill = time_frame))
+
  geom_bar(stat = "identity", position = "dodge") +
  labs(x = "Weekday", y = "Ride Count", fill = "Time Frame") +
  ggtitle("Ride Counts by Weekday and Time Frame") +
  facet_wrap(~member_casual)
```



Now for a clear picture of just the usage by time frame

We will use a Box Plot for this visualization.

```
ggplot(Daily_rides, aes(x = time_frame, y = totalcount, fill = time_frame)) +
  geom_boxplot() +
  labs(x = "Time Frame", y = "Ride Count", fill = "Time Frame") +
  ggtitle("Ride Counts Distribution by Time Frame") +
  facet_wrap(~member_casual) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



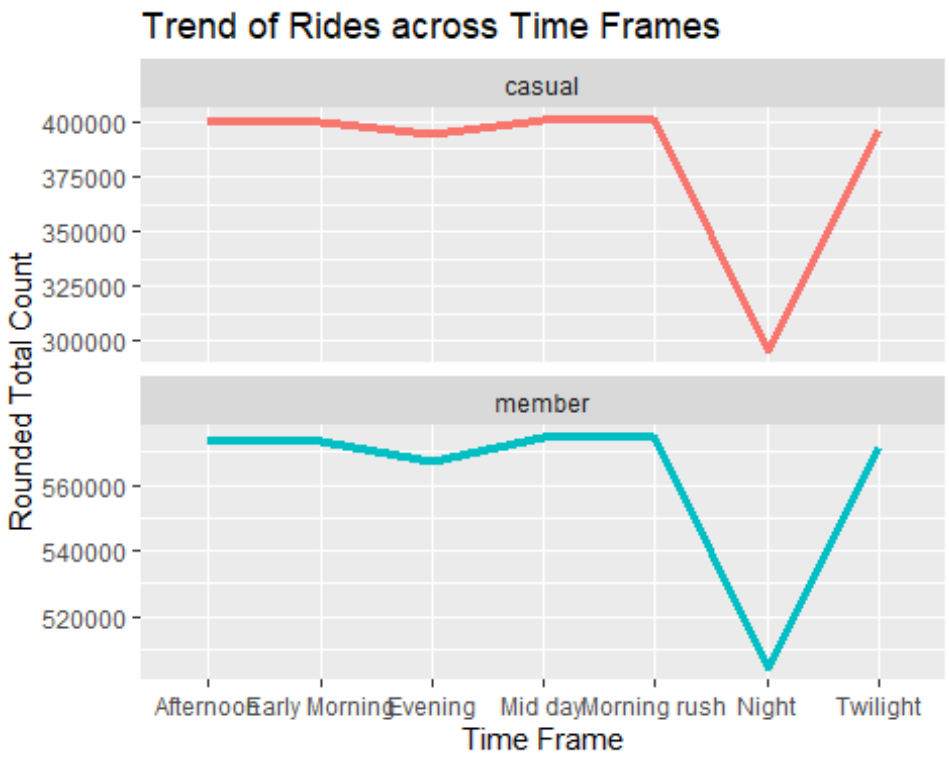
We can confirm our findings with visualizations using our other data frames

Note: `group_by` and `summarise` the rides by time frame based on the `member_casual` column.

```
line_chart_data <- member_per_day_counts %>%
  group_by(member_casual, time_frame) %>%
  summarise(total_count = sum(member_per_day__count), .groups = "drop")

line_chart_data_rounded <- line_chart_data %>%
  mutate(rounded_count = round(total_count, -2))
```

Note: Now plot the chart.

```
ggplot(line_chart_data_rounded, aes(x = time_frame, y = rounded_count, color
= member_casual, group = member_casual)) +
  geom_line(size = 1.5) +
  labs(x = "Time Frame", y = "Rounded Total Count", color = "User Type") +
  ggtitle("Trend of Rides across Time Frames") +
  facet_wrap(~ member_casual, scales = "free_y", nrow = 2) +
  theme(legend.position = "none")
```

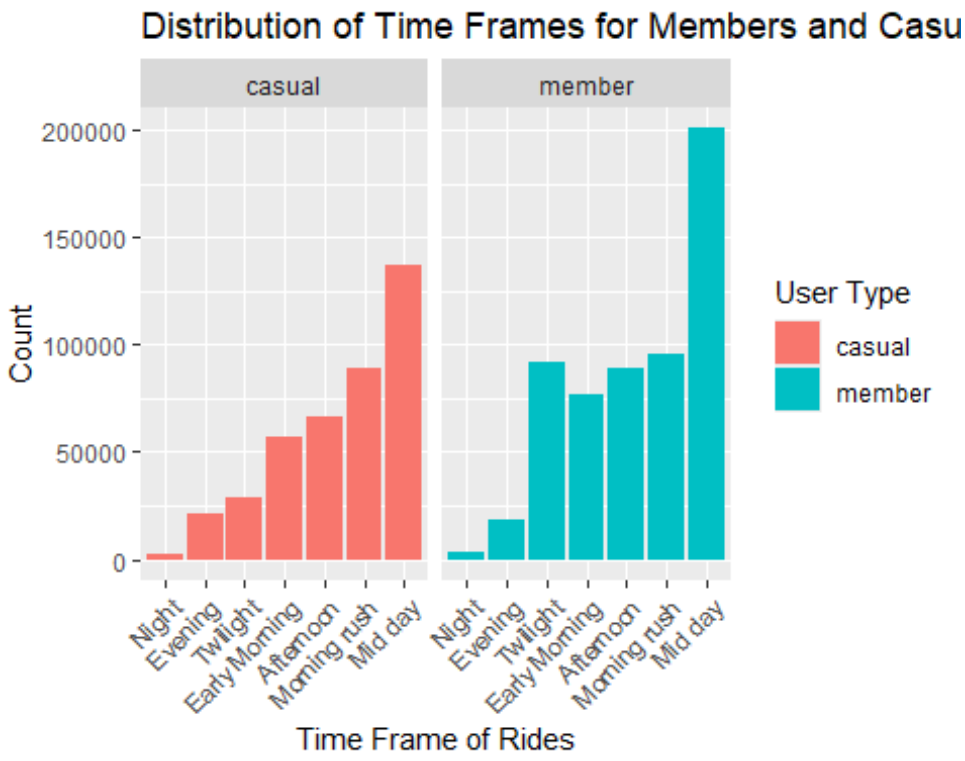


Note: Also in Bar Chart form.

```
Timeframe <- Cyclistic_2023 %>%
  group_by(member_casual, time_frame) %>%
  summarise(timeframe = n(), .groups = "drop")
```

Note: Now plot the chart.

```
ggplot(Timeframe, aes(x = reorder(time_frame, timeframe), y = timeframe, fill
= member_casual)) +
  geom_bar(stat = "identity", position = "dodge") +  # Use geom_bar for
categorical data
  labs(x = "Time Frame of Rides", y = "Count", fill = "User Type") +
  ggtitle("Distribution of Time Frames for Members and Casual Users") +
  facet_wrap(~ member_casual) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



As expected Annual Members are more likely to ride during mid week on the days of Tuesday, Wednesday and Thursday. While Casual Riders are more likely to ride during the weekend, Saturday and Sunday. Although what is surprising is that both Casual riders and Annual Members typically ride during the hrs of 10am-4pm or mid- day. Cyclistic stakeholders could target mid-day riders with marketing and membership promotions because their patterns already match those of Annual Members.

Also, Casual Riders that ride during the Evening rush could be targeted with more informative marketing explaining the benefits of the service during these hours and how Annual Members enjoy their subscriptions. Then send membership promotions once their patterns start to match those of the Annual Members.

Lets go back and get a General top 50 stations visted using our re framed data set

Note: First `group_by` and `summarise` the station names based on the `member_casual` column.

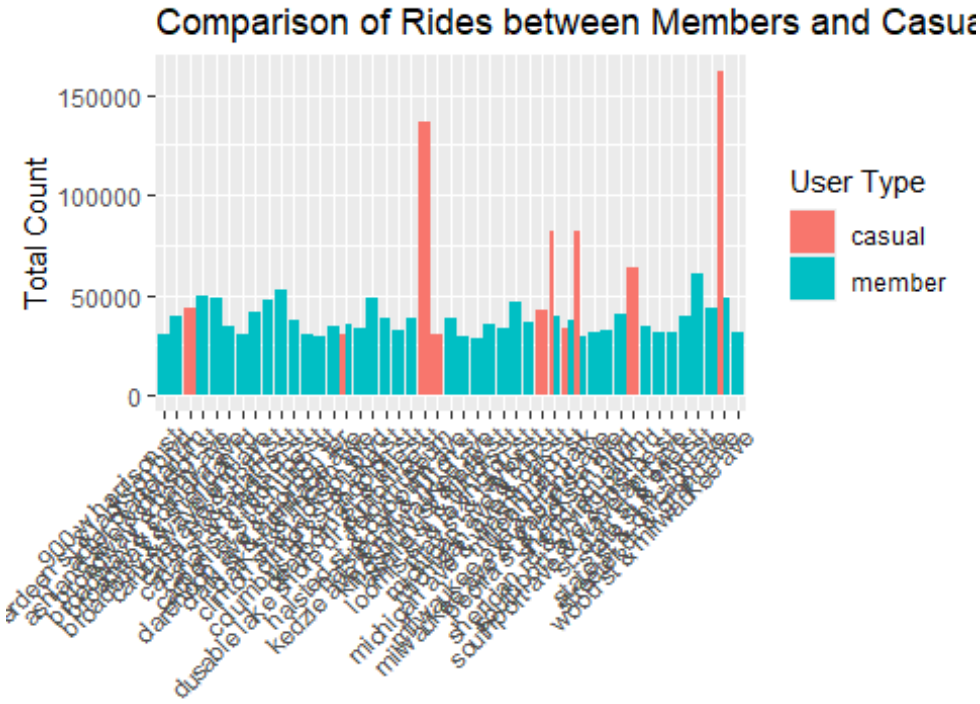
```
bar_plot_data <- member_per_day_counts %>%
  group_by(member_casual, station_name) %>%
  summarise(total_count = sum(member_per_day__count), .groups = "drop")
```

Note: Then filter out the top 50.

```
bar_plot_data <- bar_plot_data %>%
  arrange(desc(total_count)) %>%
  head(50)
```

Note: Finally plot te data.

```
ggplot(bar_plot_data, aes(x = station_name, y = total_count, fill =
member_casual)) +
  geom_bar(stat = "identity", position = "dodge") +
  labs(x = "", y = "Total Count", fill = "User Type") +
  ggtitle("Comparison of Rides between Members and Casual Users by Station")
+
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  scale_x_discrete(labels = c("Casual" = "Casual"))
```



Let's get a clearer view of the top stations visited by Casual Riders

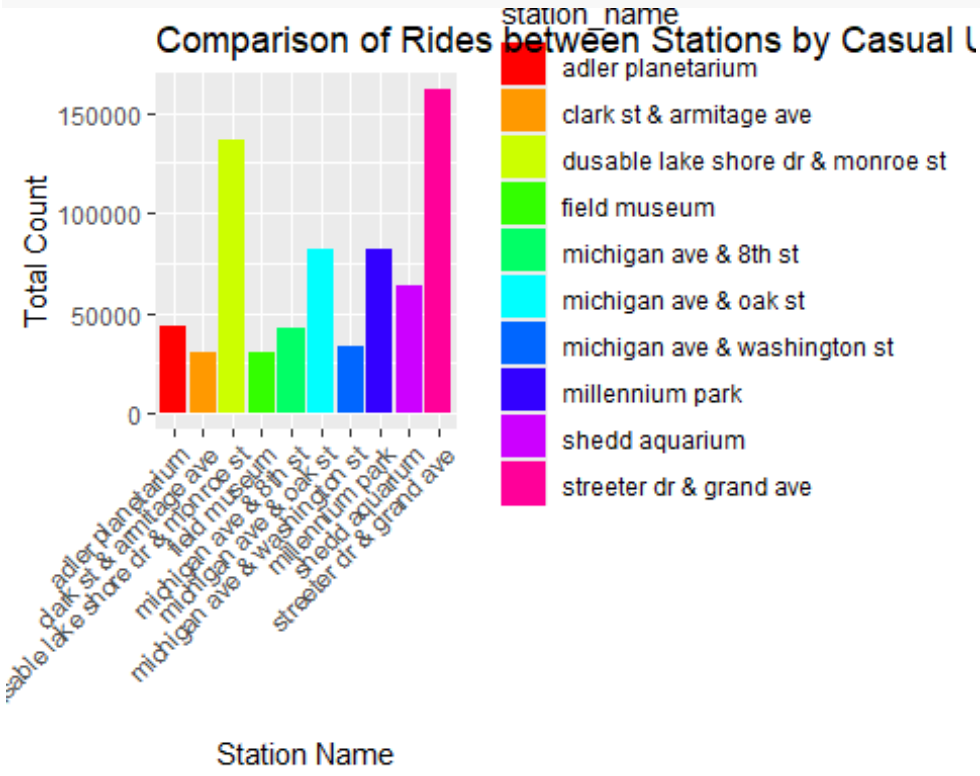
Note: STart by filtering out the casual rifers.

```
casual_stations <- bar_plot_data %>%
  filter(member_casual == "casual")
```

Note: Then plot the chart.

```
ggplot(casual_stations, aes(x = station_name, y = total_count, fill =
station_name)) +
  geom_bar(stat = "identity", position = "dodge") +
  scale_fill_manual(values =
```

```
rainbow(length(unique(casual_stations$station_name)))) +
  labs(x = "Station Name", y = "Total Count") +
  ggtitle("Comparison of Rides between Stations by Casual Users") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```

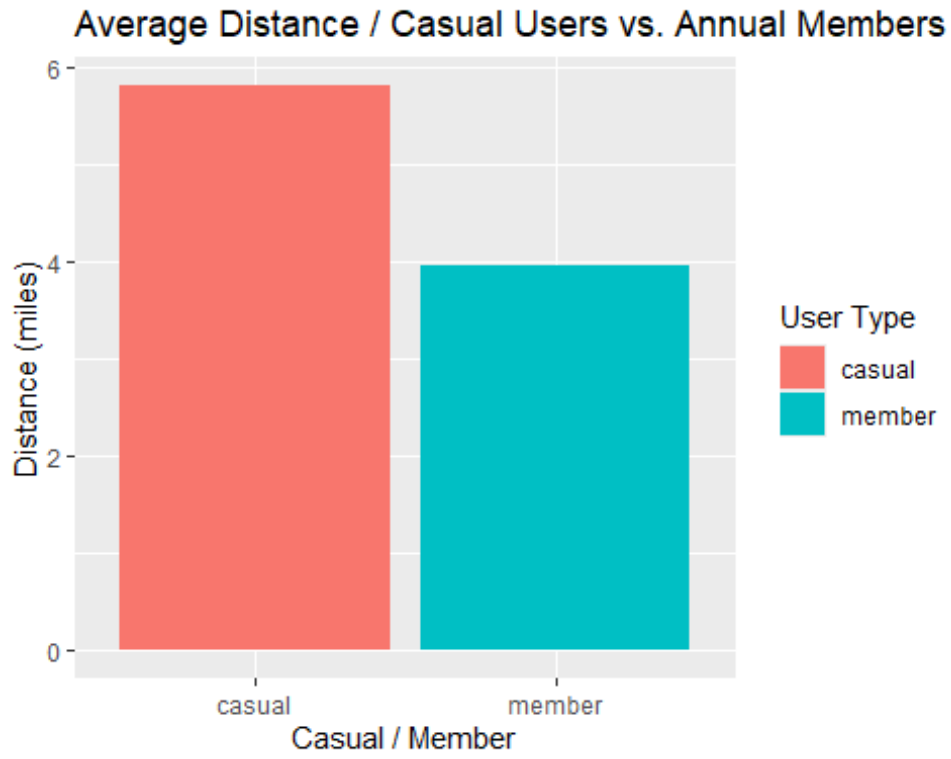


Our Top 20 Stations chart befit tatters the riders and their top routes taken. Using this to 10 stations list, stakeholders can produce a general membership campaign to Casual riders that visit these stations frequently as they prove to hold true throughout different data manipulations as the most used stations.

Lets visualize the Average Distance traveled between Casual Riders and Annual Members.

```
Discount <- Cyclistic_2023 %>%
  group_by(member_casual, rideable_type, distance_miles) %>%
  summarise(discount = n(), .groups = "drop")

Average_Distance <- Discount %>%
  group_by(member_casual, rideable_type,) %>%
  summarise(average_distance = mean(distance_miles, na.rm = TRUE), .groups =
"drop")

ggplot(Average_Distance, aes(x = reorder(member_casual, average_distance), y
= average_distance, fill = member_casual)) +
  geom_bar(stat = "identity") +
```

```
  labs(x = "Casual / Member", y = "Distance (miles)", fill = "User Type") +
  ggtitle("Average Distance / Casual Users vs. Annual Members")
```

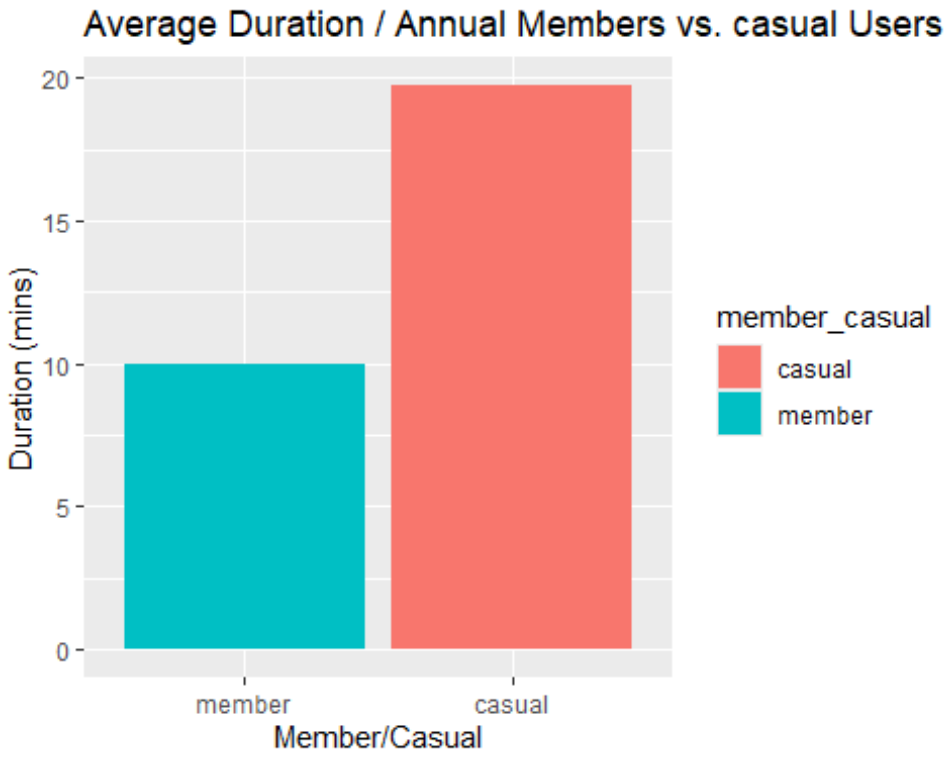


Note: Also in Line Graph view.

```
ggplot(Discount, aes(x = distance_miles, y = discount, color = member_casual,
group = member_casual)) +
  geom_line(size = 1.5) +
  labs(x = "Distance (miles)", y = "User Count", color = "User Type") +
  ggtitle("Trend of Rides Distance in Miles") +
  facet_wrap(~ member_casual, scales = "free_y", nrow = 2) +
  theme(legend.position = "none")
```

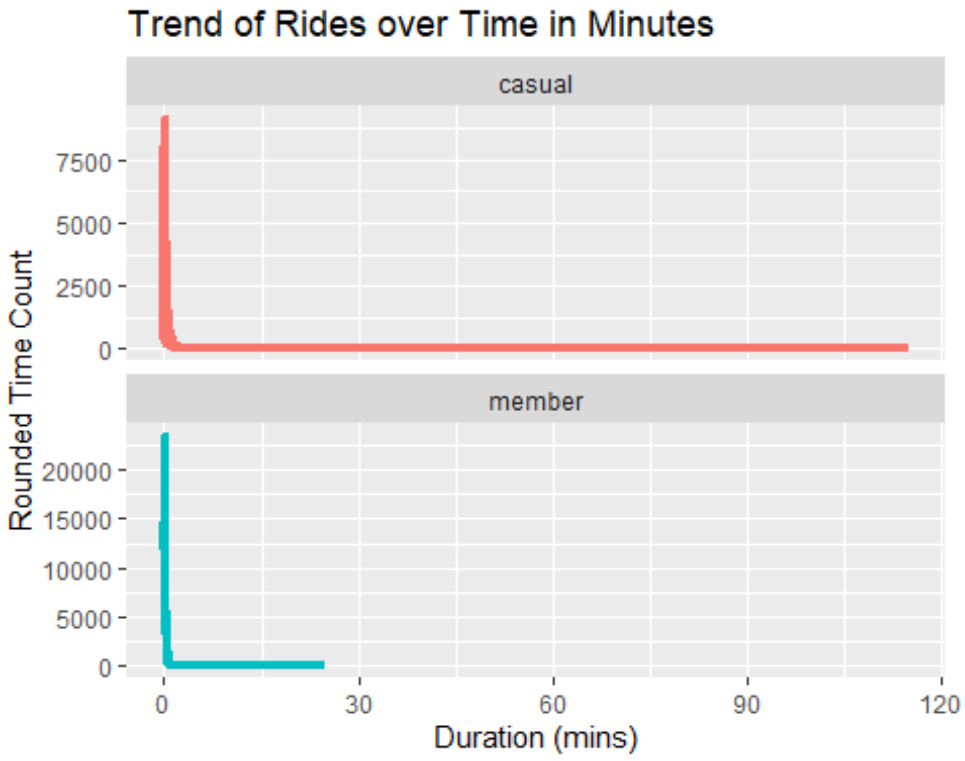
Lets visualize the Average Duration used between Casual Riders and Annual Members.

```
Timecount <- Cyclistic_2023 %>%
  filter(duration >= 0) %>%
  group_by(member_casual, rideable_type, duration) %>%
  summarise(timecount = n(), .groups = "drop")

Average_duration <- Timecount %>%
  group_by(member_casual, rideable_type) %>%
  summarise(average_duration = mean(duration, na.rm = TRUE), .groups =
"drop")

ggplot(Average_duration, aes(x = reorder(member_casual, average_duration), y
= average_duration, fill = member_casual)) +
  geom_bar(stat = "identity") +
  labs(x = "Member/Casual", y = "Duration (mins)") +
  ggtitle("Average Duration / Annual Members vs. casual Users")
```



Note: Also in Line Graph view.

```
ggplot(Timecount, aes(x = duration, y = timecount, color = member_casual,
group = member_casual)) +
  geom_line(size = 1.5) +
  labs(x = "Duration (mins)", y = "Rounded Time Count", color = "User Type")
+
  ggtitle("Trend of Rides over Time in Minutes") +
  facet_wrap(~ member_casual, scales = "free_y", nrow = 2) +
  theme(legend.position = "none")
```



It turns out that the Casual riders use the bike service on average longer per session. Also they ride farther per session, averaging out at just under 6 miles per session and just over 11 mins per session. While Annual Members averaging out at just under 4 miles and a little over 6 min per session.

If the stakeholders want to target riders using this information they should target Casual riders that eclipse 4 miles during their session multiple times detailing how they are already similar in usage pattern with Annual Members. Also, they should target Casual riders that eclipse the 10 min duration mark multiple times explaining how they use the service almost 2 times as much as Annual Members and the benefits of subscribing.

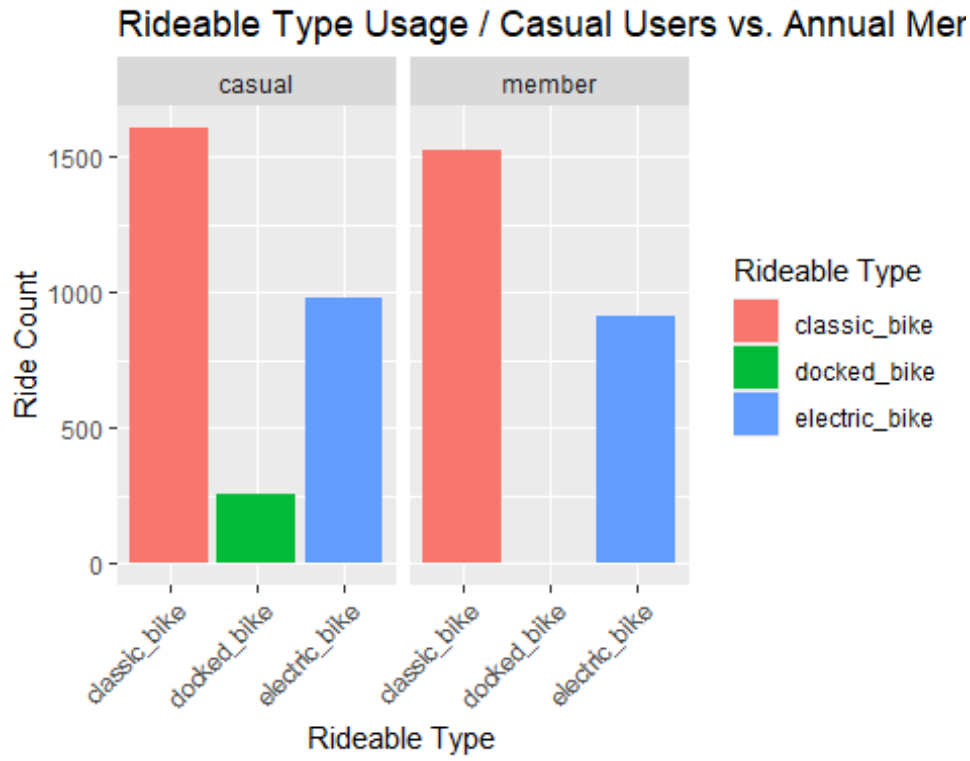
Now we can take a look at the differences in the *rideable types* between users to see if they can give us any insights

Note: Lets assume docked bikes are bikes that are being maintained or user error trying to retrive/disgard the bike

Note: we do not know so lets not remove them.

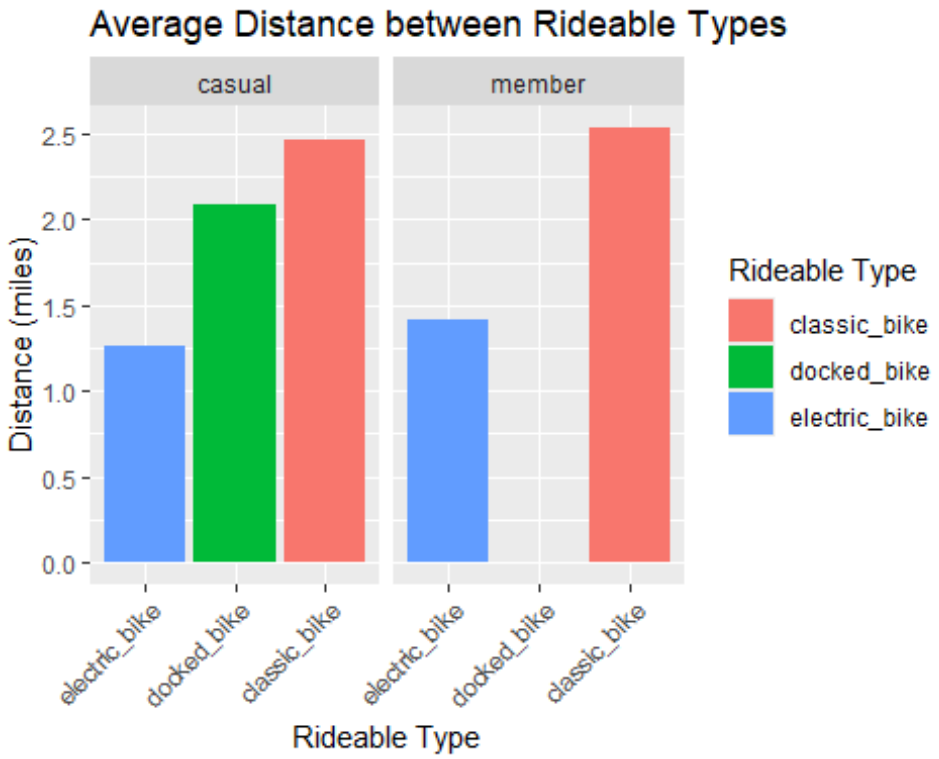
```
ggplot(Daily_rides, aes(x = rideable_type, y = totalcount, fill =
rideable_type)) +
  geom_bar(stat = "identity", position = "dodge") +
  labs(x = "Rideable Type", y = "Ride Count", fill = "Rideable Type") +
```

```
  ggtitle("Rideable Type Usage / Casual Users vs. Annual Members") +
  facet_wrap(~member_casual)+
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



We can see the Average Distance traveled between rideable types.

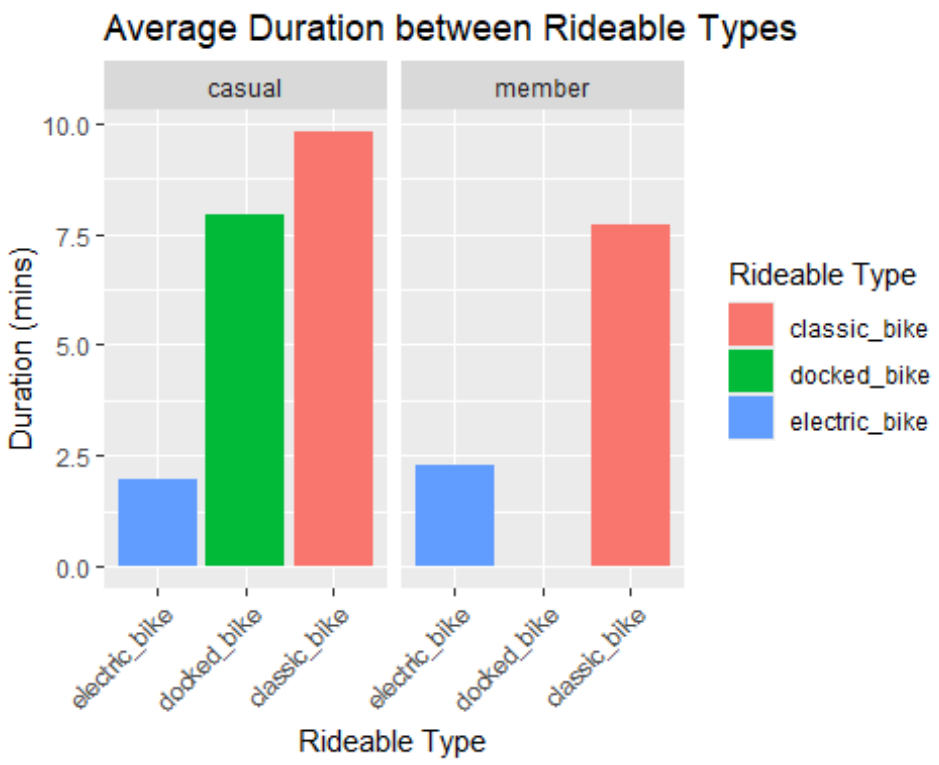
```
ggplot(Average_Distance, aes(x = reorder(rideable_type, average_distance), y
= average_distance, fill = rideable_type)) +
  geom_bar(stat = "identity") +
  labs(x = "Rideable Type", y = "Distance (miles)", fill = "Rideable Type") +
  ggtitle("Average Distance between Rideable Types") +
  facet_wrap(~member_casual) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



Note: `docked_bike` Lets assume this is error due to travel to maintainance shop

Aslo the Average Time the Classic, Electric or Docked bikes were in use.

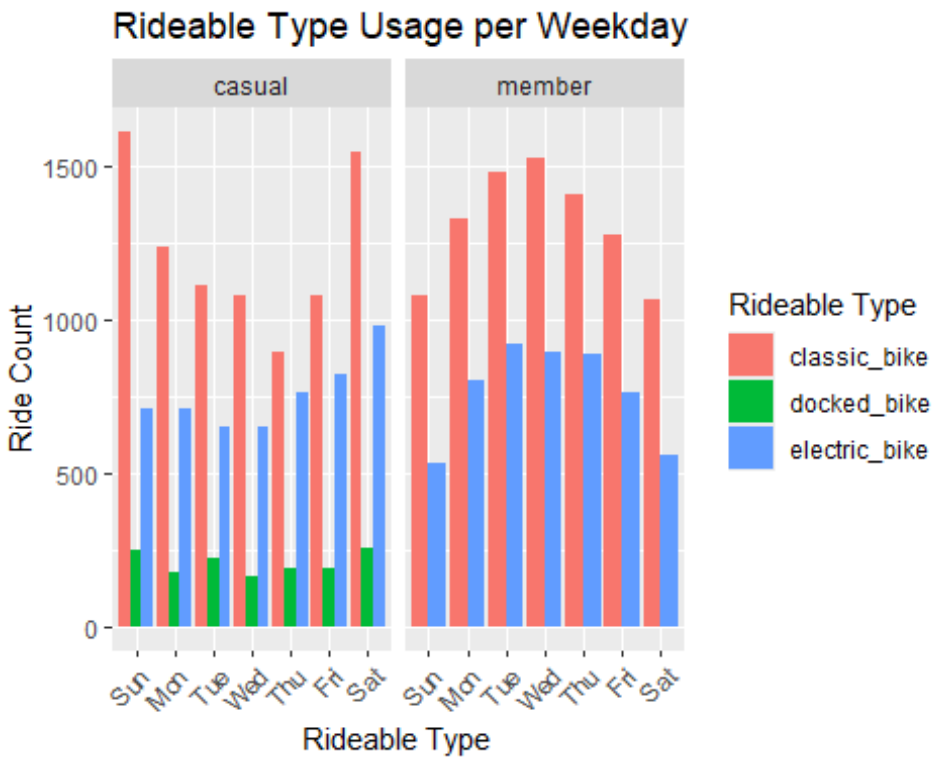
```
ggplot(Average_duration, aes(x = reorder(rideable_type, average_duration), y
= average_duration, fill = rideable_type)) +
  geom_bar(stat = "identity") +
  labs(x = "Rideable Type", y = "Duration (mins)", fill = "Rideable Type") +
  ggtitle("Average Duration between Rideable Types") +
  facet_wrap(~member_casual) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



Note: `docked_bike` Lets assume this is error in retrival or disregard of bikes

We can also view the Usage of `rideable_type` in relation to each day of the week to see if it holds true with our general weekly usage rates.

```
ggplot(Daily_rides, aes(x = day_of_week, y = totalcount, fill =
rideable_type)) +
  geom_bar(stat = "identity", position = "dodge") +
  labs(x = "Rideable Type", y = "Ride Count", fill = "Rideable Type") +
  ggtitle("Rideable Type Usage per Weekday") +
  facet_wrap(~member_casual)+
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



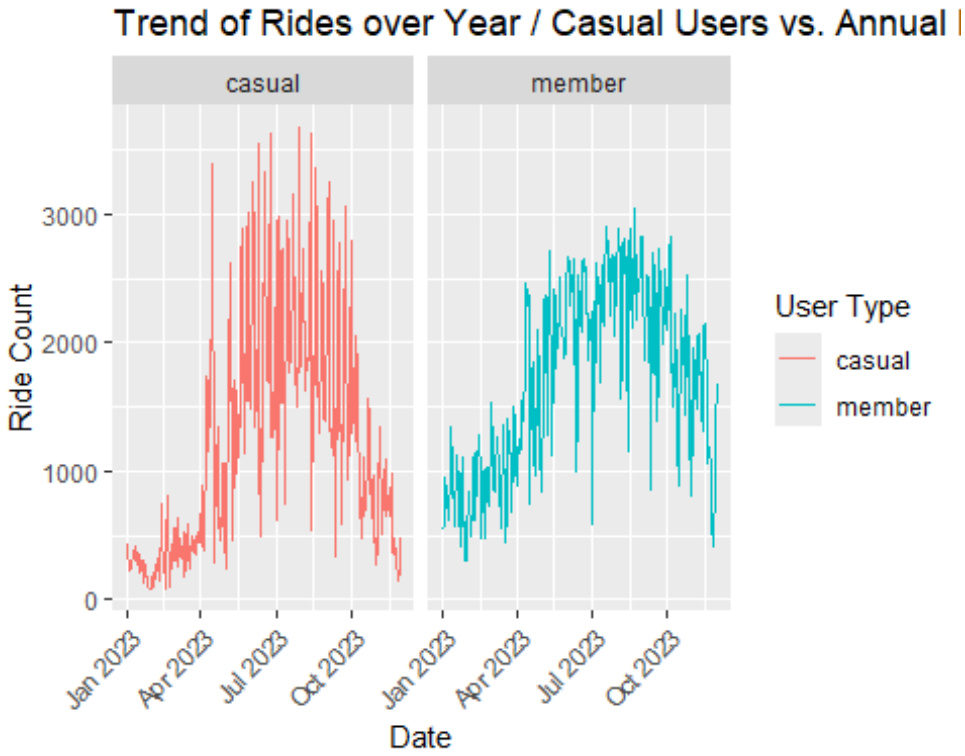
These visualizations cite that while classic bikes are widely preferred by both Casual and Member riders. The electric bikes are slightly more preferred by Annual Members over Casual Riders. On average Casual riders ride the electric bikes just over 1.25 miles while Annual Members ride for just under 1.5 miles per session on average. Classic bike usage is just under 2.5 miles and just over 2.5 miles for Casual and Member riders respectively.

One other observation I noticed is that although the Annual Members outpace Casual riders on average distance traveled. The Casual riders have higher classic bike duration times. More than a full minute per session, averaging over 5 min per session while the Members use classic bikes just under 5 min per session on average. Using this information stakeholders could immediately implement a promotional campaign designed to target Casual Classic bike riders that eclipse the 4 min duration mark multiple times. Explaining that their ride patterns outpace those of Annual Members.

Finally lets see the trend of rides across the entire year between Casual Members and Annual Members.

```
ggplot(ride_counts_time, aes(x = date, y = ride_count, color =
member_casual)) +
```

```
  geom_line() +
  labs(x = "Date", y = "Ride Count", color = "User Type") +
  ggtitle("Trend of Rides over Year / Casual Users vs. Annual Members") +
facet_wrap(~member_casual) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



In this final visualization we can see the pattern of usage between Casual Riders and Annual Members throughout the entire year that we have so far. We see that the most active times are from June-September or the summer season. Stakeholders should do a general promotion for discounted membership pricing to sign up during the summer session as this proves to be the busiest season.

Our team and stakeholders would benefit from setting up a quick survey to both Casual and Member riders to help better understand why we see these patterns in the riders and what improvements can be made.

In conclusion, our team should first set up a series of surveys to find out why Annual Members / Casual Riders take the routes that they do? Why do they choose specific days or time frames to use the service? What influences rideable type decisions? Also, whether or not these factors affect distance traveled or duration of usage? This will help in marketing the pain points of Casual Riders that match the trends and patterns of Annual Riders.

Casual Riders that match ride patterns of Annual Members should receive a two step discounted sign-up promotion. and extended membership promotions. First an informative blast detailing the ride patterns Cyclistic is trying to target, then a call to action because this segment of riders are most likely to make the switch.

Casual Riders that oppose the ride patterns of Annual Members should receive a series of marketing blasts detailing why Annual Members ride in the patterns that they do and the benefits that Annual Membership gives them to accomplish their goals. Once their ride patterns start to match those of Members. Then, send them discounted sign up promotions and extended membership promotions.