Introduction

This is analysis of Google Data Analytics Capstone: Complete a Case Study course. In this case study, I will perform many real-world tasks of a junior data analyst for marketing Team at Cyclistic, a bike-share company in Chicago.

Background

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

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

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

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

Scenario

You are a junior data analyst working in the marketing analyst team at Cyclistic, a bike-share company in Chicago. The director of marketing believes the company's future success depends on maximizing the number of annual memberships. Therefore, your team wants to understand how casual riders and annual members use Cyclistic bikes differently. From these insights, your team will design a new marketing strategy to convert casual riders into annual members. But first, Cyclistic executives must approve your recommendations, so they must be backed up with compelling data insights and professional data visualizations.

Question to Analyze

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

Prepare

Data Source

Dataset from 01-01-2021 to 31-12-2021 available from Cyclistic Bike Sharing Data. The data has been made available by Motivate International Inc. under this license. Each csv files contain 13 columns containing information related to ride id, ridership type, ride time and location and location etc. File naming: YYYYMM-divvy-tripdata.csv and stored by monthly data file (total 12 files). Next, these datasets will be organized by PostgreSQL.

Process

First, we need to create a database for our dataset. We can use right-clicking of databases icon in menu bar and create database. In this case, the name of database is ‘Cyclist’. Next, we can activate Query Tools and create the tables for our datasets by querying it. The name of headers is based on the name of csv's file header and the name of this table is 'divvy_tripdata_2021'. Finally, we can import our csv files by right-clicking 'divvy_tripdata_2021' table under schema of ‘Cyclist’ database.

After that, we can check the structure of our previous table using PSQL tool by right clicking 'Cyclist' Database. Then we can input the command '\d divvy_tripdata_2021' without quotation mark.

```
Cyclist=# \d divvy_tripdata_2021
                    Table "public.divvy_tripdata_2021"
      Column       |            Type             | Collation | Nullable | Default
-------------------+-----------------------------+-----------+----------+---------
 ride_id           | character varying(50)       |           |          |
 rideable_type     | character varying(50)       |           |          |
 started_at        | timestamp without time zone |           |          |
 ended_at          | timestamp without time zone |           |          |
 start_station_name | character varying(100)     |           |          |
 start_station_id  | character varying(100)      |           |          |
 end_station_name  | character varying(100)      |           |          |
 end_station_id    | character varying(100)      |           |          |
 start_lat         | numeric                     |           |          |
 start_lng         | numeric                     |           |          |
 end_lat           | numeric                     |           |          |
 end_lng           | numeric                     |           |          |
 member_casual     | character varying(100)      |           |          |
```

We also can check the preview of first n-values in our tables in Query Tool. In this case, I want to check first 10-values in 'divvy_tripdata_2021'.

ride_id	rideable_t	started_at	ended_at	start_station_name	start_station_id	end_station_name	end_station_id	start_lat	start_lng	end_lat	end_lng	member_casual
E19E6F1B8D4	electric_bi	23/01/2021 16:14	23/01/2021 16:24	California Ave & Cortez St	17660			41.900.340.666.666.600	-87.696.743	41.89	-87.72	member
DC88F20C2C5	electric_bi	27/01/2021 18:43	27/01/2021 18:47	California Ave & Cortez St	17660			4.190.033.283.333.330	-87.696.707	41.9	-87.69	member
EC45C94683F	electric_bi	21/01/2021 22:35	21/01/2021 22:37	California Ave & Cortez St	17660			41.900.312.666.666.600	-8.769.664.266.666.660	41.9	-87.7	member
4FA453A75A8	electric_bi	07/01/2021 13:31	07/01/2021 13:42	California Ave & Cortez St	17660			4.190.039.866.666.660	-8.769.666.216.666.660	41.92	-87.69	member
BE5E8EB4E72	electric_bi	23/01/2021 02:24	23/01/2021 02:24	California Ave & Cortez St	17660			4.190.032.633.333.330	-8.769.669.716.666.660	41.9	-87.7	casual
5D8969F88C7	electric_bi	09/01/2021 14:24	09/01/2021 15:17	California Ave & Cortez St	17660			41.900.408.666.666.600	-8.769.676.283.333.330	41.94	-87.71	casual
09275CC10F8	electric_bi	04/01/2021 05:05	04/01/2021 05:10	California Ave & Cortez St	17660			419.003.905	-8.769.675.733.333.330	41.9	-87.71	member
DF7A32A217A	electric_bi	14/01/2021 15:07	14/01/2021 15:13	California Ave & Cortez St	17660			419.003.895	-8.769.672.833.333.330	41.91	-87.7	member
C2EFC62379E	electric_bi	09/01/2021 09:57	09/01/2021 10:00	California Ave & Cortez St	17660			41.900.305.833.333.300	-8.769.679.633.333.330	41.9	-87.7	member
B9F73448DFB	classic_bik	24/01/2021 19:15	24/01/2021 19:22	California Ave & Cortez St	17660	Wood St & Augusta Blvd	657	41.900.363	-87.696.704	41.899.181	-876.722	member

Then we can also check the total number of All rows from this table.

	count bigint
1	5595063

Data Cleaning

Next step is cleaning the data. First, I want to investigate any possibility of duplicate data. In this case, I want to investigate the possibility of EXACT duplicate data in each column using this query.

```
SELECT
    ride_id, rideable_type, started_at, ended_at,
    start_station_name, start_station_id, end_station_name, end_station_id, start_lat,
    start_lng, end_lat, end_lng, member_casual, COUNT(*)
FROM public.divvy_tripdata_2021
GROUP BY
    ride_id, rideable_type, started_at, ended_at,
    start_station_name, start_station_id, end_station_name, end_station_id, start_lat,
    start_lng, end_lat, end_lng, member_casual
HAVING COUNT(*) > 1;
```

The result shows there is no duplicate data which means, this table has the unique values in each row.

Explain Data Output Messages Notifications

```
Successfully run. Total query runtime: 1 min 35 secs.
0 rows affected.
```

We may also check the duplicate data based on any certain columns to investigate it deeper. In this next case, I want to check any possibility of duplicate values even the ride_id is unique. This is likely to happen because the data is inputted more than once by accident.

```
SELECT
    ride_id, rideable_type, started_at, ended_at,
    start_station_name, start_station_id, end_station_name, end_station_id, COUNT(*)
FROM public.divvy_tripdata_2021
GROUP BY
    ride_id, rideable_type, started_at, ended_at,
    start_station_name, start_station_id, end_station_name, end_station_id
HAVING COUNT(*) > 1;
```

The last results also show its consistency if there is no duplicate data in this table.

Next, we must check the integrity of dataset. It means the data must contains right and logic value. To know it, I use duration of trips for each ride_id. In this part, I use two parameters using difference of time in the column 'diff_time' and difference times in second in the column "diff_in_second". For notes, PostgreSQL don't have the function of 'datediff' like other SQL application to extract time difference from timestamps data. In this case, I use 'date_part' function from PostgreSQL.

```
SELECT
    started_at,
    ended_at,
    (ended_at - started_at) AS diff_time,
    ((DATE_PART('day', ended_at::timestamp - started_at::timestamp) * 24 +
              DATE_PART('hour', ended_at::timestamp - started_at::timestamp)) * 60 +
              DATE_PART('minute', ended_at::timestamp - started_at::timestamp)) * 60 +
              DATE_PART('second', ended_at::timestamp - started_at::timestamp) AS diff_in_second
FROM public.divvy_tripdata_2021
ORDER BY diff_time DESC
```

	started_at timestamp without time zone	ended_at timestamp without time zone	diff_time interval	diff_in_second double precision
1	2021-06-05 02:27:26	2021-07-13 22:51:35	38 days 20:24:09	3356649
2	2021-06-04 22:03:33	2021-07-13 14:15:14	38 days 16:11:41	3341501
3	2021-05-02 02:56:07	2021-06-08 13:37:43	37 days 10:41:36	3235296
4	2021-06-05 23:33:51	2021-07-12 13:55:14	36 days 14:21:23	3162083
5	2021-07-08 19:29:49	2021-08-11 21:56:58	34 days 02:27:09	2946429
6	2021-04-02 17:50:00	2021-05-05 22:06:42	33 days 04:16:42	2866602
7	2021-06-05 21:47:40	2021-07-08 13:18:31	32 days 15:30:51	2820651
8	2021-07-08 15:13:08	2021-08-06 13:18:39	28 days 22:05:31	2498731
9	2021-08-01 18:53:10	2021-08-30 16:42:20	28 days 21:49:10	2497750

Everything seems to be correct, but we must ensure it deeper, if there are not duration of trips in in negative values. I use these two queries to double check deeper the integrity of dataset.

```
SELECT
    started_at,
    ended_at,
    (ended_at - started_at) AS diff_time,
    ((DATE_PART('day', ended_at::timestamp - started_at::timestamp) * 24 +
              DATE_PART('hour', ended_at::timestamp - started_at::timestamp)) * 60 +
              DATE_PART('minute', ended_at::timestamp - started_at::timestamp)) * 60 +
              DATE_PART('second', ended_at::timestamp - started_at::timestamp) AS diff_in_second
FROM public.divvy_tripdata_2021
WHERE  ((DATE_PART('day', ended_at::timestamp - started_at::timestamp) * 24 +
              DATE_PART('hour', ended_at::timestamp - started_at::timestamp)) * 60 +
              DATE_PART('minute', ended_at::timestamp - started_at::timestamp)) * 60 +
              DATE_PART('second', ended_at::timestamp - started_at::timestamp) < 0;
```

```
SELECT COUNT(*)
FROM public.divvy_tripdata_2021
WHERE  ((DATE_PART('day', ended_at::timestamp - started_at::timestamp) * 24 +
              DATE_PART('hour', ended_at::timestamp - started_at::timestamp)) * 60 +
              DATE_PART('minute', ended_at::timestamp - started_at::timestamp)) * 60 +
              DATE_PART('second', ended_at::timestamp - started_at::timestamp) < 0;
```

Voila, we find it. There are 147 rows with negative values. These rows are wrong data, and we need to remove these rows from dataset.

After removing the wrong data, we need to deal with Nulls values. A NULL value indicate that a data value does not exist in the database. In other words, it is just a placeholder to denote values that are missing or that we do not know. NULL can be confusing and cumbersome at first.

From the previous data preview, we can see if Nulls seems to be appeared some columns. First, I want to investigate the number of Null per column.

null_ride_id	null_rideal	null_started_at	null_ended_at	null_start_station_name	null_start_statio	null_end_station_name	null_end_statio	null_start_lat	null_start_lng	null_end_lat	null_end_lng	null_member_casual	count
0	0	00/01/1900 00:00	00/01/1900 00:00	690789	690786	739149	739149	0	0	4770	4770	0	5594916

From this result, we know most of missing values mostly happen in stations columns, both in 'Start_station_name' and 'end_station_name'. The result shows, there are maximum 739.149 rows with at least 1 missing value in their columns. It means this table has 13,21% Null values (from 5.594.916 rows).

Next, I want to investigate where does the Null mostly occur in dataset. The easiest parameter is using 'rideable_type' column because it contains 3 variables for comparison.

	rideable_type character varying (50)	null_start_station_name bigint	null_end_station_name bigint	count bigint
1	classic_bike	0	9039	3250943
2	docked_bike	0	294	312338
3	electric_bike	690789	729816	2031635

The query results show that the majority of NULLs appear in 'electric_bike' data. There are 729.816 nulls or 98,07% of the total Nulls that appear ONLY in 'electric_bike' data. This number is also equivalent to 35.92% of total 'electric_bike' data. So, instead of deleting all data with Null, which means it will delete a lot of data and cause a possible inaccurate analysis. I will treat Null data differently.

--- CASE 1: If total dif_in_second is Not 0 AND start_lat or start_lng is Not Null AND end_lat or end_lng is NOT Null THEN there is a transaction on this ride_id even though the column of 'start_station_name' or 'end_station_name' contains Null values.

--- CASE 2: If total dif_in_second is 0 AND start_lat or start_lng is NULL OR Not Null AND end_lat or end_lng is Null THEN there is no transaction on this ride_id. WE MUST REMOVE THIS ROWS.

Case 1 Sample:

	diff_in_second double precision	start_station_name character varying (100)	end_station_name character varying (100)	start_lat numeric	start_lng numeric	end_lat numeric	end_lng numeric
1	382	[null]	[null]	41.92	-87.71	41.91	-87.71
2	281	[null]	[null]	41.93	-87.72	41.92	-87.72
3	636	[null]	[null]	41.91	-87.71	41.93	-87.72
4	223	[null]	[null]	41.93	-87.71	41.93	-87.71
5	269	[null]	[null]	41.93	-87.71	41.93	-87.71
6	437	[null]	[null]	41.74	-87.59	41.73	-87.57
7	319	[null]	[null]	41.92	-87.66	41.92	-87.66
8	439	[null]	[null]	41.68	-87.54	41.69	-87.52
9	547	[null]	[null]	41.78	-87.63	41.78	-87.6
10	290	[null]	[null]	41.8	-87.59	41.79	-87.6

Case 2 Sample:

diff_in_second	started_at	ended_at	start_station_name	end_station_name	start_lat	start_lng	end_lat	end_lng
0	15/01/2021 16:40	15/01/2021 16:40	Damen Ave & Thomas St (Augusta Blvd)		41.901.315	-87.677.409	41.9	-87.68
0	29/01/2021 21:02	29/01/2021 21:02	Loomis St & Lexington St	Loomis St & Lexington St	41.872.187	-87.661.501	41.872.187	-87.661.501
0	14/01/2021 17:30	14/01/2021 17:30	Desplaines St & Kinzie St		41.888.612.333.333.300	-8.764.443.783.333.330	41.89	-87.64
0	14/01/2021 17:26	14/01/2021 17:26	Broadway & Ridge Ave		41.984.089.166.666.600	-8.766.021.233.333.330	41.98	-87.66
0	24/02/2021 21:14	24/02/2021 21:14	State St & Pearson St		41.897.620.333.333.300	-8.762.876.566.666.660	41.9	-87.63
0	08/02/2021 11:18	08/02/2021 11:18	Wells St & Huron St	Wells St & Huron St	41.894.722	-87.634.362	41.894.722	-87.634.362
0	13/02/2021 17:32	13/02/2021 17:32	Rush St & Cedar St	Rush St & Cedar St	4.190.230.870.122	-87.627.690.528	4.190.230.870.122	-87.627.690.528
0	22/02/2021 17:24	22/02/2021 17:24	Clark St & Lake St		4.188.602.082.773	-876.308.760.584	41.89	-87.63
0	24/03/2021 19:32	24/03/2021 19:32	Clark St & Wellington Ave	Clark St & Wellington Ave	419.364.968.219	-876.475.386.582	419.364.968.219	-876.475.386.582
0	05/03/2021 08:10	05/03/2021 08:10	Lake Shore Dr & Belmont Ave	Lake Shore Dr & Belmont Ave	41.940.775	-87.639.192	41.940.775	-87.639.192

Total Count of Case 2

	rideable_type character varying (50)	count bigint
1	classic_bike	197
2	docked_bike	3
3	electric_bike	306

The result shows, there are 506 deleted rows with Case 2 Criteria

After dealing with Nulls, next we need to deal with Outlier data. Outlier is a data point in the dataset that differs significantly from the other data or observations. Many statistic procedures are affected by the presence of outliers. So, in this case, removing the outlier may be an option. We can use standard deviation to remove outliers and get a "trimmed average". In PostgreSQL, the function is called stddev_samp().

```
SELECT
        member_casual,
        count(*),
        MAX(DATE_PART('day', ended_at::timestamp - started_at::timestamp) * 24 +
        DATE_PART('hour', ended_at::timestamp - started_at::timestamp)) as max_hour_outlier,
        MIN(DATE_PART('day', ended_at::timestamp - started_at::timestamp) * 24 +
        DATE_PART('hour', ended_at::timestamp - started_at::timestamp)) as min_hour_oitlier
FROM public.divvy_tripdata_2021
WHERE
        DATE_PART('day', ended_at::timestamp - started_at::timestamp) * 24 +
        DATE_PART('hour', ended_at::timestamp - started_at::timestamp) NOT IN
        (
    SELECT DATE_PART('day', ended_at::timestamp - started_at::timestamp) * 24 +
            DATE_PART('hour', ended_at::timestamp - started_at::timestamp) as diff_in_hour
    FROM public.divvy_tripdata_2021
    WHERE
        DATE_PART('day', ended_at::timestamp - started_at::timestamp) * 24 +
        DATE_PART('hour', ended_at::timestamp - started_at::timestamp) >
            (SELECT
                (AVG(DATE_PART('day', ended_at::timestamp - started_at::timestamp) * 24 +
                DATE_PART('hour', ended_at::timestamp - started_at::timestamp)) -
                STDDEV_SAMP(DATE_PART('day', ended_at::timestamp - started_at::timestamp) * 24 +
                DATE_PART('hour', ended_at::timestamp - started_at::timestamp)) * 3)
            FROM public.divvy_tripdata_2021)
        AND
        DATE_PART('day', ended_at::timestamp - started_at::timestamp) * 24 +
        DATE_PART('hour', ended_at::timestamp - started_at::timestamp) <
            (SELECT
                (AVG(DATE_PART('day', ended_at::timestamp - started_at::timestamp) * 24 +
                DATE_PART('hour', ended_at::timestamp - started_at::timestamp)) +
                STDDEV_SAMP(DATE_PART('day', ended_at::timestamp - started_at::timestamp) * 24 +
                DATE_PART('hour', ended_at::timestamp - started_at::timestamp)) * 3)
            FROM public.divvy_tripdata_2021))
GROUP BY member_casual;
```

The result:

	member_casual character varying (100)	count bigint	max_hour_outlier double precision	min_hour_oitlier double precision
1	member	1285	25	9
2	casual	7351	932	9

The result shows, there are 8636 outlier data with maximum Values 932 hour or 38 days for using bike and most of them coming from 'Casual Members'. In this case, I will use one way to deal with outliers to know the result of future analysis by removing all of outliers.

ANALYSIS AND SHARE

It's time to analyze our cleaned data using R. R is the best option when we analyze and visualize data in one platform. First, I loaded necessary libraries and read the csv file. I use 'fread' function from 'data.table' library, instead of 'read.csv' function. 'Fread' function has faster and more convenient capability to import a large file. The name of data frame for this dataset is 'trip_data'.

```
> #loading and import the csv file
> library(tidyverse)
-- Attaching packages --------------------------------------- tidyverse 1.3.1 --
v ggplot2 3.3.5     v purrr   0.3.4
v tibble  3.1.6     v dplyr   1.0.7
v tidyr   1.2.0     v stringr 1.4.0
v readr   2.1.2     v forcats 0.5.1
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
> library(dplyr)
> library(readr)
> library(data.table)
data.table 1.14.2 using 1 threads (see ?getDTthreads).  Latest news: r-datatable.co
m

Attaching package: 'data.table'

The following objects are masked from 'package:dplyr':

    between, first, last

The following object is masked from 'package:purrr':

    transpose

> trip_data <- fread("C:/Users/MOH.FARIZ/Documents/Certification Course/Google
  Data Science/Case Study/Final_Data_Without_Outliers.csv", header = TRUE, sep = "
  ,")
```

Here is the summary of trip_data:

```
   ride_id          rideable_type        started_at
 Length:5585774     Length:5585774     Min.   :2021-01-01 00:02:05
 Class :character   Class :character   1st Qu.:2021-06-07 01:57:34
 Mode  :character   Mode  :character   Median :2021-08-01 03:12:23
                                       Mean   :2021-07-29 08:11:15
                                       3rd Qu.:2021-09-24 16:46:45
                                       Max.   :2021-12-31 23:59:48

    ended_at                  start_station_name start_station_id
 Min.   :2021-01-01 00:08:39   Length:5585774     Length:5585774
 1st Qu.:2021-06-07 02:49:29   Class :character   Class :character
 Median :2021-08-01 03:38:34   Mode  :character   Mode  :character
 Mean   :2021-07-29 08:30:00
 3rd Qu.:2021-09-24 17:03:51
 Max.   :2022-01-01 03:59:48

 end_station_name   end_station_id       start_lat       start_lng
 Length:5585774     Length:5585774     Min.   :41.64   Min.   :-87.84
 Class :character   Class :character   1st Qu.:41.88   1st Qu.:-87.66
 Mode  :character   Mode  :character   Median :41.90   Median :-87.64
                                       Mean   :41.90   Mean   :-87.65
                                       3rd Qu.:41.93   3rd Qu.:-87.63
                                       Max.   :42.07   Max.   :-87.52

    end_lat         end_lng       member_casual
 Min.   :41.39   Min.   :-88.97   Length:5585774
 1st Qu.:41.88   1st Qu.:-87.66   Class :character
 Median :41.90   Median :-87.64   Mode  :character
 Mean   :41.90   Mean   :-87.65
 3rd Qu.:41.93   3rd Qu.:-87.63
 Max.   :42.17   Max.   :-87.49
 NA's   :1683    NA's   :1683
> |
```

Before starting the analysis, I want to create three new variables in the data frame. These new variables will allow us to make our analysis easier.

These variables are

trip_data: distance of trip in meter. I use distHaversine function from geosphere library to extract the actual distance between two points of lon-lat data.

```
> library(geosphere)
> trip_data <- trip_data %>% mutate(dist_in_m = distHaversine(cbind(start_lng, star
t_lat), cbind(end_lng, end_lat)))
> summary(trip_data$dist_in_m)
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.     NA's
     0.0    901.7   1639.7   2188.7   2884.9 114511.7     1683
>
```

duration_in_min: duration of trip in minutes as numeric data. I use difftime function from lubridate library to extract the time in minutes.

```
> library(lubridate)

Attaching package: 'lubridate'

The following objects are masked from 'package:data.table':

    hour, isoweek, mday, minute, month, quarter, second, wday, week,
    yday, year

The following objects are masked from 'package:base':

    date, intersect, setdiff, union

> trip_data <- trip_data %>% mutate(duration_min = as.numeric(difftime(trip_data$en
ded_at, trip_data$started_at, unit = "mins")))
> summary(trip_data$duration_min)
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
  0.0167   6.7333  11.9833  18.7575  21.7000 539.9000
```

day_of_week: the day of date time.

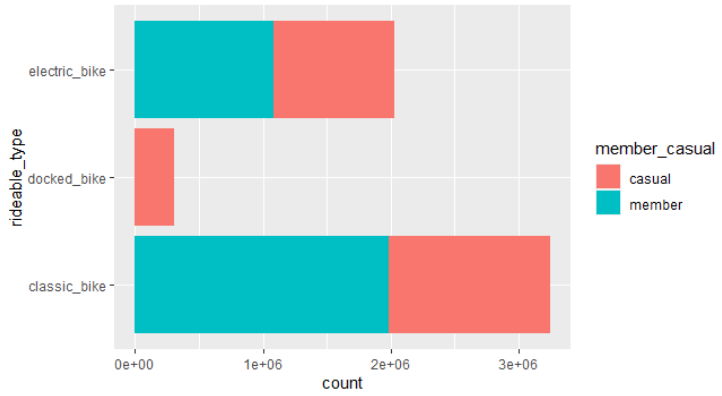
```
> trip_data$date <- as.Date(trip_data$started_at)
> trip_data$day_of_week <- format(as.Date(trip_data$date), "%A")
> summary(trip_data$day_of_week)
   Length     Class      Mode
  5585774 character character
> unique(trip_data$day_of_week)
[1] "Saturday"  "Wednesday" "Thursday"  "Monday"    "Sunday"    "Friday"
[7] "Tuesday"
>
```

Now let's start our analysis. First, for the basic one. Let's start to know the comparison of total count between casual and member.

```
> Count_Member
   member_casual       n
1:        member 3064461
2:        casual 2521313
> print ("Percentage conversion")
[1] "Percentage conversion"
> percent(Count_Member$n/sum(Count_Member$n))
[1] "54.9%" "45.1%"
```

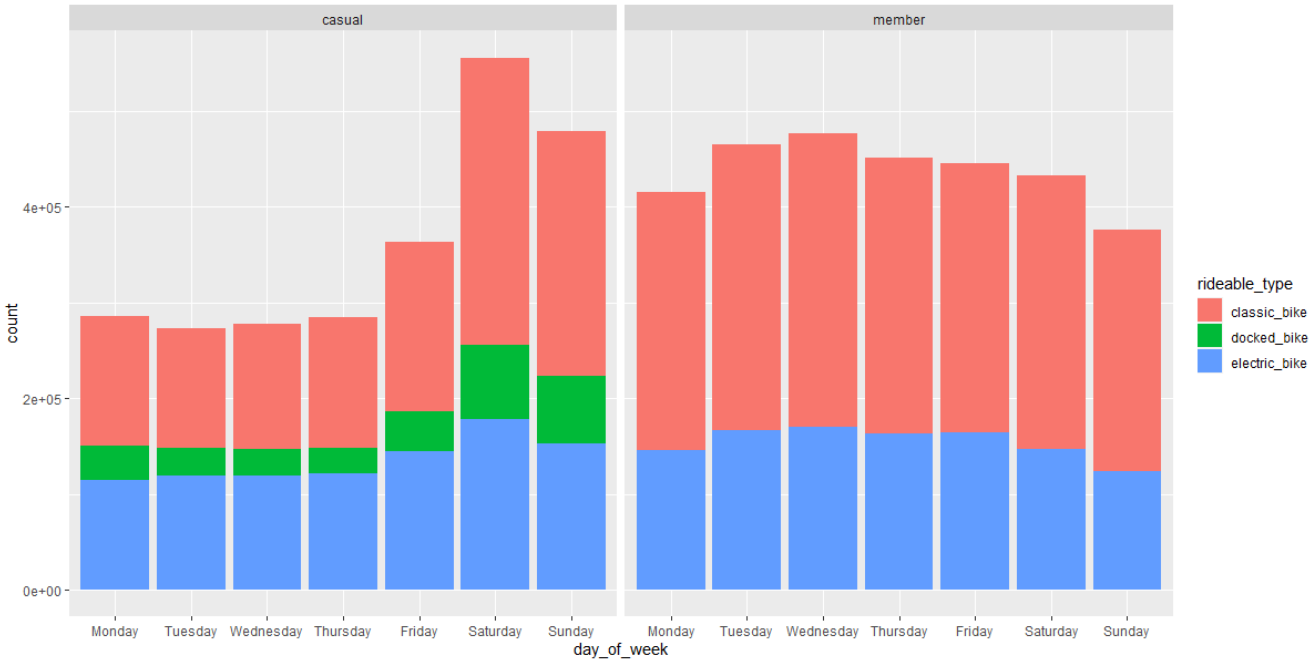
The result shows, the member users have larger percentage than casual from the total users. For the notes, it's not the actual member that represent the actual data. The data itself has the LIMITATION that all the transaction did not include the user id. So, the assumption is all the transaction is unique for each user.

Next, plot the bar chat to create first analysis to compare the casual and member in their distribution of rideable type.



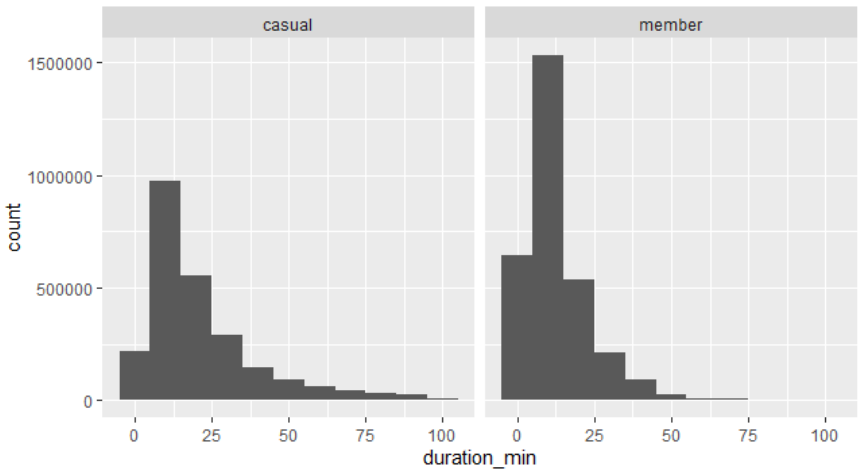
This plot shows that us member users mostly being dominant by using classic bike and little bit higher for using electric bike than casual users. In other hand, docked bike only being used by casual users.

Let’s move the analysis by the day of week.

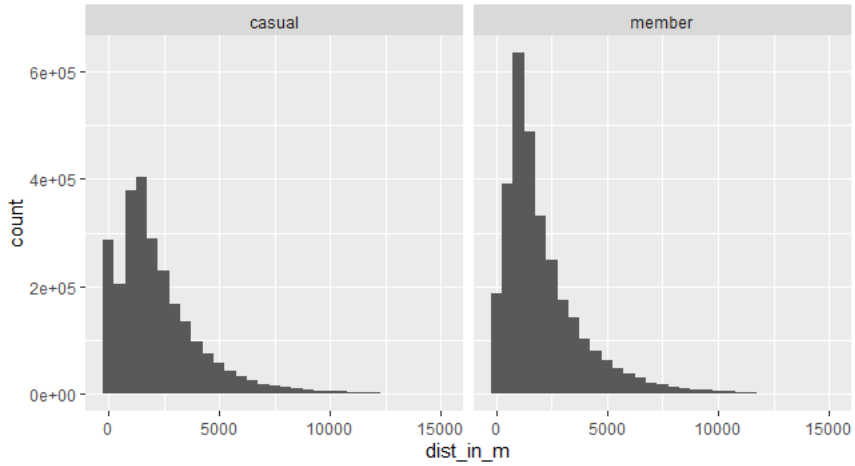


The plot shows us that member users use the service higher than casual users on weekdays. For the member users the usage services on weekdays are relatively same and even lower on weekends. In other hand, casual users use the service mostly at weekend, especially on Saturday. The usages on weekdays are relatively lower than on weekend for the users. We can assume if the reason for members using the services is for supporting daily activities like working and for casual users is for entertainment during the weekends.

Now let’s observe the duration in minutes between casual and member users using histogram. The result show, that the casual users take longer duration than the member users. But the maximum of duration is longer for member users.



Still using the histogram, it hard to observe the detail of difference between casual and member users for their distance of trip. But we can shortly conclude, if the member users have the highest count of distance traveled than casual users. In other hand, casual users seem to have the highest distance of traveled. Let’s breakdown it further.



First, I make the filtered variable among the member and casual users. Then I breakdown the summary of each user for their day of week, duration, and distance. Here is the clear summary of each user:

```
> filtered_member <- filter(trip_data, member_casual=="member")
> filtered_casual <- filter(trip_data, member_casual=="casual")
> summary(drop_na(select(filtered_member, c('day_of_week', 'dist_in_m', 'duration_m
in'))))
    day_of_week       dist_in_m        duration_min
 Monday   :415889   Min.   :    0.0   Min.   :  0.0167
 Tuesday  :465218   1st Qu.:  878.2   1st Qu.:  5.5667
 Wednesday:476861   Median : 1558.1   Median :  9.6000
 Thursday :451224   Mean   : 2129.0   Mean   : 13.1519
 Friday   :446080   3rd Qu.: 2778.3   3rd Qu.: 16.5667
 Saturday :432689   Max.   :32058.7   Max.   :539.9000
 Sunday   :375792
> summary(drop_na(select(filtered_casual, c('day_of_week', 'dist_in_m', 'duration_m
in'))))
    day_of_week       dist_in_m         duration_min
 Monday   :285408   Min.   :     0.0   Min.   :  0.0167
 Tuesday  :273519   1st Qu.:   946.2   1st Qu.:  9.0500
 Wednesday:278151   Median :  1731.6   Median : 15.9167
 Thursday :285169   Mean   :  2261.4   Mean   : 25.4987
 Friday   :362833   3rd Qu.:  3010.9   3rd Qu.: 29.0833
 Saturday :556012   Max.   :114511.7   Max.   :539.8667
 Sunday   :479246
```

The result shows some points:

- The highest peak for member users in day of week is Wednesday and for casual users is Saturday.
- The mean of distance is quietly similar although it is higher for casual users than member users. The maximum distance for casual users is far higher than member users.
- The casual users also have the longer mean of duration for using the services than member users. Both of users almost have the same maximum duration for services.

Next, I want to know the most popular starting stations between member and casual users.

```
> head(count(filtered_member, start_station_name, sort=TRUE), n=10)
          start_station_name      n
 1:                          373106
 2:        Clark St & Elm St  24725
 3:    Wells St & Concord Ln  23703
 4: Kingsbury St & Kinzie St  23547
 5:        Wells St & Elm St  21014
 6:    Dearborn St & Erie St  19581
 7:      Wells St & Huron St  19180
 8:   St. Clair St & Erie St  18893
 9:     Broadway & Barry Ave  17795
10:  Clinton St & Madison St  16906
> head(count(filtered_casual, start_station_name, sort=TRUE), n=10)
          start_station_name      n
 1:                          317678
 2:   Streeter Dr & Grand Ave  66136
 3:          Millennium Park  33429
 4:     Michigan Ave & Oak St  29709
 5:           Shedd Aquarium  23190
 6:       Theater on the Lake  21301
 7:     Wells St & Concord Ln  19849
```

The result looks interesting. The count of starting stations for member users is almost same for each station. But the difference for the Top Popular starting station than others for casual users is far higher. Streeter Dr & Grand Ave is the most popular starting station for casual members.

CONCLUSION

From the previous analysis, it's still hard to give the best suggestion based on current analysis because the data have the critical limitation, especially from the uniqueness of each transaction from each user. However, we still have some useful conclusions that might be applied for future decisions. Let's summarize what we got:

- Classic bike is the most favorites choice for both casual and member users, but the largest user for this service is member users. Docked bike is the only used by casual users. Both of users quietly love electric bike service.
- The usage of service for member users is mostly during weekdays and it peak is Wednesday. For the casual users is during weekend and its peak is Saturday.
- The mean duration and distance of trip for casual users is longer than member users. A great opportunity if we can convert these users.
- The count of starting station name almost similar for member users but for casual users, there is a favorite starting station to use these services. That is Streeter Dr & Grand Ave station.

According to the previous conclusions, my suggestions to answer the second question which is converting casual to member users are:

- Grabbing momentum during the weekend in the most popular starting station. Placing the promotion like event/programs and other advertisement are the best movements like in Streeter Dr & Grand Ave station.
- Dive the best pricing for using docked bike to targeting casual users who join membership since we know docked bike is the popular service on casual users.
- We can make a gamification program like exclusive 'quest' for member that reach certain distance and duration trip with special rewards.
- During the process, we found if there are huge numbers of missing and outlier data from casual members that using electric bike service. It should be concern for the future recording.