Data Analysis on Bike Share Trips

Reza Farshchi

2023-05-05

Table of content:

Abstract:

This is the full report of the analysis done on the open-source dataset related to the previous 12 months of Cyclistic trip data. Cyclistic is a bike-share company and the goal of this analysis is to suggest executive team with recommendations based on data to generate more revenue for the company. More details about the goal and the business task if provided in the ask section. My approach to applying data analysis on different tasks is generally digested in 6 steps: Ask, Prepare, Process, analyze, share, act. Although there are other approaches to achieve data-driven solutions, the content of most of these approaches are almost the same so we will stick to the mentioned approach in this analysis. The goal of each phase comes first in that section and after that we will explain the approach and actions done in that phase.

Ask:

Goal

Ask phase is a crucial phase in our analysis. The goal of this phase is actually understanding the goal of the whole project. In this part of the project we focus on the problem we are trying to solve. Making sure that we are on the same page with the stakeholders is important in this phase too, Because we are trying to give them recommendations based on data. So if we misunderstand their expectations we might waste a big part of the teams time and resources. The main goal of this phase is to get a clear statement of the business task.

Approach:

There are 3 types of pricing plans for the company's services: single-ride passes, full-day passes, and annual memberships The director of marketing believes the company's future success depends on maximizing the number of annual memberships. Therefore, our team wants to understand how casual riders and annual members use Cyclistic bikes differently. So as a data analyst we are facing this single question as our business task: How do annual members and casual riders use Cyclistic bikes differently? This phase of the analysis is almost under influence of our manager's opinion, so we don't dig deeper in this phase. But the whole idea of this phase is to get a clear statement of the business task. We get to this point by focusing on these two main things. The problem we are trying to solve and considering the key stakeholders and their expectations.

Prepare:

Goal:

In this phase of our analysis, we try to find the most reliable and related source(s) of data and organize them into a rational order to reach the answer to our main question.

Approach:

After searching for a source of data that has the ROCCC traits (Reliable, Original, Comprehensive, Current and Cited) we found the data of bike-share sessions for each month. I decided to use most recent 12 months data for this analysis. The source of this csv files is here(202204->202303) Now that we have the reliable source of data it's time for the data organization. We have 12 csv files including 12 months of data for the bike usage. It is better to merge them all in one csv file. We used following code in R to merge them all in one data frame and then one csv file.

```r
# first we read all of our 12 csv files.
d1 <- read.csv("202204-divvy-tripdata.csv")
d2 <- read.csv("202205-divvy-tripdata.csv")
d3 <- read.csv("202206-divvy-tripdata.csv")
d4 <- read.csv("202207-divvy-tripdata.csv")
d5 <- read.csv("202208-divvy-tripdata.csv")
d6 <- read.csv("202209-divvy-tripdata.csv")
d7 <- read.csv("202210-divvy-tripdata.csv")
d8 <- read.csv("202211-divvy-tripdata.csv")
d9 <- read.csv("202212-divvy-tripdata.csv")
d10 <- read.csv("202301-divvy-tripdata.csv")
d11 <- read.csv("202302-divvy-tripdata.csv")
d12 <- read.csv("202303-divvy-tripdata.csv")

# then we create a function to check if all the column names of two data frames
# are equal or not. We want to make sure all of our files are equal at least in
# terms of column names.
my_func <- function(x, y) {
    for (i in names(x)) {
        if (!(i %in% names(y))) {
            print("Warning: Names are not the same")
            break
        } else if (i == tail(names(y), n = 1)) {
            print("Names are identical")
        }
    }
}

# now we use our function with our 12 data frames to check if they have equal
# column names or not. (they do.)
my_func(d1, d2)
```

```
## [1] "Names are identical"
```

```r
my_func(d1, d3)
```

```
## [1] "Names are identical"
```

```r
my_func(d1, d4)
```

```
## [1] "Names are identical"
```

```r
my_func(d1, d5)
```

```
## [1] "Names are identical"
```

```r
my_func(d1, d6)
```

```
## [1] "Names are identical"
```

```r
my_func(d1, d7)
```

```
## [1] "Names are identical"
```

```r
my_func(d1, d8)
```

```
## [1] "Names are identical"
```

```r
my_func(d1, d9)
```

```
## [1] "Names are identical"
```

```r
my_func(d1, d10)
```

```
## [1] "Names are identical"
```

```r
my_func(d1, d11)
```

```
## [1] "Names are identical"
```

```r
my_func(d1, d12)
```

```
## [1] "Names are identical"
```

```r
# now we create a new data frame using all of our existing data frames.
df_list <- list(d1, d2, d3, d4, d5, d6, d7, d8, d9, d10, d11, d12)
main_df <- Reduce(function(x, y) merge(x, y, all = TRUE), df_list)

# and in the end we save our integrated data frame somewhere to use later in
# next phases of the analysis.
write.csv(main_df, "C:\\Users\\reza farshchi\\OneDrive\\Desktop\\main_df.csv", row.names = FALSE)
```

Now that we have all the data organized in one csv file we can head to the next phase of analysis to clean the data and make it ready for the analysis.

Process:

Goal:

In this phase of the analysis, we make sure that we are all set start the analysis on the dataset. In this step we make sure that our data has integrity and is all clean. Also, we choose the tools we are about to use in this step.

Approach:

Our goal is to spot human error or inconsistency that exists within our data and make them disappear. Following code is used to clean our data. Every needed detail is mentioned in the code as comments.

```r
# Since we already have 'dplyr' installed, we don't install it and just call
# the library() function to use it
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
# first we read the csv file we already have created in the prepration phase
df <- read.csv("main_df.csv")

# we call glimpse()(or str() ) function to see our data frame's column names
# along with their data types
glimpse(df)
```

```
## Rows: 5,803,720
## Columns: 13
## $ ride_id            <chr> "00000179CF2C4FB5", "0000038F578A7278", "0000047373~
## $ rideable_type      <chr> "electric_bike", "electric_bike", "electric_bike", ~
## $ started_at         <chr> "2022-07-28 09:02:27", "2022-12-15 15:36:06", "2022~
## $ ended_at           <chr> "2022-07-28 09:13:51", "2022-12-15 15:44:35", "2022~
## $ start_station_name <chr> "Ellis Ave & 55th St", "Lakeview Ave & Fullerton Pk~
## $ start_station_id   <chr> "KA1504000076", "TA1309000019", "13022", "13193", "~
## $ end_station_name   <chr> "Ellis Ave & 55th St", "Clark St & Elm St", "Calume~
## $ end_station_id     <chr> "KA1504000076", "TA1307000039", "13102", "TA1305000~
## $ start_lat          <dbl> 41.79424, 41.92583, 41.89221, 41.92185, 41.90000, 4~
## $ start_lng          <dbl> -87.60134, -87.63882, -87.61183, -87.64402, -87.690~
## $ end_lat            <dbl> 41.79430, 41.90297, 41.85761, 41.90312, 41.91000, 4~
## $ end_lng            <dbl> -87.60145, -87.63128, -87.61941, -87.67394, -87.690~
## $ member_casual      <chr> "casual", "casual", "casual", "casual", "member", "~
```

```r
# now we examine different aspects of these data alone and in relation to each
# other to find out if there is anything to clean in the data frame

# check what columns include any null value
null_cols <- names(df)[colSums(is.na(df)) > 0]
print(null_cols)
```

```
## [1] "end_lat" "end_lng"
```

```r
# we sort our data frame based on started_at to have the trips in order of
# occurrence which seems valid for now, but might need to be changed later.
df <- df %>%
    arrange(started_at)

# we move forward in the order above. Let's see how many unique 'ride_id's
# exist
length(unique(df$ride_id))
```

```
## [1] 5803720
```

```r
# [1] 5803720 total unique 'ride_id' are equal to total rows.And its data type
# seems valid. Pretty sane so far.

# Let's check out how many 'rideable_type's are out there
unique(df$rideable_type)
```

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

```r
# There is no sign of human error in this column. This column looks valid
# alone. We might check its validity in comparison to other columns later, but
# it is good to go to the next column for now.

# started_at and ended_at columns are both has datetime information but are
# type character. We convert them to datetime format and also create a new
# column named duration based on the difference in started_at and ended_at.
df$started_at <- as.POSIXct(df$started_at, format = "%Y-%m-%d %H:%M:%S")
df$ended_at <- as.POSIXct(df$ended_at, format = "%Y-%m-%d %H:%M:%S")

# now when we call glimpse these two columns type are <dttm> as expected. Let's
# create duration column using mutate() function

df <- df %>%
    mutate(duration = difftime(ended_at, started_at, units = "mins"))

# now that we have duration column along with started_at and ended_at with
# correct type, we check some of the logical aspects of these data We know that
# the data we are analyzing is related to 2022-04 to 2023-03. Let's check
# maximum and minimum values of started_at to see if it fits in the expected
# timeline.
max_started_at <- max(df$started_at)
min_started_at <- min(df$started_at)
print(max_started_at)
```

```
## [1] "2023-03-31 23:59:28 +0330"
```

```r
print(min_started_at)
```

```
## [1] "2022-04-01 00:01:48 +0430"
```

```r
# Seems like it fits in our expected timeline.

# lets check out different values of duration
max_duration <- max(df$duration)
min_duration <- min(df$duration)
print(max_duration)
```

```
## Time difference of 41387.25 mins
```

```r
print(min_duration)
```

```
## Time difference of -10413.35 mins
```

```r
# under zero values are for sure invalid. We assume that in these cases
# started_at and ended_at records are swapped. we exchange them in these cases
# and also make duration value positive. For maximum amount the total duration
# minutes seems a lot. So we check out more rows to check if these amounts are
# mistaken or fine.

# we try to check the most 10 duration's started_at and ended_at
sorted_df <- df[order(-df$duration), ]
result_df <- sorted_df[1:10, c("started_at", "ended_at")]
print(result_df)
```

```
##                  started_at            ended_at
## 4099592 2022-10-01 15:04:38 2022-10-30 08:51:53
## 458848  2022-05-08 00:28:53 2022-06-02 04:46:41
## 1338536 2022-06-15 07:56:59 2022-07-10 04:57:37
## 1977009 2022-07-09 01:02:46 2022-08-01 19:11:35
## 1977017 2022-07-09 01:03:19 2022-08-01 19:11:26
## 1977008 2022-07-09 01:02:45 2022-08-01 18:51:57
## 5207195 2023-01-08 11:08:52 2023-01-31 19:12:36
## 1225248 2022-06-10 16:13:34 2022-07-03 04:16:32
## 1877441 2022-07-04 18:37:11 2022-07-27 00:32:38
## 4253010 2022-10-09 11:24:11 2022-10-31 04:33:40
```

```
# after checking these times out we realize that this pattern of renting bikes
# for around 1 month is repeated. So we ignore this pattern and assume that
# these records are valid.

# swap values and make duration positive using a temp value function
neg_dur_mask <- df$duration < 0
tmp <- df$started_at[neg_dur_mask]
df$started_at[neg_dur_mask] <- df$ended_at[neg_dur_mask]
df$ended_at[neg_dur_mask] <- tmp
# and after swapping started_at and ended at in cases of negative duration we
# make all duration values positive
df$duration <- abs(df$duration)
# now if we print minimum amount of duration it shows 0! Another cleaning task.

# let's count how many of them have duration of 0
sum(df$duration == 0)
```

```
## [1] 440
```

```
# the result is 440. at least it is not all of them and the duration is
# calculated right. 440 is very very small number in comparison to 5803720! so
# it is completely fine to remove these rows with duration of 0 since they
# might also have other problems that we might not find.

# to remove observations with duration of 0, we create a subset of df with all
# observations with duration!=0 and replace this subset with our original df
df <- subset(df, duration != 0)
# finally our cleaning based on started_at, ended_at, and duration is over. We
# can move forward to next column, start_station_name!

# first we monitor how many unique station names are in starting point.
length(unique(df$start_station_name))
```

```
## [1] 1700
```

```
# turns out there are 1700 of them. Cannot check if all of them seem to be
# valid or not. We already checked if there are any null value in this column
# and have seen there are none. But lets check if there are any of them with
# less than 4 characters.

# we want to find all unique 'start_station_name's with less than 4 characters.
# which might be wrong station names. we use [ ] to find subsets of df with
# just mentioned condition.
short_names <- unique(df$start_station_name[nchar(df$start_station_name) < 4])
print(short_names)
```

```
## [1] ""
```

```
# we see that with this condition there is '' station name. Let's see how many
# of them are with '' station name.
sum(df$start_station_name == "")
```

```
## [1] 839079
```

```
# turns out that 839079 of all 5803280 remaining observations are started from
# '' station. about 14.4% of all observations. Clearly we cannot remove all
# these rows. Also it might not be invalid value for station name. Let's check
# if this station name is repeated in end_station_name too or not.

short_names <- unique(df$end_station_name[nchar(df$end_station_name) < 4])
print(short_names)
```

```
## [1] ""
```

```
sum(df$end_station_name == "")
```

```
## [1] 896132
```

```
# And yes. we have 15.4% of all remaining rows with end_station_name of ''.
# After searching about this condition we realize that you can scan any bike in
# the city and use it, not just the bikes that are in the stations. And also
# you can leave the bikes anywhere in the city not just the stations. So now
# all these '' stations make sense and we can move on to the next columns.

# Let's see unique values of member_casual values using unique() function
unique(df$member_casual)
```

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

```
# the result is 'member' 'casual'. These values are as expected. It doesn't
# seem to need cleaning. Since it is a the main column that we want to
# investigate about this data frame, the fact that this column is already clean
# is a good news for us, which makes our findings more reliable.

# Considering the question of our analysis it seems like having day of the week
# of the trip in the data set might help later. So here we create it using the
# mutate() and weekdays() functions.
df <- df %>%
    mutate(day_of_week = weekdays(df$started_at))

# We already know we have null values in end_lat and end_lng, Let's see how
# many of these observations have either of these columns as null using is.na()
# function.
num_null <- sum(is.na(df$end_lng) | is.na(df$end_lat))
print(num_null)
```

```
## [1] 5855
```

```
# 5855 observations have one of these columns as null. this value is about 0.1%
# of all the observations. We can delete ro impute these rows. Let's delete
# them.

# we use complete_cases() function to keep rows with no missing values in
# end_lat or end_lng
df <- df[complete.cases(df[, c("end_lng", "end_lat")]), ]

# We skip cleaning start_station_id and end_station_id since we will work with
# the names and there is no added value in cleaning or even keeping them. we
# remove them from our data frame.
df <- select(df, -end_station_id, -start_station_id)

# One last look at our data set.
str(df)
```

```
## 'data.frame':    5797425 obs. of  13 variables:
##  $ ride_id           : chr  "AFDC60E4BD0755EB" "6DFBE82AEF4187C0" "3A2D92FB6FBAD9EE" "B0AD95376AEB3B
##  $ rideable_type     : chr  "electric_bike" "electric_bike" "electric_bike" "electric_bike" ...
##  $ started_at        : POSIXct, format: "2022-04-01 00:01:48" "2022-04-01 00:01:53" ...
##  $ ended_at          : POSIXct, format: "2022-04-01 00:07:46" "2022-04-01 00:02:15" ...
##  $ start_station_name: chr  "Base - 2132 W Hubbard Warehouse" "Kedzie Ave & 48th Pl" "" "Michigan Av
##  $ end_station_name  : chr  "" "Kedzie Ave & 48th Pl" "" "Lakeview Ave & Fullerton Pkwy" ...
##  $ start_lat         : num  41.9 41.8 41.8 41.9 42 ...
##  $ start_lng         : num  -87.7 -87.7 -87.6 -87.6 -87.7 ...
##  $ end_lat           : num  41.9 41.8 41.8 41.9 42 ...
##  $ end_lng           : num  -87.6 -87.7 -87.6 -87.6 -87.7 ...
##  $ member_casual     : chr  "member" "casual" "member" "casual" ...
##  $ duration          : 'difftime' num  5.96666666666667 0.366666666666667 8.35 33.7 ...
##   ..- attr(*, "units")= chr "mins"
##  $ day_of_week       : chr  "Friday" "Friday" "Friday" "Friday" ...
```

```
# Looks like we have our data set organized and cleaned and we are ready for
# analysis!

# Let's save our clean version of data frame in a csv file.
write.csv(df, "C:\\Users\\reza farshchi\\OneDrive\\Desktop\\clean_df.csv", row.names = FALSE)
```

Analyze:

Goal:

Now that we have our data cleaned and organized, it's time for the analyze phase. In this part of our project, we start calculations and trying to find trends, relations, differences, and similarities in our data.

Approach:

We try to find the differences that exists in the data of casual and premium users. We use R language for this phase too. The code we use to solve our problem is here. Every detail about every line of code is mentioned in the comments.

```
library(dplyr)
library(lubridate)
```

```
##
## Attaching package: 'lubridate'

## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

```
df <- read.csv("clean_df.csv")

# Let's take a quick look at column names and think about our different options
# to find differences between member users and casual users
colnames(df)
```

```
##  [1] "ride_id"            "rideable_type"      "started_at"
##  [4] "ended_at"           "start_station_name" "end_station_name"
##  [7] "start_lat"          "start_lng"          "end_lat"
## [10] "end_lng"            "member_casual"      "duration"
## [13] "day_of_week"
```

```
# Let's count trips by membership type and rideable type
rideable_counts <- table(df$member_casual, df$rideable_type)

# And Calculate percentages by membership type
rideable_percentages <- prop.table(rideable_counts, 1) * 100

print(rideable_counts)
```

```
##
##          classic_bike docked_bike electric_bike
##   casual       887308      171121       1273633
##   member      1748921           0       1716442
```

```
print(rideable_percentages)
```

```
##
##          classic_bike docked_bike electric_bike
##   casual    38.048217    7.337755     54.614028
##   member    50.468623    0.000000     49.531377
```

```
# So one insight is that none of members use docked_bikes, and in comparison to
# casual users they are more interested in classic bikes rather that electric
# ones.

# Count trips by start station and membership type
start_station_counts <- as.data.frame(table(df$start_station_name, df$member_casual))
start_station_counts <- start_station_counts[order(-start_station_counts$Freq), ]
```

```r
# Count trips by end station and membership type
end_station_counts <- as.data.frame(table(df$end_station_name, df$member_casual))
end_station_counts <- end_station_counts[order(-end_station_counts$Freq), ]
print(head(start_station_counts, 10))
```

```
##                                    Var1   Var2   Freq
## 1701                                    member 491971
## 1                                       casual 347108
## 1541            Streeter Dr & Grand Ave casual  56770
## 356   DuSable Lake Shore Dr & Monroe St casual  31333
## 2311           Kingsbury St & Kinzie St member  25266
## 792                     Millennium Park casual  25060
## 785               Michigan Ave & Oak St casual  24932
## 357  DuSable Lake Shore Dr & North Blvd casual  23272
## 1941                  Clark St & Elm St member  22956
## 3309              Wells St & Concord Ln member  21951
```

```r
print(head(end_station_counts, 10))
```

```
##                                    Var1   Var2   Freq
## 1725                                    member 488206
## 1                                       casual 402071
## 1564            Streeter Dr & Grand Ave casual  59030
## 358   DuSable Lake Shore Dr & Monroe St casual  28958
## 795                     Millennium Park casual  26420
## 788               Michigan Ave & Oak St casual  26133
## 359  DuSable Lake Shore Dr & North Blvd casual  25789
## 2337           Kingsbury St & Kinzie St member  25280
## 1967                  Clark St & Elm St member  23259
## 3359              Wells St & Concord Ln member  22647
```

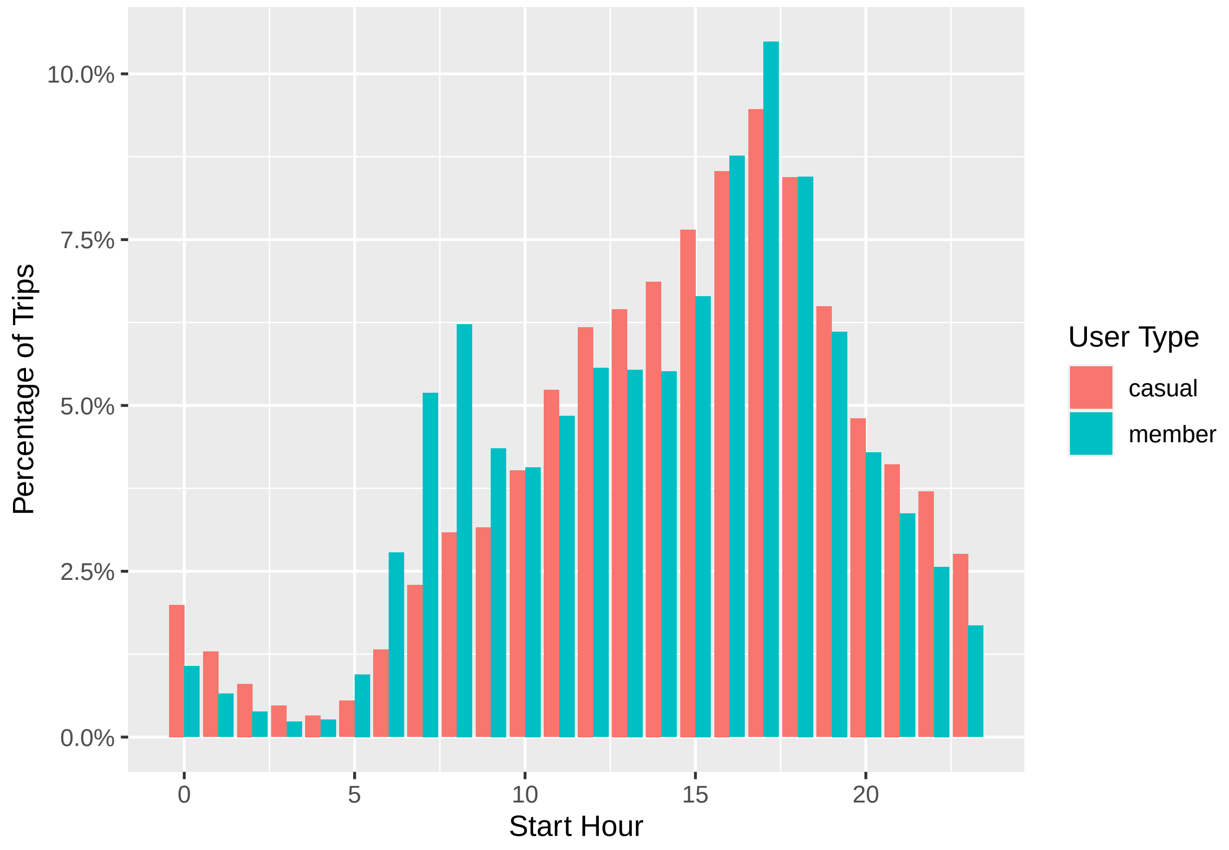
```r
# Most popular for both start and end stations for casual members are 'Streeter
# Dr & Grand Ave' station. And most popular station for both start and end for
# members is 'Kingsbury St & Kinzie St' station. There is a good opportunity of
# marketing in the first station and a good opportunity of study on why members
# choose second station.


# using summarize() to calculate average trip duration for members and casual
# users.
df %>%
    group_by(member_casual) %>%
    summarize(avg_trip_duration = mean(duration), median_trip_duration = median(duration))
```

```
## # A tibble: 2 x 3
##   member_casual avg_trip_duration median_trip_duration
##   <chr>                     <dbl>                <dbl>
## 1 casual                     21.4                12.6
## 2 member                     12.2                 8.67
```

```
# now we want to visualize what percentage of trips for each user type started
# in what hour of the day.
df_summary <- df %>%
    mutate(start_hour = hour(started_at)) %>%
    group_by(member_casual) %>%
    mutate(total_trips = n()) %>%
    group_by(start_hour, member_casual, total_trips) %>%
    summarise(count = n()) %>%
    mutate(percentage = count/total_trips)
```

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

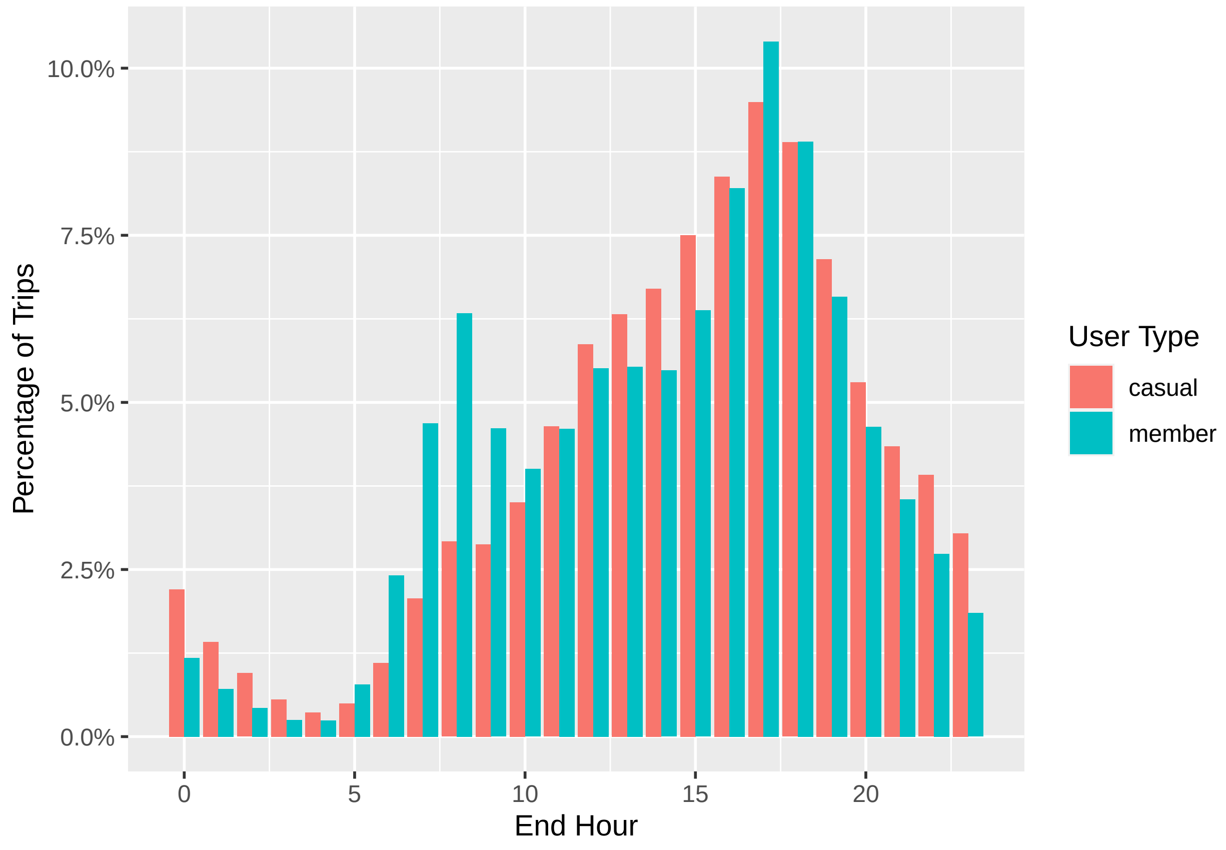
```
library(ggplot2)
ggplot(df_summary, aes(x = start_hour, y = percentage, fill = member_casual)) + geom_bar(stat = "identi
    position = "dodge") + scale_y_continuous(labels = scales::percent) + labs(x = "Start Hour",
    y = "Percentage of Trips", fill = "User Type")
```



```
# now we go through same steps to gain same insights for end hour of the trips
# for each user type.
df_summary <- df %>%
    mutate(end_hour = hour(ended_at)) %>%
    group_by(member_casual) %>%
    mutate(total_trips = n()) %>%
    group_by(end_hour, member_casual, total_trips) %>%
    summarise(count = n()) %>%
    mutate(percentage = count/total_trips)
```

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

```
library(ggplot2)
ggplot(df_summary, aes(x = end_hour, y = percentage, fill = member_casual)) + geom_bar(stat = "identity
    position = "dodge") + scale_y_continuous(labels = scales::percent) + labs(x = "End Hour",
    y = "Percentage of Trips", fill = "User Type")
```



```
# With these charts being analyzed we can act with more efficiency. If the
# marketing team wants to plan a marketing campaign for casual users to make
# them join the annual members the best time of the day is 16-19 if the
# campaign takes 3 hours or 17-18 if it is one hour long. Earlier we discovered
# the most popular stations. So we kinda discovered the best time and place for
# the most efficient marketing campaign.

# let's take a look at frequency and percentage of trips for both user types in
# week.
df %>%
    group_by(member_casual) %>%
    count(day_of_week) %>%
    mutate(percent = n/sum(n) * 100)
```

```
## # A tibble: 14 x 4
## # Groups:   member_casual [2]
##    member_casual day_of_week      n percent
##    <chr>         <chr>        <int>   <dbl>
##  1 casual        Friday      340231    14.6
##  2 casual        Monday      275070    11.8
##  3 casual        Saturday    470309    20.2
##  4 casual        Sunday      388419    16.7
##  5 casual        Thursday    311116    13.3
##  6 casual        Tuesday     270801    11.6
##  7 casual        Wednesday   276116    11.8
##  8 member        Friday      489715    14.1
##  9 member        Monday      484446    14.0
## 10 member        Saturday    452001    13.0
## 11 member        Sunday      397878    11.5
## 12 member        Thursday    551020    15.9
## 13 member        Tuesday     544196    15.7
## 14 member        Wednesday   546107    15.8
```

```
# Okay we had the best place and time of the day for our marketing event. What
# if we hold it on Saturday when most of our casual users are expected to take
# a trip.
```

Share

Goal:

In this phase we are supposed to share our findings to stakeholders and suggest them some ideas about the question they asked. In this case our stakeholders wanted to find out the differences between the casual and member users records so they can encourage casual users to become member. We analyzed the data and found few insights.

Approach

The best way to share our findings is through a presentation to the stakeholders. The executive team are not interested in a report as long as this one. They have limited time and we should give them the insights in fast, direct way that is backed with data. We create a presentation and it can be found in another file named "presentation" where you downloaded this one.

Act

Goal

This is that last step in our analysis and we should act upon our findings. Executive/marketing team will act upon our insights to hold a marketing event on our suggested time/location. But our team can focus on other things like further investigations.

Approach

Data team can work on creating new data sets to move this study to next level. One suggestion might be more columns in the data set. One thing that has been missing in this data set is the timeline of each user. This data could show us insights about what makes our users join our annual membership.