Cyclistic Case Study Report

P. Katekomol

11/4/2021

Overview

This report is a part of the “Google Data Analytics Professional Certificate’s” capstone project offered by Coursera.

In this report, the chosen case study is the first problem in track#1, also known as ‘**Cyclistic**,’ an imaginary bike-sharing company in Chicago. The company offers both traditional and assistive bikes to both casual riders and riders who are annual members.

The quoted scenario provided by the course:

“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.”

1. “Ask”

The questions that the stakeholders want answer in this scenario are:

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?

From these questions, the priority-1 task in this project is to identify the differences of bike usage between annual members and casual riders. Then, use those differences to figure out how to keep the existing and attract new annual members to maximize Cyclistic’s profit.

2. “Prepare”

Note: All the data preparation processes and tools involved were all under **Windows 10 (Home)** operating system.

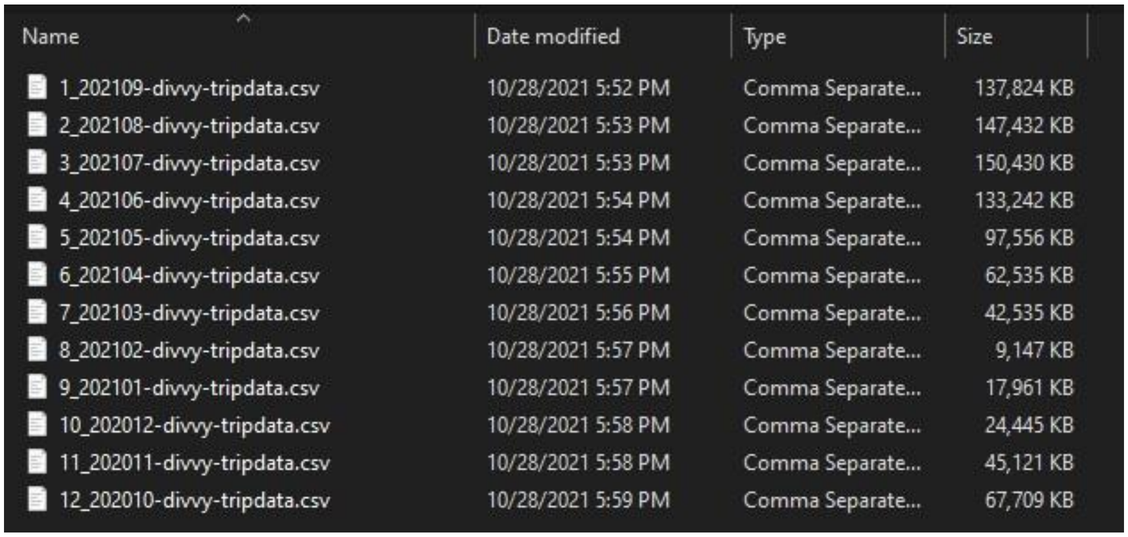
The **data** used in this scenario was provided by the course under this **license**. There were 12 data files for the data collected in the last 12 months. Each file was originally compressed in a .zip file format. The .zip files were named according to the month in the year the data was collected. For example, the file containing the data from **September 2021** was named “**202109-divvy-tripdata.zip**.”

From the data provided, the latest data was from **September 2021**, and the oldest **October 2020**. Therefore, the first and the last downloaded files were "**202109-divvy-tripdata.zip**" and "**202010-divvy-tripdata.zip**", respectively. The following is the complete list of downloaded files in descending chronological order:

202109-divvy-tripdata.zip
202108-divvy-tripdata.zip
202107-divvy-tripdata.zip
202106-divvy-tripdata.zip
202105-divvy-tripdata.zip
202104-divvy-tripdata.zip
202103-divvy-tripdata.zip
202102-divvy-tripdata.zip
202101-divvy-tripdata.zip
202012-divvy-tripdata.zip
202011-divvy-tripdata.zip
202010-divvy-tripdata.zip

Once unzipped, the data files were in **comma-separated values (.csv)** format. The files were **renamed** with **number prefixes starting from "1_" to "12_"**, from the first downloaded file containing the most up-to-date data to the last. The added prefixes helped make it easier to view the file in chronological order in a folder under Windows 10 operating system.

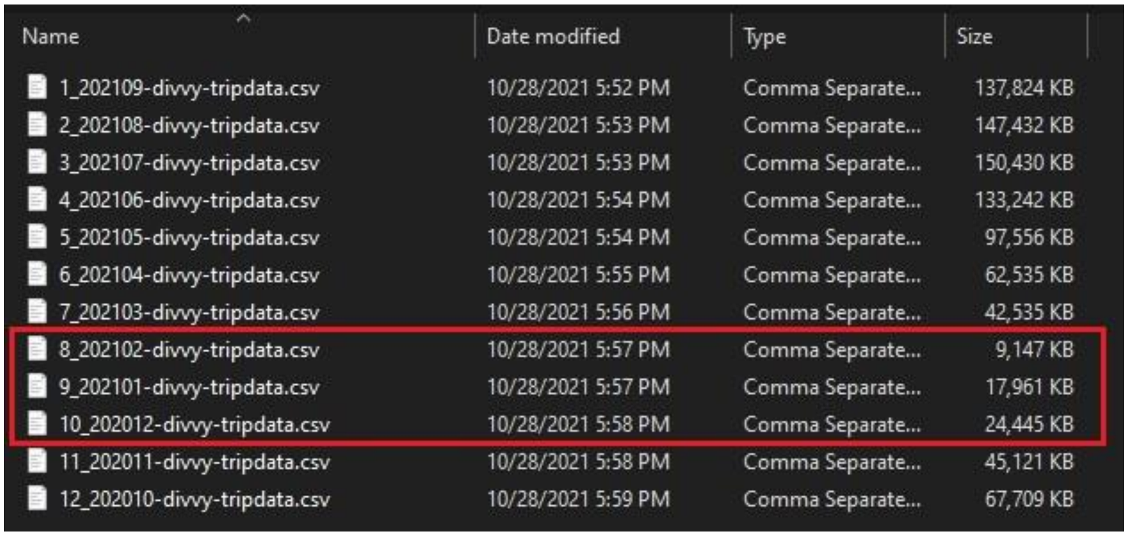
The following picture shows the unzipped and renamed files automatically sorted by the given prefixes in a folder under Windows 10 operating system:



Name	Date modified	Type	Size
1_202109-divvy-tripdata.csv	10/28/2021 5:52 PM	Comma Separate...	137,824 KB
2_202108-divvy-tripdata.csv	10/28/2021 5:53 PM	Comma Separate...	147,432 KB
3_202107-divvy-tripdata.csv	10/28/2021 5:53 PM	Comma Separate...	150,430 KB
4_202106-divvy-tripdata.csv	10/28/2021 5:54 PM	Comma Separate...	133,242 KB
5_202105-divvy-tripdata.csv	10/28/2021 5:54 PM	Comma Separate...	97,556 KB
6_202104-divvy-tripdata.csv	10/28/2021 5:55 PM	Comma Separate...	62,535 KB
7_202103-divvy-tripdata.csv	10/28/2021 5:56 PM	Comma Separate...	42,535 KB
8_202102-divvy-tripdata.csv	10/28/2021 5:57 PM	Comma Separate...	9,147 KB
9_202101-divvy-tripdata.csv	10/28/2021 5:57 PM	Comma Separate...	17,961 KB
10_202012-divvy-tripdata.csv	10/28/2021 5:58 PM	Comma Separate...	24,445 KB
11_202011-divvy-tripdata.csv	10/28/2021 5:58 PM	Comma Separate...	45,121 KB
12_202010-divvy-tripdata.csv	10/28/2021 5:59 PM	Comma Separate...	67,709 KB

Figure 2.1: The list of unzipped files seen in a folder on a Windows 10 machine

Some files are relatively smaller than the rest. This can be an evidence of significantly smaller number of observations in those files:



Name	Date modified	Type	Size
1_202109-divvy-tripdata.csv	10/28/2021 5:52 PM	Comma Separate...	137,824 KB
2_202108-divvy-tripdata.csv	10/28/2021 5:53 PM	Comma Separate...	147,432 KB
3_202107-divvy-tripdata.csv	10/28/2021 5:53 PM	Comma Separate...	150,430 KB
4_202106-divvy-tripdata.csv	10/28/2021 5:54 PM	Comma Separate...	133,242 KB
5_202105-divvy-tripdata.csv	10/28/2021 5:54 PM	Comma Separate...	97,556 KB
6_202104-divvy-tripdata.csv	10/28/2021 5:55 PM	Comma Separate...	62,535 KB
7_202103-divvy-tripdata.csv	10/28/2021 5:56 PM	Comma Separate...	42,535 KB
8_202102-divvy-tripdata.csv	10/28/2021 5:57 PM	Comma Separate...	9,147 KB
9_202101-divvy-tripdata.csv	10/28/2021 5:57 PM	Comma Separate...	17,961 KB
10_202012-divvy-tripdata.csv	10/28/2021 5:58 PM	Comma Separate...	24,445 KB
11_202011-divvy-tripdata.csv	10/28/2021 5:58 PM	Comma Separate...	45,121 KB
12_202010-divvy-tripdata.csv	10/28/2021 5:59 PM	Comma Separate...	67,709 KB

Figture 2.2: Files that are relatively small compared to the rest

At this point, the data was ready to be processed. It is worth noting that this data was, per the scenario, provided directly by Cyclistic. This data is **secondary data**. This means that the analyst didn't collect this data by himself, the data was provided to the analyst by a party/organization that collected the data and directly involve in the analysis process as stakeholders.

3. "Process"

The tools used in this project are **RStudio** (2021.09.0 Build 351) runs on **R** (version 4.1.1) and **Microsoft Excel** (Office 365 version), all, as stated above, run under Windows 10 (Home) operating system.

R was chosen because it can manage large data sets much quicker than spreadsheets. It can also visualize and create well-formatted reports, complete with code chunks and pictures. Excel, in this case, is for creating a temporary summary tables and visualizations which were later ported into RStudio to create this report.

3.1 Importing data into RStudio

Before importing any data into RStudio, the working directory must be changed to match the directory in which the data is stored. The quickest way to do this is to use Windows' GUI to navigate to the folder containing those files, then copy path string from the address bar, for example, "**C:\Users\...\Cyclistic_data**", then paste the copied path string in the function '**setwd()**' but change all the **backslash** (\) into **slash** (/) to comply with R syntax:

```
#Code chunk 3.1.1

setwd("C:/Users/.../Cyclistic_data--Example file location")
```

Next, use the "**read_csv()**" function to import data from each .csv file and store in R's data objects as data frames. In this case, each data frame was given its corresponding name according to the data stored in each file. For example, the data collected from **September 2021** will be stored in a data frame named "**SEP21**."

The following code chunk shows the data import step:

```
#Code chunk 3.1.2

SEP21 <- read_csv("1_202109-divvy-tripdata.csv")
```

The data in the rest of the downloaded files was imported with the above code and stored in data frames named **SEP21, AUG21,…,OCT20**, totaling to 12 objects in R's environment.

3.2 Data Integrity Checkups

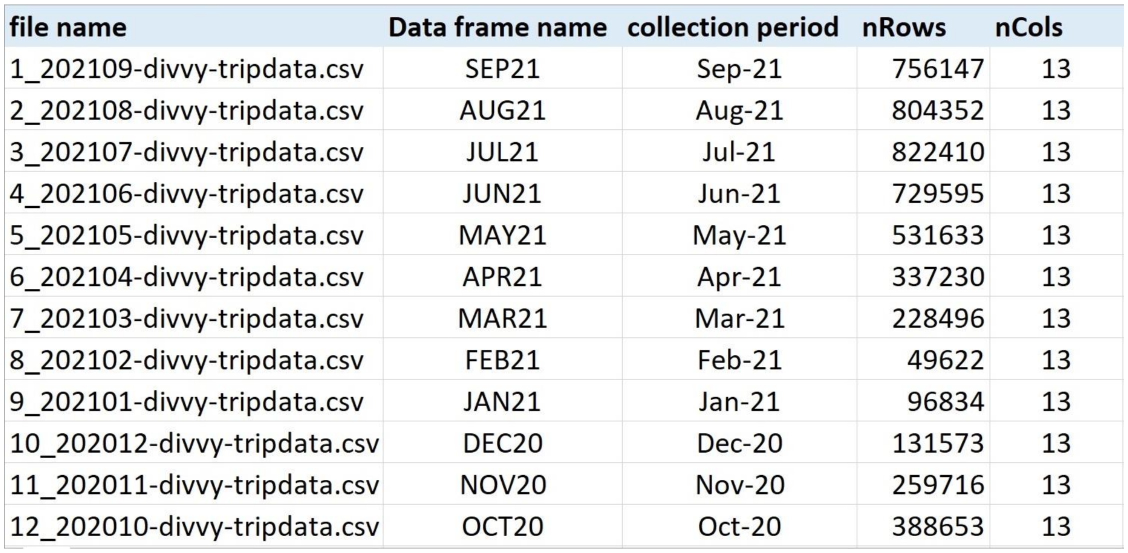
3.2.1 Glimpses of the Data Integrity

Next, the data frames were subjected to the integrity checkup process. First, the "**glimpse()**" function was used to get an overview of each data object, including the number of rows and columns, and data type of each column.

```
#Code chunk 3.2.1.1

> glimpse(SEP21)
Rows: 756,147
Columns: 13
$ ride_id            <chr> "9DC7B962304CBFD8", "F930E2C6872~
$ rideable_type      <chr> "electric_bike", "electric_bike"~
$ started_at         <dttm> 2021-09-28 16:07:10, 2021-09-28~
$ ended_at           <dttm> 2021-09-28 16:09:54, 2021-09-28~
$ start_station_name <chr> NA, NA, NA, NA, NA, NA, NA, NA, ~
$ start_station_id   <chr> NA, NA, NA, NA, NA, NA, NA, NA, ~
$ end_station_name   <chr> NA, NA, NA, NA, NA, NA, NA, NA, ~
$ end_station_id     <chr> NA, NA, NA, NA, NA, NA, NA, NA, ~
$ start_lat          <dbl> 41.89000, 41.94000, 41.81000, 41~
$ start_lng          <dbl> -87.68000, -87.64000, -87.72000,~
$ end_lat            <dbl> 41.89, 41.98, 41.80, 41.81, 41.8~
$ end_lng            <dbl> -87.67, -87.67, -87.72, -87.72, ~
$ member_casual      <chr> "casual", "casual", "casual", "c~
```

From the above result, each of the **13 variables** in **SEP21** seems to have been assigned its correct data type. Every data frame was checked with the **glimpse()** function and the number of rows and columns and other exploratory information were recorded in a temporary summary table, in this case, in **Excel**:



file name	Data frame name	collection period	nRows	nCols
1_202109-divvy-tripdata.csv	SEP21	Sep-21	756147	13
2_202108-divvy-tripdata.csv	AUG21	Aug-21	804352	13
3_202107-divvy-tripdata.csv	JUL21	Jul-21	822410	13
4_202106-divvy-tripdata.csv	JUN21	Jun-21	729595	13
5_202105-divvy-tripdata.csv	MAY21	May-21	531633	13
6_202104-divvy-tripdata.csv	APR21	Apr-21	337230	13
7_202103-divvy-tripdata.csv	MAR21	Mar-21	228496	13
8_202102-divvy-tripdata.csv	FEB21	Feb-21	49622	13
9_202101-divvy-tripdata.csv	JAN21	Jan-21	96834	13
10_202012-divvy-tripdata.csv	DEC20	Dec-20	131573	13
11_202011-divvy-tripdata.csv	NOV20	Nov-20	259716	13
12_202010-divvy-tripdata.csv	OCT20	Oct-20	388653	13

Figure 3.2.1.1: A temporary table summarizing the exploratory results of the data sets in an Excel table

From the above table, each data frame contains varying number of rows, but the most important point is that they all have the same number of columns, presumably with the same data type in each column.

3.2.2 Inspecting and comparing data type in each column of every data frame

To make sure that the data type in each of the column in every data frame is consistent, a data type check up was performed using the "**sapply()**" function to apply "**typeof()**" across every column in a data frame. The information about the data type of each column in a data frame can be extracted and stored in a separate object named according to its original data frame. For example, a data type object extracted from the data frame **SEP21** was named **dtSEP21**.

Each data type object extracted from each data frame was then compared by using the "**identical()**" function. All 12 data type objects were compared following the **transitive property** that states that "**if a = b and b = c, then a = c**." The following is the code chunk demonstrating the comparison process and its results:

```
#Code chunk 3.2.2.1

> dtSEP21 <- sapply(SEP21,typeof)
> dtAUG21 <- sapply(AUG21,typeof)
> dtJUL21 <- sapply(JUL21,typeof)
> dtJUN21 <- sapply(JUN21,typeof)
> dtMAY21 <- sapply(MAY21,typeof)
> dtAPR21 <- sapply(APR21,typeof)
> dtMAR21 <- sapply(MAR21,typeof)
> dtFEB21 <- sapply(FEB21,typeof)
> dtJAN21 <- sapply(JAN21,typeof)
> dtDEC20 <- sapply(DEC20,typeof)
> dtNOV20 <- sapply(NOV20,typeof)
> dtOCT20 <- sapply(OCT20,typeof)

> identical(dtSEP21,dtAUG21)
[1] TRUE
> identical(dtAUG21,dtJUL21)
[1] TRUE
> identical(dtJUL21,dtJUN21)
[1] TRUE
> identical(dtJUN21,dtMAY21)
[1] TRUE
> identical(dtMAY21,dtAPR21)
[1] TRUE
> identical(dtAPR21,dtMAR21)
[1] TRUE
> identical(dtMAR21,dtFEB21)
[1] TRUE
> identical(dtFEB21,dtJAN21)
[1] TRUE
> identical(dtJAN21,dtDEC20)
[1] TRUE
> identical(dtDEC20,dtNOV20)
[1] FALSE
> identical(dtNOV20,dtOCT20)
[1] TRUE
```

From the result of the code chunk above, the data type integrity was broken in the data frames **NOV20** and **OCT20**, the last two data frames. Further inspections with **glimpse()** revealed that the **start_station_id** and **end_station_id** were assigned **double** type, instead of the **character** type. This was fixed by re-assigning the correct data types to those columns in both data frames with the following code chunks:

```
#Code chunk 3.2.2.2 - check the data types of NOV20 and OCT20 data frames

> glimpse(NOV20)
Rows: 259,716
Columns: 13
$ ride_id            <chr> "BD0A6FF6FFF9B921", "96A7A7A4BDE4~
$ rideable_type      <chr> "electric_bike", "electric_bike",~
$ started_at         <dttm> 2020-11-01 13:36:00, 2020-11-01 ~
$ ended_at           <dttm> 2020-11-01 13:45:40, 2020-11-01 ~
$ start_station_name <chr> "Dearborn St & Erie St", "Frankli~
$ start_station_id   <dbl> 110, 672, 76, 659, 2, 72, 76, NA,~
$ end_station_name   <chr> "St. Clair St & Erie St", "Noble ~
$ end_station_id     <dbl> 211, 29, 41, 185, 2, 76, 72, NA, ~
$ start_lat          <dbl> 41.89418, 41.89096, 41.88098, 41.~
$ start_lng          <dbl> -87.62913, -87.63534, -87.61675, ~
$ end_lat            <dbl> 41.89443, 41.90067, 41.87205, 41.~
$ end_lng            <dbl> -87.62338, -87.66248, -87.62955, ~
$ member_casual      <chr> "casual", "casual", "casual", "ca~

> glimpse(OCT20)
Rows: 388,653
Columns: 13
$ ride_id            <chr> "ACB6B40CF5B9044C", "DF450C72FD10~
$ rideable_type      <chr> "electric_bike", "electric_bike",~
$ started_at         <dttm> 2020-10-31 19:39:43, 2020-10-31 ~
$ ended_at           <dttm> 2020-10-31 19:57:12, 2020-11-01 ~
$ start_station_name <chr> "Lakeview Ave & Fullerton Pkwy", ~
$ start_station_id   <dbl> 313, 227, 102, 165, 190, 359, 313~
$ end_station_name   <chr> "Rush St & Hubbard St", "Kedzie A~
$ end_station_id     <dbl> 125, 260, 423, 256, 185, 53, 125,~
$ start_lat          <dbl> 41.92610, 41.94817, 41.77346, 41.~
$ start_lng          <dbl> -87.63898, -87.66391, -87.58537, ~
$ end_lat            <dbl> 41.89035, 41.92953, 41.79145, 41.~
$ end_lng            <dbl> -87.62607, -87.70782, -87.60005, ~
$ member_casual      <chr> "casual", "casual", "casual", "ca~

#Code chunk 3.2.2.3 - Assigning the correct data types

> NOV20 <- transform(NOV20, start_station_id = as.character(start_station_id)
)
> NOV20 <- transform(NOV20, end_station_id = as.character(end_station_id))
> OCT20 <- transform(OCT20, start_station_id = as.character(start_station_id)
)
> OCT20 <- transform(OCT20, end_station_id = as.character(end_station_id))
```

```
#Code chunk 3.2.2.4 - Confirming the data type transformation

> glimpse(NOV20)
Rows: 259,716
Columns: 13
$ ride_id            <chr> "BD0A6FF6FFF9B921", "96A7A7A4BDE4~
$ rideable_type      <chr> "electric_bike", "electric_bike",~
$ started_at         <dttm> 2020-11-01 13:36:00, 2020-11-01 ~
$ ended_at           <dttm> 2020-11-01 13:45:40, 2020-11-01 ~
$ start_station_name <chr> "Dearborn St & Erie St", "Frankli~
$ start_station_id   <chr> "110", "672", "76", "659", "2", "~
$ end_station_name   <chr> "St. Clair St & Erie St", "Noble ~
$ end_station_id     <chr> "211", "29", "41", "185", "2", "7~
$ start_lat          <dbl> 41.89418, 41.89096, 41.88098, 41.~
$ start_lng          <dbl> -87.62913, -87.63534, -87.61675, ~
$ end_lat            <dbl> 41.89443, 41.90067, 41.87205, 41.~
$ end_lng            <dbl> -87.62338, -87.66248, -87.62955, ~
$ member_casual      <chr> "casual", "casual", "casual", "ca~


> glimpse(OCT20)
Rows: 388,653
Columns: 13
$ ride_id            <chr> "ACB6B40CF5B9044C", "DF450C72FD10~
$ rideable_type      <chr> "electric_bike", "electric_bike",~
$ started_at         <dttm> 2020-10-31 19:39:43, 2020-10-31 ~
$ ended_at           <dttm> 2020-10-31 19:57:12, 2020-11-01 ~
$ start_station_name <chr> "Lakeview Ave & Fullerton Pkwy", ~
$ start_station_id   <chr> "313", "227", "102", "165", "190"~
$ end_station_name   <chr> "Rush St & Hubbard St", "Kedzie A~
$ end_station_id     <chr> "125", "260", "423", "256", "185"~
$ start_lat          <dbl> 41.92610, 41.94817, 41.77346, 41.~
$ start_lng          <dbl> -87.63898, -87.66391, -87.58537, ~
$ end_lat            <dbl> 41.89035, 41.92953, 41.79145, 41.~
$ end_lng            <dbl> -87.62607, -87.70782, -87.60005, ~
$ member_casual      <chr> "casual", "casual", "casual", "ca~
```

From the above code chunk, the data type transformation from “**double**” to “**character**” was successful. Another data type checkup was done to confirm that the data type integrity was intact for all the 12 data frames. The following code chunk shows the process of data type extraction and comparison of the newly fixed two data frames, using the **DEC20** data frame as a reference starting point:

```
#Code chunk 3.2.2.5: data type extraction and comparison

> dtNOV20 <- sapply(NOV20,typeof)
> dtOCT20 <- sapply(OCT20,typeof)

> identical(dtDEC20,dtNOV20)
[1] TRUE
> identical(dtNOV20,dtOCT20)
[1] TRUE
```

It was confirmed with the above code chunk that all the data types across every columns in every data frame were exactly the same at this point, according to the transitive property.

3.2.3 Handling NAs

As seen earlier in code chunk 3.2.1.1, there were several **NAs** (missing "not available" values) readily visible when the **glimpse()** function was applied to the **SEP21** data frame. This also happened to the other data frames as well.

In this case, rows containing NAs weren't removed because it was suspected that they were not input errors, and learning about their occurrence pattern could reveal more insights about the data.

Initially, the occurrence of NAs in different columns of each data frame was recorded with the following code, demonstrated with the "**SEP21**" data frame:

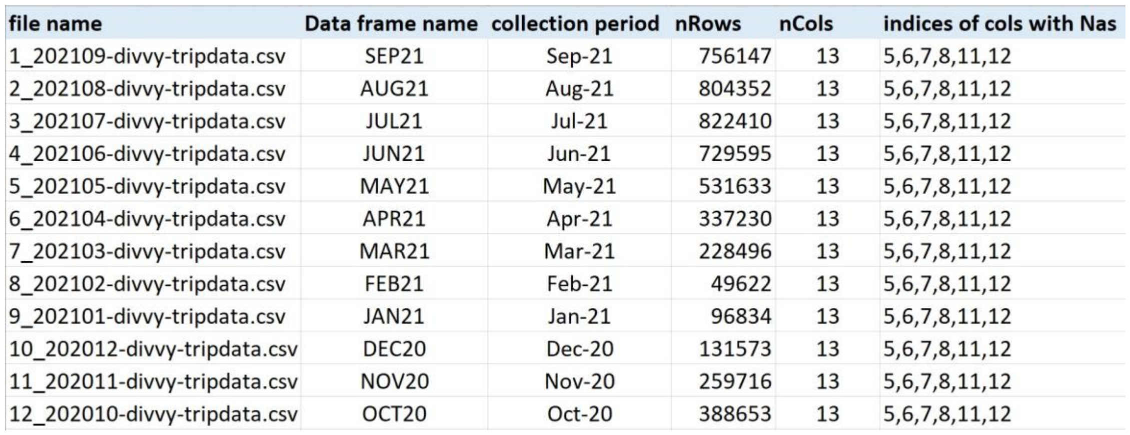
```
#Code chunk 3.2.3.1: Find the column index with NAs in them

> which(colSums(is.na(SEP21))!=0)
start_station_name   start_station_id   end_station_name 
                 5                  6                  7 
    end_station_id            end_lat            end_lng 
                 8                 11                 12 
```

From the code above, the columns of the **SEP21** data frame with NAs in them were:

column index: **5**, column name: **start_station_name**
column index: **6**, column name: **start_station_id**
column index: **7**, column name: **end_station_name**
column index: **8**, column name: **end_station_id**
column index: **11**, column name: **end_lat**
column index: **12**, column name: **end_lng**

This specific information about NAs was also obtained from the rest of the data frames and was added to the Excel temporary summary table:



file name	Data frame name	collection period	nRows	nCols	indices of cols with Nas
1_202109-divvy-tripdata.csv	SEP21	Sep-21	756147	13	5,6,7,8,11,12
2_202108-divvy-tripdata.csv	AUG21	Aug-21	804352	13	5,6,7,8,11,12
3_202107-divvy-tripdata.csv	JUL21	Jul-21	822410	13	5,6,7,8,11,12
4_202106-divvy-tripdata.csv	JUN21	Jun-21	729595	13	5,6,7,8,11,12
5_202105-divvy-tripdata.csv	MAY21	May-21	531633	13	5,6,7,8,11,12
6_202104-divvy-tripdata.csv	APR21	Apr-21	337230	13	5,6,7,8,11,12
7_202103-divvy-tripdata.csv	MAR21	Mar-21	228496	13	5,6,7,8,11,12
8_202102-divvy-tripdata.csv	FEB21	Feb-21	49622	13	5,6,7,8,11,12
9_202101-divvy-tripdata.csv	JAN21	Jan-21	96834	13	5,6,7,8,11,12
10_202012-divvy-tripdata.csv	DEC20	Dec-20	131573	13	5,6,7,8,11,12
11_202011-divvy-tripdata.csv	NOV20	Nov-20	259716	13	5,6,7,8,11,12
12_202010-divvy-tripdata.csv	OCT20	Oct-20	388653	13	5,6,7,8,11,12

Figure 3.2.3.1: A temporary table summarizing the exploratory results of the data sets in an Excel table, including column indices of each data frame containing NAs

From the above table, it could be seen that every data frame have the same columns with NAs values. These columns contains the information about the starting and ending stations of the bikes. NAs in these columns can mean that riders did not take off from stations, nor did they park the bikes at stations.

Another interesting observation is that there are also some NAs in the columns containing the actual position of the bikes after each trip. This can mean that the bikes' positioning devices ran out of battery before they reached their destinations.

These observations concerning NAs could reveal more insights about customer behaviors and maintenance requirements. Further investigations and their results will be shown later in this report.

3.2.4 Check for Duplicated Rows

Checking for duplicated entries of data should be one of the first tasks in data processing. However, the nature of this particular data allow for many duplicated values in all but the "**ride_id**" column. Each ride id represents one unique bike trip. There should be no two identical ride ids, however the opposite, where there are at least two rows with unique ride ids but the rest of the fields contain the exact same information, is possible.

In this case, the "**red flag**" appears only when duplicated ride ids are found. The following code chunk checked for duplicated ride ids, as an example, in the data frame **SEP21**:

```
#Code chunk 3.2.4.1: Check for duplicated ride ids

> SEP21 %>% group_by(ride_id) %>% filter(n()>1)
# A tibble: 0 x 13
# Groups:   ride_id [0]
# ... with 13 variables: ride_id <chr>, rideable_type <chr>,
#   started_at <dttm>, ended_at <dttm>,
#   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>
>
```

A tibble of 0 rows and 13 columns was returned from the code above. This means that there was no duplicated ride id for the **SEP21** data frame. After every data frame was checked, no one showed up with duplicated ride ids.

3.2.5 Combining All Data Frames

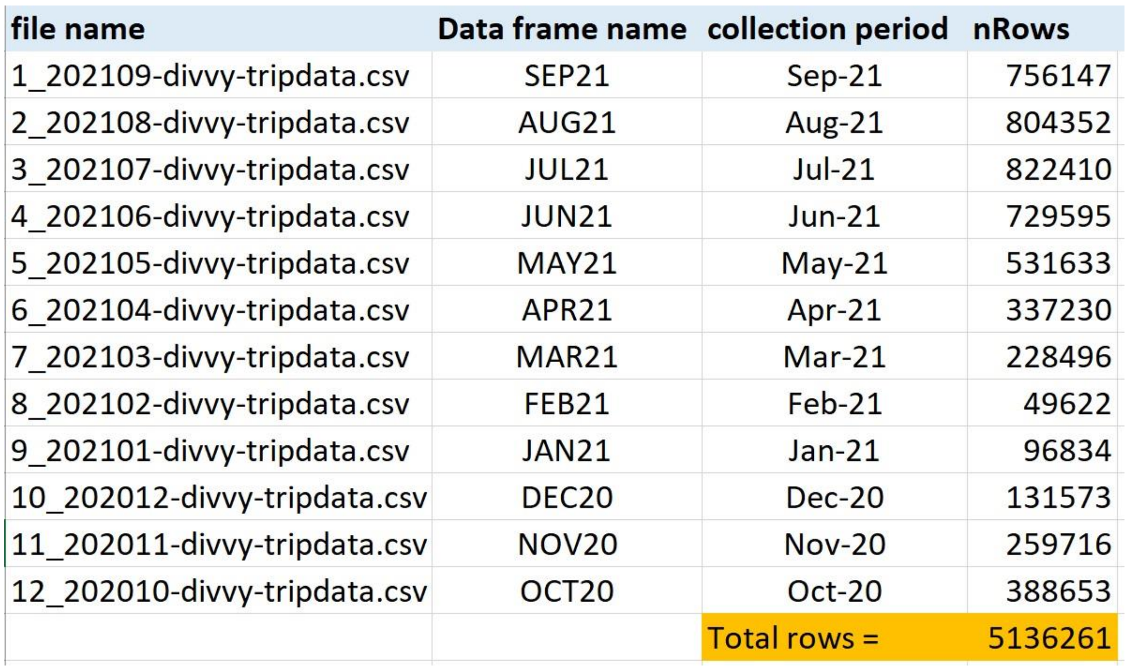
At this point, the data integrity was checked and confirmed its consistency. There was nothing to add or remove from the data frames just yet. They were ready to be combined into one big data frame with complete information ready for the analysis.

The following code combined all data frame together into a single big data frame named "**alldata**":

```
#Code chunk 3.2.5.1: Combining all data frames into one complete data frame r
eady for analysis

> alldata <- bind_rows(SEP21,AUG21,JUL21,JUN21,MAY21,APR21,MAR21,FEB21,JAN21,
DEC20,NOV20,OCT20)
>
> glimpse(alldata)
Rows: 5,136,261
Columns: 13
$ ride_id            <chr> "9DC7B962304CBFD8", "F930E2C6872D6~
$ rideable_type      <chr> "electric_bike", "electric_bike", ~
$ started_at         <dttm> 2021-09-28 16:07:10, 2021-09-28 1~
$ ended_at           <dttm> 2021-09-28 16:09:54, 2021-09-28 1~
$ start_station_name <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA~
$ start_station_id   <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA~
$ end_station_name   <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA~
$ end_station_id     <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA~
$ start_lat          <dbl> 41.89000, 41.94000, 41.81000, 41.8~
$ start_lng          <dbl> -87.68000, -87.64000, -87.72000, -~
$ end_lat            <dbl> 41.89, 41.98, 41.80, 41.81, 41.88,~
$ end_lng            <dbl> -87.67, -87.67, -87.72, -87.72, -8~
$ member_casual      <chr> "casual", "casual", "casual", "cas~
```

The result of "**glimpse(alldata)**" returned a data frame of **5,136,261** rows and **13** columns. The number 5,136,261 was equal to the sum of all rows in all of the smaller data frames.



file name	Data frame name	collection period	nRows
1_202109-divvy-tripdata.csv	SEP21	Sep-21	756147
2_202108-divvy-tripdata.csv	AUG21	Aug-21	804352
3_202107-divvy-tripdata.csv	JUL21	Jul-21	822410
4_202106-divvy-tripdata.csv	JUN21	Jun-21	729595
5_202105-divvy-tripdata.csv	MAY21	May-21	531633
6_202104-divvy-tripdata.csv	APR21	Apr-21	337230
7_202103-divvy-tripdata.csv	MAR21	Mar-21	228496
8_202102-divvy-tripdata.csv	FEB21	Feb-21	49622
9_202101-divvy-tripdata.csv	JAN21	Jan-21	96834
10_202012-divvy-tripdata.csv	DEC20	Dec-20	131573
11_202011-divvy-tripdata.csv	NOV20	Nov-20	259716
12_202010-divvy-tripdata.csv	OCT20	Oct-20	388653
		Total rows =	5136261

Figure 3.2.5.1: The total number of rows in the temporary excel table matches that calculated with the glimpse() function applied to the "alldata" data frame

The data type integrity of the **alldata** data frame was also preserved.

3.2.6 Change Data Type of Some Variables

In this case, some variable contained categorical values that could be converted into R's "**factor**" type. Doing so would reduce data redundancy and save some memory space. The following code performed data conversion to "**factor**":

```
#Code chunk 3.2.6.1: convert data type to 'factor'

> alldata <- alldata %>%
    mutate(rideable_type=as.factor(rideable_type)) %>%
    mutate(member_casual=as.factor(member_casual))
```

The result of the following code confirmed that the data in the "**rideable_type**" and the "**member_casual**" had been converted to "**factor**".

```
#Code chunk 3.2.6.2: Data type conversion transformation

> summary(alldata)
   ride_id                    rideable_type
 Length:5136261     classic_bike :2750831
 Class :character   docked_bike  : 677980
 Mode  :character   electric_bike:1707450

   started_at                         ended_at
 Min.   :2020-10-01 00:00:06   Min.   :2020-10-01 00:05:09
 1st Qu.:2021-04-11 18:50:57   1st Qu.:2021-04-11 19:15:05
 Median :2021-06-21 18:01:31   Median :2021-06-21 18:20:59
 Mean   :2021-05-25 22:30:57   Mean   :2021-05-25 22:51:34
 3rd Qu.:2021-08-11 21:13:51   3rd Qu.:2021-08-11 21:33:57
 Max.   :2021-09-30 23:59:48   Max.   :2021-10-01 22:55:35

 start_station_name start_station_id   end_station_name
 Length:5136261     Length:5136261     Length:5136261
 Class :character   Class :character   Class :character
 Mode  :character   Mode  :character   Mode  :character

 end_station_id       start_lat       start_lng
 Length:5136261     Min.   :41.64   Min.   :-87.84
 Class :character   1st Qu.:41.88   1st Qu.:-87.66
 Mode  :character   Median :41.90   Median :-87.64
                    Mean   :41.90   Mean   :-87.65
                    3rd Qu.:41.93   3rd Qu.:-87.63
                    Max.   :42.08   Max.   :-87.52

    end_lat         end_lng       member_casual
 Min.   :41.51   Min.   :-88.07   casual:2358287
 1st Qu.:41.88   1st Qu.:-87.66   member:2777974
 Median :41.90   Median :-87.64
 Mean   :41.90   Mean   :-87.65
 3rd Qu.:41.93   3rd Qu.:-87.63
 Max.   :42.17   Max.   :-87.44
 NA's   :4821    NA's   :4821
```

3.2.7 Create a Calculated Column: "ride_duration_min"

In the "**alldata**" data frame, the information about the starting and ending time for each trip was available. It would be useful to extract the information about the duration of each trip from this available information and store it in a new column.

However, before creating a column containing the ride durations, the "**impossible values**" in the existing date-time columns should be considered.

Logically, all the values in the "**started_at**" column must be less than their counterparts in the "**ended_at**" column. In this case, if any observation went in the opposite direction, a simple value swapping between the two columns would be applied.

The following code checked if there are any so-called “**impossible values**”:

```
#Code chunk 3.2.7.1: Checking for "Impossible Values"

> alldata %>% filter(started_at > ended_at)
# A tibble: 3,284 x 13
   ride_id     rideable_type started_at          ended_at
   <chr>       <fct>         <dttm>              <dttm>
 1 3A47A9240B~ classic_bike  2021-09-29 14:52:22 2021-09-29 14:52:09
 2 FA44EBC79C~ electric_bike 2021-09-28 17:43:33 2021-09-28 17:43:31
 3 D5E5F806D0~ classic_bike  2021-09-01 18:42:09 2021-09-01 18:41:40
 4 2EDC266B1A~ classic_bike  2021-09-04 09:42:26 2021-09-04 09:41:54
 5 137BEA38C2~ classic_bike  2021-09-02 18:38:19 2021-09-02 18:37:48
 6 7F7679196D~ classic_bike  2021-09-01 06:55:19 2021-09-01 06:55:11
 7 D4A976387C~ electric_bike 2021-09-04 17:20:37 2021-09-04 17:20:35
 8 3E68DE6382~ electric_bike 2021-09-29 17:44:02 2021-09-29 17:40:51
 9 88BA6AC732~ classic_bike  2021-09-29 15:26:23 2021-09-29 15:26:22
10 6ACD4446FA~ electric_bike 2021-09-29 17:32:54 2021-09-29 17:28:32
# ... with 3,274 more rows, and 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 <fct>
```

As a result, there were still more than 3,000 rows containing illogical date-time values.

The following code checked for any time value in **started_at** that was higher than the value in **ended_at** in the same row and swaped them:

```
#Code chunk 3.2.7.2: Checking if values in started_at are <
#values in ended_at (which is illogical) and swap them

for (i in 1:nrow(alldata)) {
  if(alldata[i,3] > alldata[i,4]) {
    c <- alldata[i,3]
    alldata[i,3] <- alldata[i,4]
    alldata[i,4] <- c
  }
}
```

Next, the following code confirmed that there were no more impossible values:

```
#Code chunk 3.2.7.3: Confirm that there are no more impossible values:

> alldata %>% filter(started_at > ended_at)
# A tibble: 0 x 13
# ... with 13 variables: ride_id <chr>, rideable_type <fct>,
#   started_at <dttm>, ended_at <dttm>, 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 <fct>
```

From the result, there were no more impossible values. It would be OK to proceed to create the calculated column "**ride_duration_min**" for trip duration at this point.

The following code calculateed each trip's duration in minutes and populates the calculation result in the new column, "**ride_duration_min**":

```
#Code chunk 3.2.7.4: Create a calculated column to store ride duration in min
utes

> alldata <- alldata %>%
  mutate(ride_duration_min=abs(difftime(alldata$ended_at,
  alldata$started_at,units="mins")))

> glimpse(alldata)
Rows: 5,136,261
Columns: 14
$ ride_id            <chr> "9DC7B962304CBFD8", "F930E2C6872D6B32",~
$ rideable_type      <fct> electric_bike, electric_bike, electric_~
$ started_at         <dttm> 2021-09-28 16:07:10, 2021-09-28 14:24:~
$ ended_at           <dttm> 2021-09-28 16:09:54, 2021-09-28 14:40:~
$ start_station_name <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
$ start_station_id   <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
$ end_station_name   <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
$ end_station_id     <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA,~
$ start_lat          <dbl> 41.89000, 41.94000, 41.81000, 41.80000,~
$ start_lng          <dbl> -87.68000, -87.64000, -87.72000, -87.72~
$ end_lat            <dbl> 41.89, 41.98, 41.80, 41.81, 41.88, 41.8~
$ end_lng            <dbl> -87.67, -87.67, -87.72, -87.72, -87.71,~
$ member_casual      <fct> casual, casual, casual, casual, casual,~
$ ride_duration_min  <drtn> 2.7333333 mins, 15.2333333 mins, 3.683~
```

The calculation resulted in an additional column at the last position called "**ride_duration_min**", storing the duration of each bike trip in minutes.

At this point, the types of information available were comprehensive, with numerical, positional, categorical, and chronological. The data frame was saved as a .csv format file with the following code:

```
#Code chunk 3.2.7.5: Saving data to a .csv file
> write_csv(alldata,"allcata.csv")
```

The next step of data analysis could finally be commenced.

4. “Analyze”

In this part of the report, insights extracted from the data with various visualizations will be discussed along with codes and technique used to created them.

Note: Only relevant visualization will be shown here. The full documentation about this phase can be accessed via this link:

4.1 Overview

The data about the monthly number of rides recorded in the temporary table in Excel can be quickly visualized to demonstrate that there are more bike trips in warm months than in cold months. This trend is to be expected from the data and can serve as another layer of confirmation that the data collected was valid.

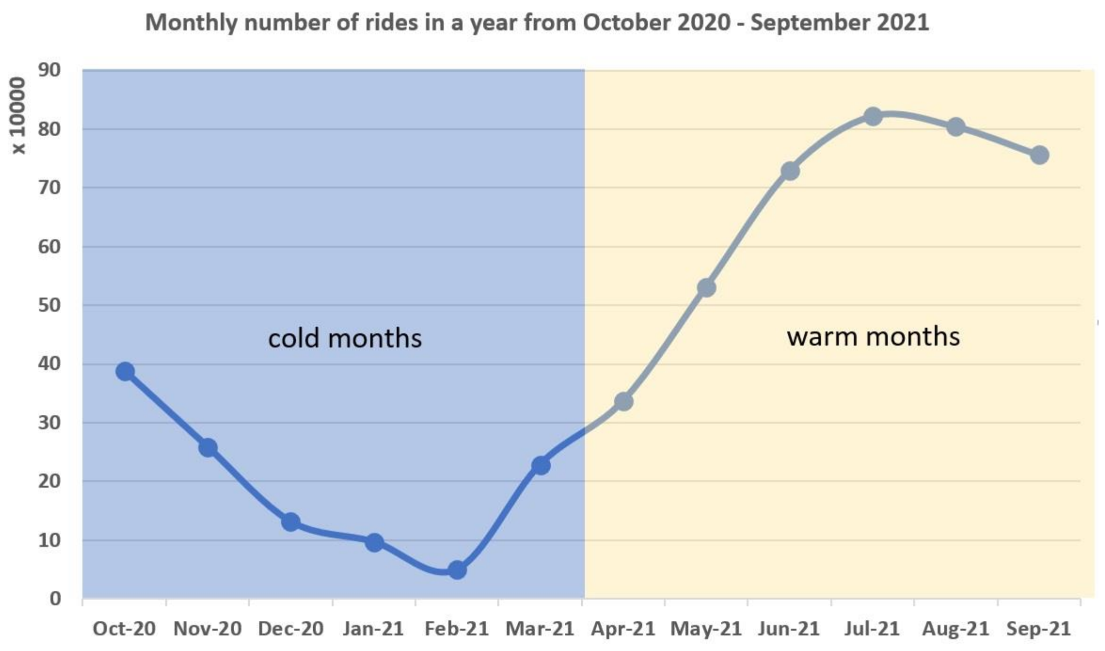


Figure 4.1.1: A quick visualization that shows number of monthly rides from October 2020 to September 2021. Notice how the number increases and decreases in different times of year.

4.2 Comparing number of rides between “casual” and “member” customers

The main questions of this data analysis project are:
1. How do annual members and casual riders use Cyclistic bikes differently?
2. Why would casual riders buy Cyclistic annual memberships?
3. How can Cyclistic use digital media to influence casual riders to become members?

The questions #1 and #2 can be changed to: "**How to convert casual customers to member customers?**" This question requires the comparison of insights extracted from both types of customers.

4.2.1 Counting Numbers of Rides from Each Type of Customer

First, the basic information about number of rides from each customer type was obtain by simply counting each ride entry from each group with the following code:

```
#Code chunk 4.2.1.1: counting the number of rides from each customer type

> table(alldata$member_casual)

 casual  member 
2358287 2777974 
```

From the entire year, there were 2,358,287 casual and 2,777,974 member riders. These figures were visualize as a pie chart by the following code:

```
#Code chunk 4.2.1.2: Number of rides from each type of customer as a pie char
t

ggplot(data=alldata)+
  geom_bar(mapping=aes(x="",fill=member_casual))+
  coord_polar("y")+
  annotate("text",label="Member",x=1,y=900000,size=6)+
  annotate("text",label="2,777,974",x=1,y=1200000,size=6)+
  annotate("text",label="54.1%",x=1,y=1500000,size=6)+
  annotate("text",label="Casual",x=1,y=4220000,size=6)+
  annotate("text",label="2,358,287",x=1,y=3950000,size=6)+
  annotate("text",label="45.9%",x=1,y=3650000,size=6)+
  theme(plot.background=element_blank(),
        panel.background=element_blank(),
        axis.title=element_blank(),
        axis.ticks=element_blank(),
        axis.text=element_blank(),
        legend.position="none")+
  ggtitle(label="Number of Rides from Each Customer Type",
          subtitle="Source: Cyclistic")
```

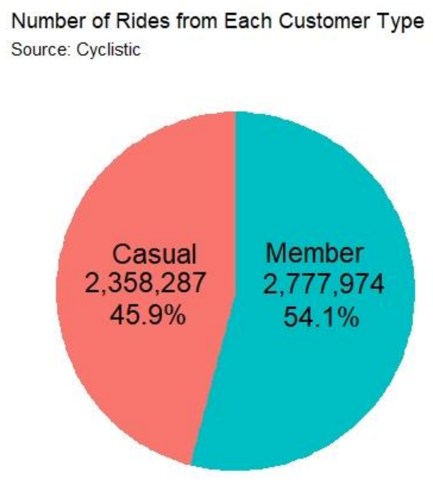


Figure 4.2.1.1: A pie chart showing numbers of rides and their percentage values from each customer type

4.2.2 Looking at the Sattistics of the Time each Customer Type Spent Riding

Next, two sets of statistics were compared to reveal the difference of the time each type of customer spent riding. A box plot was created from the data from both customer types with the following code:

```
#Code chunk 4.2.2.1: The first box plot

ggplot(data=alldata)+
  geom_boxplot(mapping=aes(x=member_casual,y=ride_duration_min))+
  ggtitle("Bike Trips Duration (mins)",
          subtitle="Source: Cyclistic")+
  scale_x_discrete(name=element_blank(),
                   labels=c("Casual Riders","Member Riders"))+
  ylab("Ride Duration (mins)")
```

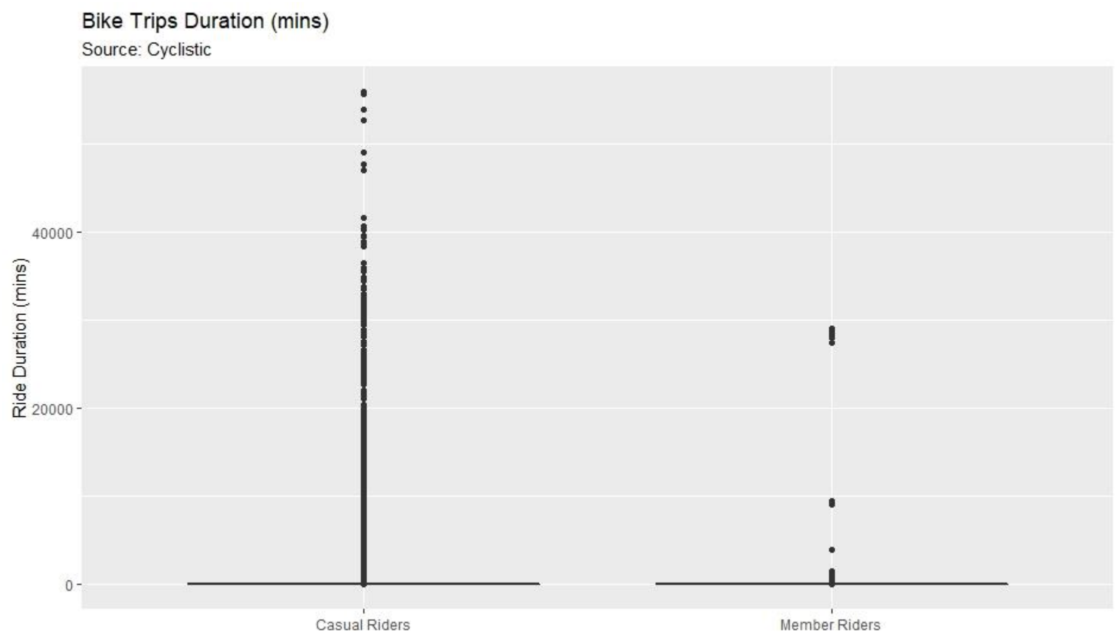


Figure 4.2.2.1: The first box plot that doesn't give any statistical view. There are simply too many outliers.

Not much information could be gained from this plot, save for the obvious that there were too many outliers with values much higher than the means from the two boxes that they were crushed flat on the x-axis.

A procedure to spot and remove outliers was implemented using the interquartile range. This technique finds the interquartile range of a list of numbers, the width of the range, and defines boundary outside of which lie removable outliers.

The following formula define the said range outside of which outliers can be removed:

Formula 4.2.2.1: Setting the inter quantile boundary to remove outliers

$$[Q1 - 1.5IQR, Q3 + 1.5IQR]$$

Where:
Q1 is the 1st quartile value
Q3 is the 3rd quartile value
IQR is th width of the interquantile range

Before removing the outliers, some statistics of the ride duration was checked with the following code:

```
#Code chunk 4.2.2.2: Check the stats of the column ride_duration_mins
> summary(alldata$ride_duration_min)
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max. 
    0.00     7.10    12.60    24.83    22.82 55944.15
```

The statistics summary of the ride duration showed that the shortest trip time was zero, and the longest trip time was 55,944.15 minutes, or 15.5 hours.

In this case, the zeros on the lower end were not sensible. There should be no ride duration time of "**0 minute**". The 0 duration were most probatly errors from data entry. However, the ride duration of **15.5 hours**, albeit very long, was possible due to the genuine ride or due to the customer forgetting to return the bike.

The data of both customer groups was spitted into two new sets called "**casualdata**," and "**memberdata**" according to the following code:

```
#Code chunk 4.2.2.3: Splitting data into "casualdata" and "memberdata" data s
ets:
casualdata <- alldata %>%
  filter(member_casual=="casual")

memberdata <- alldata %>%
  filter(member_casual=="member")
```

The zeros were removed from both data sets by the following code:

```
#Code chunk 4.2.2.4: Removing zeros from both data sets

casualdata <- casualdata %>%
  filter(!(ride_duration_min==0))

memberdata <- memberdata %>%
  filter(!(ride_duration_min==0))
```

Outliers were spotted and removed according to its interquantile range (IQR), and a new data set was created to store the result by the following set of codes:

```
#Code chunk 4.2.2.5 Spotting and removing outliers from the data
#install statistical packages to deal with outliers
install.packages("ggstatplot")

#find the quantile range for "casualdata":
Q_casual <- quantile(casualdata$ride_duration_min,probs=c(.25,.75),na.rm=F)
#find the IQR of "casualdata"
iqr_casual <- IQR(casualdata$ride_duration_min)
#find the upper cut-off value:
casual_up <- Q_casual[2]+1.5*iqr_casual
```

```r
#find the lower cut-off value:
casual_low <- Q_casual[1]-1.5*iqr_casual
#removing the outliers from "casualdata":
casualdata_adj <- casualdata %>%
  mutate(ride_duration_min=ifelse(ride_duration_min>casual_up,NA,
                                  ride_duration_min)) %>%
  mutate(ride_duration_min=ifelse(ride_duration_min<casual_low,NA,
                                  ride_duration_min))

#find the quantile range for "memberdata":
Q_member <- quantile(memberdata$ride_duration_min,probs=c(.25,.75),na.rm=F)
#find the IQR of "memberdata"
iqr_member <- IQR(memberdata$ride_duration_min)
#find the upper cut-off value:
member_up <- Q_member[2]+1.5*iqr_member
#find the lower cut-off value:
member_low <- Q_member[1]-1.5*iqr_member
#removing the outliers from "memberdata":
memberdata_adj <- memberdata %>%
  mutate(ride_duration_min=ifelse(ride_duration_min>member_up,NA,
                                  ride_duration_min)) %>%
  mutate(ride_duration_min=ifelse(ride_duration_min<member_low,NA,
                                  ride_duration_min))

#merging the outlier-free data sets:
alldata_adj <- bind_rows(casualdata_adj,memberdata_adj)

#Sorting the data by time back to its original order
#This will affect the map plot
alldata_adj <- alldata_adj[rev(order(alldata_adj$started_at)),
                           decreasing=TRUE,na.last=FALSE]

#creating box plot of both data sets together:
ggplot(data=alldata_adj)+
  geom_boxplot(mapping=aes(x=member_casual,y=ride_duration_min))+
  ggtitle("Bike Trips Duration (mins)--adjusted",
          subtitle="Source: Cyclistic")+
  scale_x_discrete(name=element_blank(),labels=c("Casual Riders",
                                                 "Member Riders"))+
  ylab("Ride Duration (mins)")+
  annotate("text",label="0.02",x=1.1,y=1.5)+
  annotate("text",label="8.90",x=1.1,y=10.9)+
  annotate("text",label="15.12",x=1.1,y=17)+
  annotate("text",label="25.53",x=1.1,y=27.5)+
  annotate("text",label="50.47",x=1.1,y=52)+
  annotate("segment",x=1.05,xend=1,y=0,yend=0)+
  annotate("segment",x=1.05,xend=1,y=50.47,yend=50.47)+
  annotate("text",label="0.02",x=2.1,y=1.5)+
  annotate("text",label="5.67",x=2.1,y=7.5)+
  annotate("text",label="9.55",x=2.1,y=11.5)+
  annotate("text",label="15.75",x=2.1,y=17.5)+
  annotate("text",label="30.87",x=2.1,y=32)+
  annotate("segment",x=2.05,xend=2,y=0,yend=0)+
  annotate("segment",x=2.05,xend=2,y=30.87,yend=30.87)
```

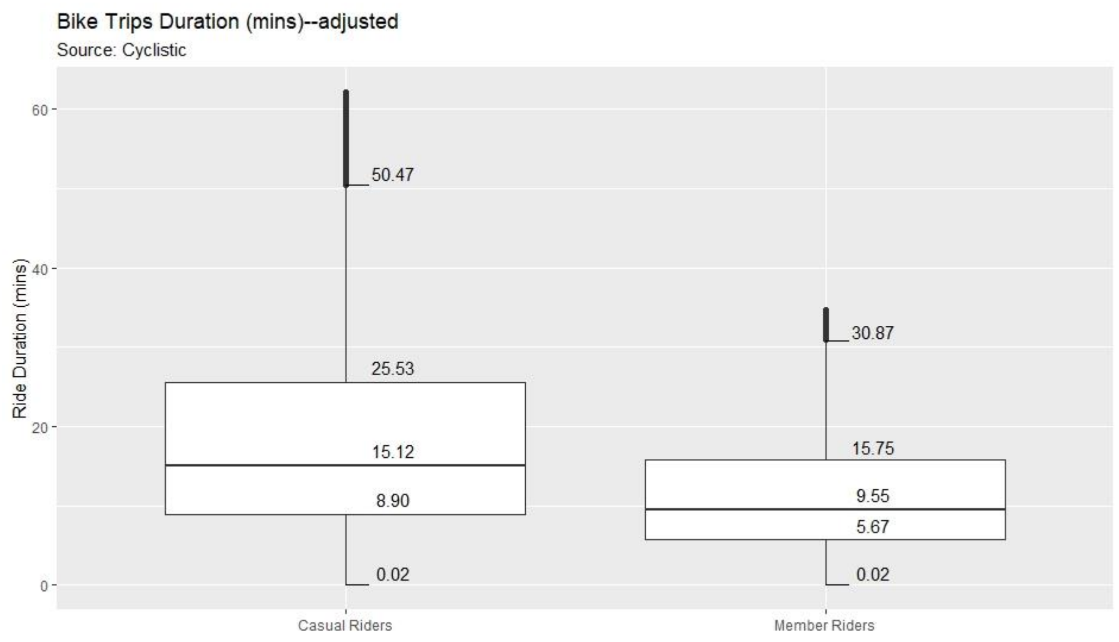


Figure 4.2.2.2: A new box plot with statistics of ride duration from both customer groups

Data records containing zeros duration were removed from the original data set. The zero-free data was named "**alldata_adj**. From this point onward, any new modification of the data set would be based on the **alldata_adj** data set.

The new box plot showed that the **casual** riders spent more time on the bikes than the **member** riders in average. This probably due to the fact that they weren't have any time constrain or have any fixed destinations in their commutes.

The outliers were still present after the first removal, but they weren't interfere with the interpretation of the result anymore.

Still, the **minimum duration** was shown to be **0.02 minutes**. This is no more possible than the 0 minute duration. However, the strange observations here might due to the fact that the data was first obtained from an imaginary source.

4.2.3 The most popular bike type

There are three types of bikes in the data set provided by Cyclistic, according to the following code:

```
#Code chunk 4.2.3.1: Showing 3 types of bikes

> unique(alldata_adj$rideable_type)
[1] "electric_bike" "docked_bike"   "classic_bike"
```

From the data provided, Cyclistic offers 3 types of bike: **electric**, **docked**, and **classic**. It's clear that the electric and the classic bikes are different, but it's not clear what exactly is the **docked** type. Again, this is an imaginary data set and observations don't have to match the reality.

Learning about the popularity of these bikes among the customer groups can help with future marketing campaigns.

The following code create a grouped bar chart visualization for the number of rides for each type of bikes from different customer groups:

```
#Code chunk 4.2.3.2: The popularity of different bike types among two custome
r groups

ggplot(data=alldata_adj)+
  geom_bar(mapping=aes(x=rideable_type,fill=member_casual),
           position="dodge")+
  scale_x_discrete(name=element_blank(),labels=c("Classic Bike","Docked Bike"
,
                                                   "Electric Bike"))+
  scale_y_continuous(name=expression(Number ~ of ~ Rides ~ (x10^5)),
                     labels=function(x) x / 100000,
                     limits=c(0,1700000))+
  labs(fill="Customer Type")+
  annotate("text",label="1,120,625",x=0.75,y=1180000)+
  annotate("text",label="1,630,033",x=1.2,y=1700000)+
  annotate("text",label="407,763",x=1.75,y=480000)+
  annotate("text",label="270,169",x=2.25,y=350000)+
  annotate("text",label="829,649",x=2.75,y=905000)+
  annotate("text",label="877,563",x=3.25,y=957563)+
  ggtitle("Number of Rides of Different Bike Types",
          subtitle="Source: Cyclistic")
```

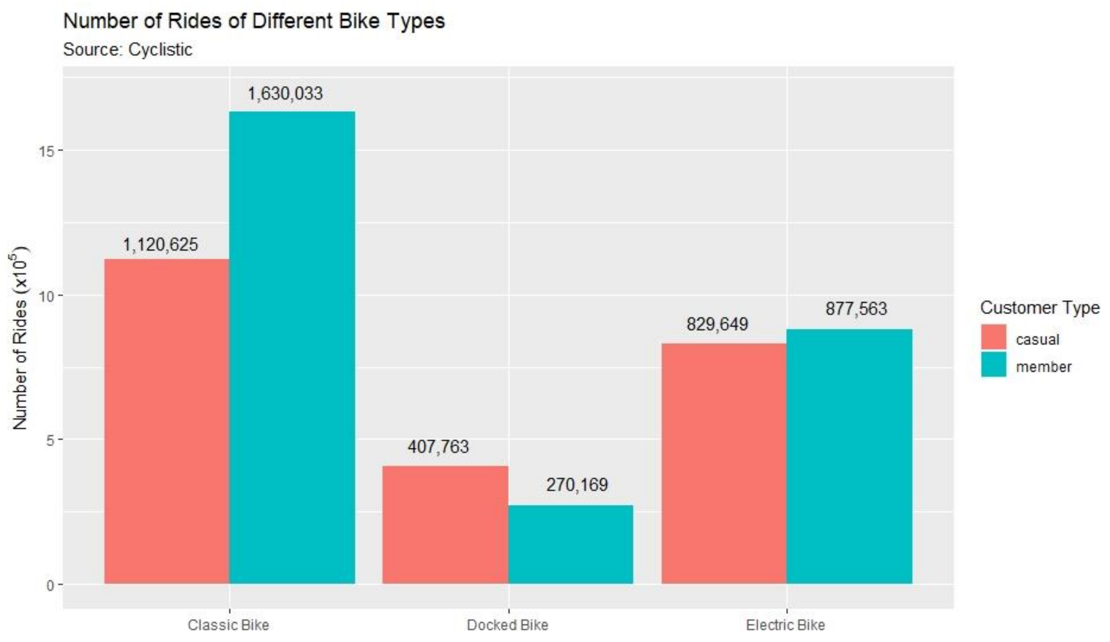


Figure 4.2.3.1: Number of Rides of Different Bike Types from both Customer Groups

From the bar chart above, it appeared that the "**classic bike**" was the most popular bike type for both group of customers. The least popular was the **docked** type which could mean the bikes that were parked "docked" at designated stations.

4.2.4 Study the starting and ending locations of different customer groups

Another aspect of customer behavior that can be studied to spot differences is the difference in locations that they started and ended their bike trips. Luckily, Cyclistic also supplied geolocation data for the starting point and ending point of each bike trip.

Map plots are required to visualize this difference. **NOTE**: Plotting maps requires installing the "**ggmap**" package and a **Google Maps API key**.
The following code plot map charts showing the starting and ending locations of each customer groups:

```
#Code chunk 4.2.4.1: Creating map charts showing the starting and the ending
points of each bike trip in each customer group

#Installing the ggmap package
install.packages("ggmap")

#Loading the ggmap package
library(ggmap)

#Registering Google Map API key
register_google(key="...xxx...xxx...xxx...")
```

```
#(The key can be obtained for free with limited usage from
#Google Cloud service)

#Getting coordinates information of "Chicago, Illinois"
chg <-geocode("Chicago, IL")

#Plotting map chart for starting locations

ggmap(get_map(chg,maptype="toner-lite"))+
  theme(plot.background=element_blank(),
        panel.background=element_blank(),
        axis.title=element_blank(),
        axis.ticks=element_blank(),
        axis.text=element_blank())+
  geom_point(data=alldata_adj,
             mapping=aes(x=start_lng,y=start_lat,
                         color=member_casual),alpha=0.6)+
  ggtitle("Bike Trip Starting Locations",
          subtitle="Source: Cyclistic")+
  scale_color_discrete(name="Customer Type")+
  theme(legend.key=element_blank())

#Plotting map chart for ending locations

ggmap(get_map(chg,maptype="toner-lite"))+
  theme(plot.background=element_blank(),
        panel.background=element_blank(),
        axis.title=element_blank(),
        axis.ticks=element_blank(),
        axis.text=element_blank())+
  geom_point(data=alldata_adj,
             mapping=aes(x=end_lng,y=end_lat,
                         color=member_casual),alpha=0.6)+
  ggtitle("Bike Trip Ending Locations",
          subtitle="Source: Cyclistic")+
  scale_color_discrete(name="Customer Type")+
  theme(legend.key=element_blank())
```

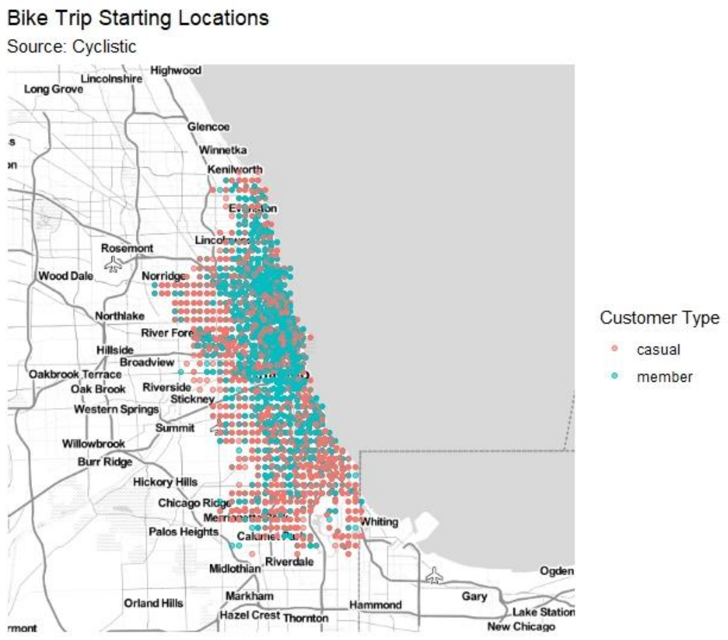


Figure 4.2.4.1: Starting locations in Chicago, IL for both customer groups

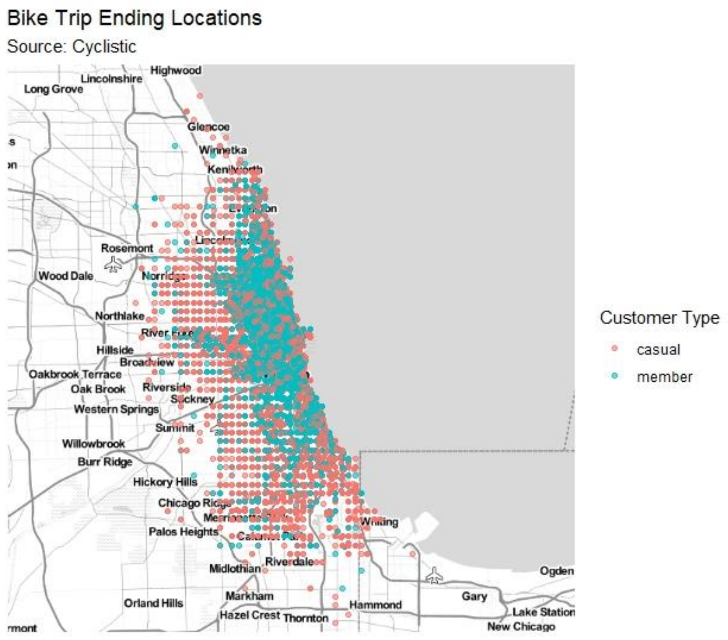


Figure 4.2.4.2: Ending locations in Chicago, IL for both customer groups

From the resulting map charts, it is obvious that the "**member**" customers have more fixed trip destinations than the "**casual**" group. The member's destinations are also more clustered in the center area of Chicago than the casual's destination.

According to the information originally provided by Cyclistic, the casual customers use the bikes as a part of their leisure activities. This information agrees with this observation and can lead to a conclusion that casual customers ride the bikes for fun and sight-seeing activities and go anywhere they please in Chicago, while the members ride the bikes regularly to and from their workplaces which have fixed points on the map.

4.2.5 The study of time: how many people ride on each day of week and each hour of day?

Another revealing insight is to learn about the relationship of time and number of rides. What day of week and what time of day do people ride most? These questions can be answered by visualizations the date-time data originally provided by Cyclistic.

First, the number of rides according to different days of week was studied. The days in the weeks of the whole data set was extracted from the column "**started_at**" for this visualization.

NOTE: The "**lubridate**" package is required to work with "datetime" type of data.

The following code created a grouped bar chart visualizing total annual number of rides according to each day of the week:

```
#Code chunk 4.2.5.1: Creating a grouped bar chart visualizing total annual nu
mber of rides on different days of the week

#Installing "lubridate" package:
install.packages("lubridate")

#Loading the package:
library(lubridate)

#Counting number of rides on each day for "casual" customers
> alldata_adj %>% group_by(weekdays(alldata_adj$started_at)) %>%
+   filter(member_casual=="casual") %>% tally()
# A tibble: 7 x 2
  `weekdays(alldata_adj$started_at)`      n
  <chr>                               <int>
1 Friday                             339374
2 Monday                             266417
3 Saturday                           523829
4 Sunday                             444997
5 Thursday                           274664
6 Tuesday                            251483
7 Wednesday                          257273
```

```
#Counting number of rides on each day for "member" customers
> alldata_adj %>% group_by(weekdays(alldata_adj$started_at)) %>%
+   filter(member_casual=="member") %>% tally()
# A tibble: 7 x 2
  `weekdays(alldata_adj$started_at)`      n
  <chr>                               <int>
1 Friday                             405902
2 Monday                             375841
3 Saturday                           400268
4 Sunday                             345273
5 Thursday                           420539
6 Tuesday                            406833
7 Wednesday                          423109

#Getting statistics from "casual" customers
> alldata_adj %>% group_by(weekdays(alldata_adj$started_at)) %>%
+   filter(member_casual=="casual") %>% tally() %>% summary()
 weekdays(alldata_adj$started_at)       n
 Length:7                         Min.   :251483
 Class :character                 1st Qu.:261845
 Mode  :character                 Median :274664
                                  Mean   :336862
                                  3rd Qu.:392186
                                  Max.   :523829

#Getting statistics from "member" customers
> alldata_adj %>% group_by(weekdays(alldata_adj$started_at)) %>%
+   filter(member_casual=="member") %>% tally() %>% summary()
 weekdays(alldata_adj$started_at)       n
 Length:7                         Min.   :345273
 Class :character                 1st Qu.:388055
 Mode  :character                 Median :405902
                                  Mean   :396824
                                  3rd Qu.:413686
                                  Max.   :423109

#Plotting the chart
ggplot(data=alldata_adj)+
  geom_bar(mapping=aes(x=weekdays(alldata_adj$started_at),
                                  fill=member_casual),
           position="dodge")+
  scale_x_discrete(name=element_blank())+
  scale_y_continuous(name=expression(Number ~ of ~ Rides ~ (x10^5)),
                     labels=function(x) x / 100000,
                     limits=c(0,700000))+
  labs(fill="Customer Type")+
  annotate("text",label="casual riders",
           x=4,y=700000)+
  annotate("text",label="max = 523,829 (SAT)",
           x=4,y=650000)+
  annotate("text",label="min = 251,483 (TUE)",
           x=4,y=600000)+
  annotate("text",label="avg = 336,862",
           x=4,y=550000)+
  annotate("text",label="member riders",
           x=6,y=700000)+
  annotate("text",label="max = 423,109 (WED)",
           x=6,y=650000)+
  annotate("text",label="min = 345,273 (SUN)",
           x=6,y=600000)+
  annotate("text",label="avg = 396,824",
           x=6,y=550000)+
  ggtitle("Number of Rides on Different Days of Week",
          subtitle="Source: Cyclistic")
```

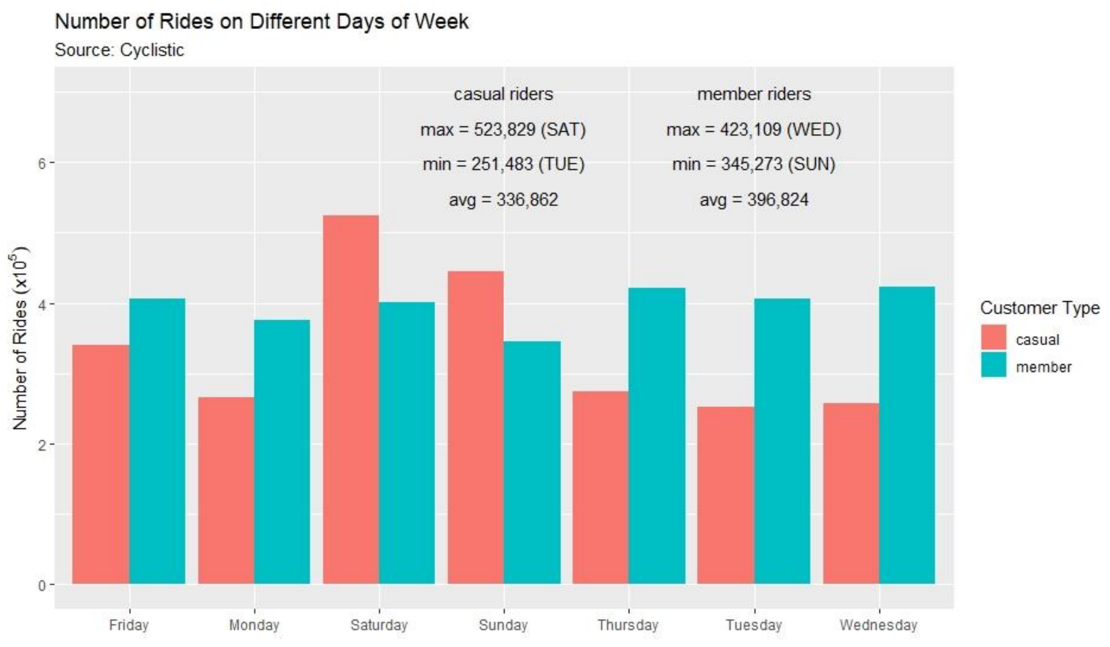


Figure 4.2.5.1: Grouped bar plot showing the total annual number of rides according to different days of the week

From the weekdays bar chart, there is an obvious spike of casual customers on Saturdays and Sundays, the weekend holidays. On the other hand, the number of rides from the member group looks to be constant throughout the week, and drops to the lowest on Sundays.

This observation may be cluing the fact that most of the casual customers are tourists from elsewhere using the bikes for sightseeing trips around Chicago, while most of the member customers are already living and working their regular jobs in Chicago. However, there's not enough data to conclude this at the moment.

To dig a little deeper into the insights about time, another grouped bar chart was created to show total number of rides at different hour of day by the following code:

```
#Code chunk 4.2.5.2: Create a data set with hourly data (Every 1 hour)
alldata_adjh <- alldata_adj %>%
  + mutate(hours=hour(alldata_adj$started_at))

#Plot a bar chart to show hourly customer for the entire year
#!!must create a set of hours in the day fist
hr <- c(0:23)
#Start plotting
ggplot(data=alldata_adjh)+
  geom_bar(mapping=aes(x=hours,fill=member_casual),
           position="dodge")+
  scale_x_continuous(name="Hour of Day",
                     labels=as.character(hr),
                     breaks=hr)+
  scale_y_continuous(name=expression(Number ~ of ~ Rides ~ (x10^5)),
                     labels=function(x) x / 100000)+
  labs(fill="Customer Type")+
  theme(legend.position=c(0.18,0.7))+
  ggtitle("Number of Rides at Different Hour of Day",
          subtitle="Source: Cyclistic")
```

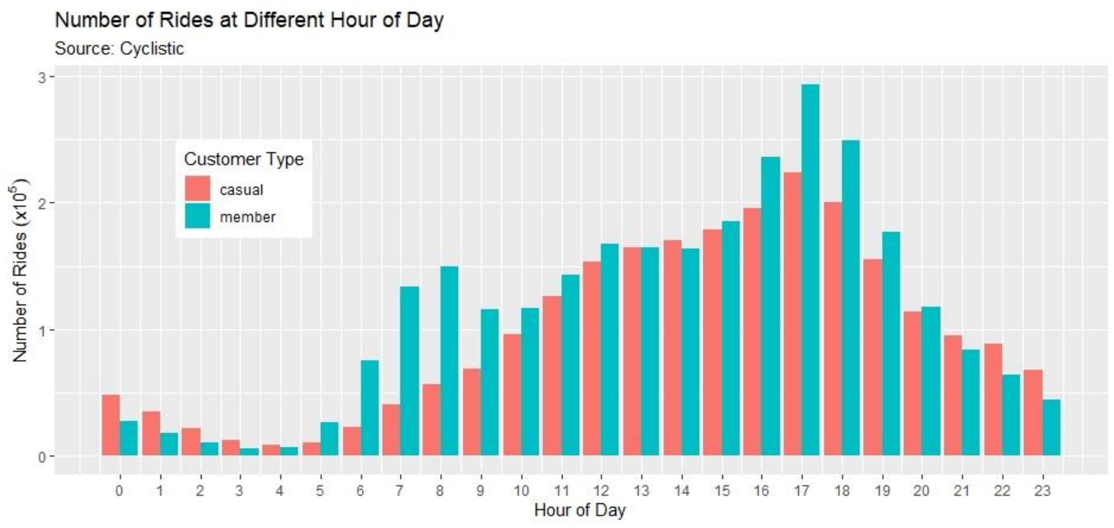


Figure 4.2.5.2: A grouped bar chart visualizing total annual number of rides at each hour of day

From the chart, there are obvious **spikes during 6 - 9 o'clock**, and **during 16 - 19 o'clock** "**the rush hour**" of office workers in **member** customer group. The said spikes don't appear in the casual group.

It is quite clear now that the **member** customers use Cyclistic bikes to commute to and from their workplaces, while the **casual** customers use the bikes for recreational purposes.

4.2.6 Which group of customer is more “flexible” when it comes to taking and parking the bikes?

As shown earlier in the section **3.2.3** about handling NAs, the records with “**not available**” geolocation data are worth studying for they can reveal more insights about customer behaviors.

In this case, the number of records in the **start_lat**, **start_lng**, **end_lat**, and **end_lng** columns in different customer group tells about their flexibility. The more these records appear, the more “flexible” the customers tend to be about where they they take and park the bikes.

The following code created a pie chart summarizing number of NAs in the aforementioned columns in each customer group:

```
#Code chunk 4.2.6.1: Create a pie chart summarizing numbers of NAs for each c
ustomer group

ggplot(data=na_data)+
  geom_bar(mapping=aes(x="",fill=member_casual))+
  coord_polar("y")+
  annotate("text",label="Member",x=0.9,y=60000,size=6)+
  annotate("text",label="48.6%",x=0.9,y=80000,size=6)+
  annotate("text",label="Casual",x=0.9,y=249000,size=6)+
  annotate("text",label="51.4%",x=0.9,y=229000,size=6)+
  theme(plot.background=element_blank(),
        panel.background=element_blank(),
        axis.title=element_blank(),
        axis.ticks=element_blank(),
        axis.text=element_blank(),
        legend.position="none")+
  ggtitle(label="Unidentified Starting or Ending Stations",
          subtitle="Source: Cyclistic")
```

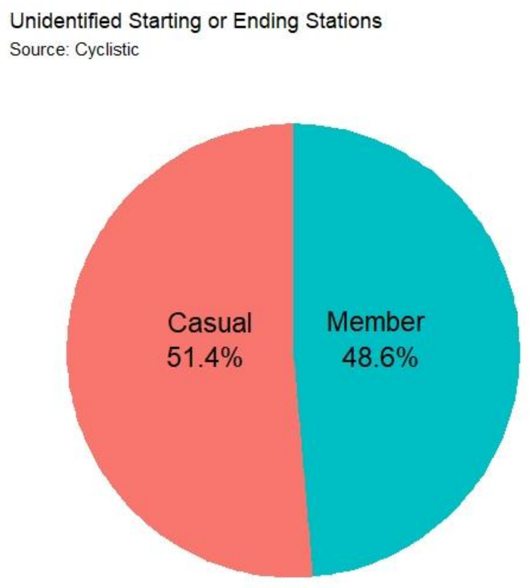


Figure 4.2.6.1: A pie chart summarizing number of unidentified starting or ending stations for each customer group

According to the pie chart, both customer groups are almost at the same level of flexibility. A few percentage of "**casual**" customers is more flexible at taking and parking the bike outside any of the designated station. This is well in accordance with their recreational purposes for using the bikes.

5. "Share"

The deliverable for this section is a presentation file with visualizations and short but clear explanations about insights obtained from them.

The deliverable can be accessed here.

6. "Act"

6.1 Insight summary

According to the analyses, it's clear that the majority in the **casual** and **member** customer groups use Cyclistic's bikes for different purposes. **Members** use the bikes daily to commute to and from their workplaces or their other regular destinations, while **casuals** use the bikes for sightseeing and other recreational purposes.

This conclusion was drawn from the fact that **casual** customers peak during weekends, while the opposite was observed for the **member** group. In addition, peak times for **member** customers are during rush-hour both morning and evening every day. This feature is a signature of office/local workers and was not observed from the **casual** group. Another tell-tale sign to support this conclusion is that the locations of **members**' trips are concentrated in down-town Chicago, which should be where the places of businesses are.

On the other hand, bike usage locations of **casual** customers are more dispersed throughout the city.

With this insight, it can be misleading to ask a question such as "**How to convert casual customers to member customers?**" Asking such a question is like asking a professional runner to buy a swimsuit and go swimming instead of buying a pair of running shoes and go running. It is not entirely impossible, though, but must be very difficult and not worth the effort.

On the other hand, with this insight already distilled from the data, asking such question like "**How to increase the number of customer in general?**" sounds valid and the answers can be much more practical.

6.2 Suggestions/answer to the question/and acations

6.2.1 Expand the branches

If more data about home address of customers were available, it could be possible to tell how many of the casual customers are tourists from other cities or even other countries. If Cylistic's goal is still to convert casual to member customers, it should be possible to realize this idea back in the customers' home cities where Cyclistic might open new branches there. This is because Cyclistic members use the bikes for their regular daily commute. Tourists will also do so when back in their home cities. And the fact that they used Cyclistic's bikes while they were in Chicago says that they liked Cyclistic's services and would willingly become a member and use Cyclistic bikes in their home cities for their daily commutes should the option would be available to them.

6.2.2 More flexible price structure

Cyclistic can also try to convert casual customers for more member customers by offering more flexible membership prices. For example, there should be a membership price for only weekend users which should be much cheaper than the regular users' price.

6.2.3 Unconventional strategy

A strategy which is not very conventional is that Cyclistic can try to force convert the casual customers into member customers by limiting some accesses to the bikes or the other Cyclistic's facilities for casual customers. The accessibility can be fully redeemed when one becomes a member customer. For example, only member customers can take/park bikes from outside designated stations. Casual customers can't do the same and if they don't return the bikes at the designated places, fines will incur. According to the insight from the map chart, both member and casual customers almost enjoy the same level of freedom and flexibility about locations they can take or park bikes. Taking this freedom away from the casual customers will surely has an effect on their decisions about converting to become member customers.

But this path is a risky one and Cyclistic risks loosing a lot of customers if they choose to go down by it.

6.2.4 Tackle marketing challenges with digital media platforms

This is to answer the last of the original questions, Cyclistic can do much about social media strategy with the data in hand. With whatever strategy the stakeholder choose to follow, Cyclistic's social media team should influences potential or existing casual customers to bend on using Cyclistic's bike for their holiday/recreation time in Chicago, then give them special offers for becoming a member.

Cyclistics can also use the knowledge gained from the insights about peak days for casual customers to spread the news about its services. The company can initiate a social media campaign such that casual customers will receive reduced membership prices if they share about their "**weekend experience with Cyclistic**" on their social media platforms.