Cyclistic Case Study

Joseph Brockly-Anderson

2023-08-16

Cyclistic is a (fictional) bike-sharing service operating out of New York City that successfully launched in 2016. Since their opening, their fleet has grown to 5824 bicycles, all of which are geotracked and connected to a network of nearly 700 stations across the city. From one station, bikes can be unlocked for a small fee and returned to any other station within the network.

People Wanting to use the service can either buy single-ride passes, full-day passes, or annual memberships. Those who buy single ride passes or full day passes are referred to as casual riders, Whereas those who purchase annual memberships are referred to as Cyclistic members. This flexibility of Cyclistic's pricing plans gave it a competitive edge when building the general awareness of the brand and marketing to a broad demographic of potential customers.

Financial analysts at Cyclistic have demonstrated successfully that annual members are much more profitable than casual riders, but why do some people buy single-ride and full-day passes while others purchase annual memberships? Does the length of rides have anything to do with this decision? How can analyzing the past twelve month's worth of ride-share data help Cyclistic create a more targeted campaign ad to convert casual riders into Cyclistic members?

By asking the right questions, keeping our sights focused on the business task, gathering and preparing relevant data for processing, analyzing it to look for any trends or helpful insights, and using simple maps and charts to visualize our analysis, I hope to tell a story that can inform Cyclistic's marketing team of potential next steps in their journey to activate annual members.

In order to provide Cyclistic, their marketing team, executive team, and other relevant stakeholders with valuable, accurate insights and data-driven suggestions to convert casual riders into annual members in pursuit of fiscal growth, I'll be following a programmatic process that includes the following steps:

$$Ask - Prepare - Process - Analyze - Share$$

Each process will iteratively build on the former, though there will be times we go back to a certain step and repeat the process again and again for more clarity. We've already begun asking solid questions: why do some people buy single-ride and full-day passes while others purchase annual memberships? Does the length of rides have anything to do with this decision? How can analyzing the past twelve month's worth of ride-share data help Cyclistic create a more targeted campaign ad to convert casual riders into Cyclistic members?

Preparing our data

Calling up the required packages

First, we're going to call in the tidyverse collection for it's `readr`, `dplyr`, `tidyr`, and `ggplot2` packages. We'll also call up a package that will allow us to stack our table together, a package that will help us calculate the distances between two geospatial coordinates, and a package that can render aesthetically pleasing tables to organize our data frame overview.

```r
library(tidyverse) # an avowed collection of data analytics packages
library(readr) # helps us pull in the data
library(dplyr) # helps us organize the data
library(data.table) # helps us organize the data
library(tidyr) # helps us clean the data
library(validate) # this helps us validate the data
library(assertr) # this helps us validate the data
library(ggplot2) # helps us visualize the data
library(geosphere) # helps us calculate distances between geospatial coordinates
```

Calling up our data.

This data was collected and made available by **Motivate LLC** (formerly **Alta Bicycle Share** and also **Motivate International Inc.**) under [license]. The original collection of data sets contains a total of 13 columns, 5723606 rows, and details data from the months of August 2022 to July 2023.

Combining the datasets into one manageable data frame

In order to clean and analyze this data, we first need to combine all of the rows in each of the 12 data sets into one data frame with 13 columns that correspond to the original data sets' variable types. We can do this with the `list()` and `rbindlist()` functions.

```r
trips_df <-
  list(df202208, df202209, df202210,
       df202211, df202212, df202301,
       df202302, df202303, df202304,
       df202305, df202306, df202307) # concatenating each month's data source as a list

df_dirty <- rbindlist(trips_df) # stacking the rows into a single data frame

## The code below is specifically for cleaning up our environment once we've pulled all the data in.
rm(list = ls(pattern = "^df202")) # this removes all data objects that begin with "df202"
rm(trips_df)
## These are not needed because we've combined them into `df_dirty` already.
```

Cleaning the data

Getting acquainted with the data

Now that we've successfully gathered our data and confirmed it's integrity, this section will help us understand its structure, how many rows and columns our data frame contains, the names of our variables, what they measure, and how these data types are codified. Through this process we've discovered that the data frame contains 5,723,606 observations (trips taken on a Cyclistic bicycle within the period between August 2022 and July 2023) with 13 separate attributes which we've included in a data dictionary below.

Data Dictionary

Variable	Meaning
ride_id	A signifier used to uniquely identify each bike ride
rideable_type	Whether the bike ridden was electric, classic, or docked
started_at	The date and time the trip was started
ended_at	The date and time the trip was ended
start_station_name	The name of the station where the trip began

Variable	Meaning
start_station_id	A unique identifier given to the station where the trip began
end_station_name	The name of the station where the trip ended
end_station_id	A unique identifier given to the station where the trip ended
start_lat	The latitude of the starting location
start_lng	The longitude of the starting location
end_lat	The latitude of the ending location
end_lng	The longitude of the ending location
member_casual	Whether a rider is casual or an annual member

Checking for duplicates

Before exploring any trends and correlations in this data set, it's necessary for us to process the data to make sure it contains no incorrect or duplicate observations. We also need to manage any gaps in the information and make sure the formatting and data types remain consistent across our data sources. To do this, we'll first check for duplicate observations with the `duplicated()` function. Luckily, every observation is unique, so we don't have to worry about deleting any values. This has confirmed the uniqueness of data values in the `rideable_type` and `member_casual` columns.

```
sum(duplicated(df_dirty)) # the count of duplicated values
```

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

```
n_distinct(df_dirty$rideable_type) # there should be three types of bike
```

```
## [1] 3
```

```
uniqueN(df_dirty$member_casual) # there should be two types of rider
```

```
## [1] 2
```

Checking for null values

Let's now check for missing values, which we can deal with by thinking critically about their context and making use of professional outreach and statistical modeling techniques dependent on the type of data that we need to find. When we execute these checks, we discover this data set has quite a large amount of null values (a total of 3,600,037), 99.66% of which occur in just 4 columns related to station names and IDs. Another small amount of null values occurs in the `end_lat` and `end_lng` columns.

```
## [1] 3600037
```

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

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

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

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

```
## [1] 868772
```

```
## [1] 868904
```

```
## [1] 925008
```

```
## [1] 925149
```

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

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

```
## [1] 6102
```

```
## [1] 6102
```

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

What could be causing this lack of data? Will these null values jeopardize our analysis or are they sufficiently random? After further investigation, it has come to light that motivate LLC 's bike fleet is outfitted with on board GPS trackers to track their location from within the entire service area. The fleet is technically designated as a "dockless" one. From this information, we can conclude why no values appear in the start station name, start station ID, end station name, and end station ID columns: Neither casual riders nor annual members need to pick up or drop off bikes at designated stations. This makes dealing with these null values significantly simpler in our analysis, but what about the missing values in a few thousand bike trips' ending locations?

Barring a full investigation and keeping within a predetermined allotment of time, it seems reasonable to believe that these missing values could be coming from: * human error on part of the database gathering systems in use at Cyclistic * technical issues with GPS tracking caused by electronic malfunctions * user-related error emerging from a failure to properly conclude bike trips

We've gone ahead and checked the starting and ending coordinates to check if they are correlated for non-missing values. The correlation between starting longitude and ending longitude is fairly weak. There is a moderate correlation between starting latitude and ending latitude, but nothing has pointed us to a belief these null values were systematically incorporated into the data. From this, we're going to assume that these null values are randomized sufficiently enough to avoid complications in our analysis, though further investigation may reveal the necessity to impute them for more accurate analytical results. For now, we've dropped the observations (accounting for ~0.107% of our total) they're associated with out of our analysis.

```
## creating a data frame that filters out the null values
df_pending <- df_dirty %>%
  filter(complete.cases(dplyr::across(c(end_lat, end_lng))))

## checking that they were filtered
sum(is.na(df_pending$end_lat))
```

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

```
sum(is.na(df_pending$end_lng))
```

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

```
## finding the necessary correlations
cor(df_pending$start_lat,
    df_pending$end_lat)
```

```
## [1] 0.566093
```

```
cor(df_pending$start_lng,
    df_pending$end_lng)
```

```
## [1] 0.183352
```

Ensuring proper formats

As a further measure of data processing, we'll make sure our columns are structured correctly. This is an invaluable part of keeping an organized project and will save us time should we encounter bugs in our code during the analysis phase.

Date-time

```
## the POSIXct stores date/time values
is.POSIXct(df_dirty$started_at)
```

```
## [1] TRUE
```

```
is.POSIXct(df_dirty$ended_at)
```

```
## [1] TRUE
```

Strings

```
is.character(df_pending$ride_id)
```

```
## [1] TRUE
```

```
is.character(df_pending$rideable_type)
```

```
## [1] TRUE
```

```
is.character(df_pending$start_station_name)
```

```
## [1] TRUE
```

```
is.character(df_pending$start_station_id)
```

```
## [1] TRUE
```

```
is.character(df_pending$end_station_name)
```

```
## [1] TRUE
```

```
is.character(df_pending$end_station_id)
```

```
## [1] TRUE
```

```
is.character(df_pending$member_casual)
```

```
## [1] TRUE
```

Integers

```
# a "double" atomic vector is commonly referred to as a "float" in other languages and programs
is.double(df_pending$start_lat)
```

```
## [1] TRUE
```

```
is.double(df_pending$start_lng)
```

```
## [1] TRUE
```

```
is.double(df_pending$end_lat)
```

```
## [1] TRUE
```

```
is.double(df_pending$end_lng)
```

```
## [1] TRUE
```

Removing Leading/Trailing Spaces

Leading and trailing spaces in a character string can frustrate our attempts as analysts to capture that data and manipulate it in our analysis. We need some way to find and remove leading and trailing spaces within the data frame that we're working with. This is a solid practice to safeguard the quality of our data before analysis, streamline our troubleshooting process, and avoid any setbacks later on. To do this, we can use the `stringr` package.

```
# the `str_trim()` function only works on string data
df_pending$ride_id <-
  str_trim(df_pending$ride_id,
           "left")
df_pending$ride_id <-
  str_trim(df_pending$ride_id,
           "right")

df_pending$rideable_type <-
  str_trim(df_pending$rideable_type,
           "left")
df_pending$rideable_type <-
  str_trim(df_pending$rideable_type,
           "right")

df_pending$start_station_name <-
  str_trim(df_pending$start_station_name,
           "left")
df_pending$start_station_name <-
  str_trim(df_pending$start_station_name,
           "right")

df_pending$start_station_id <-
  str_trim(df_pending$start_station_id,
           "left")
df_pending$start_station_id <-
  str_trim(df_pending$start_station_id,
           "right")

df_pending$end_station_name <-
  str_trim(df_pending$end_station_name,
           "left")
df_pending$end_station_name <-
  str_trim(df_pending$end_station_name,
           "right")

df_pending$end_station_id <-
  str_trim(df_pending$end_station_id,
           "left")
df_pending$end_station_id <-
  str_trim(df_pending$end_station_id,
           "right")

df_pending$member_casual <-
  str_trim(df_pending$member_casual,
           "left")
df_pending$member_casual <-
```

```
  str_trim(df_pending$member_casual,
           "right")
```

Data Validation

The data were working with can't just be filled with any value, so we should validate each column to ensure that it contains the range of values that we assume actually exists within the data set. To do this, we make use of the `validate()` package (van der Loo and de Jonge, 2021). Upon further investigation, the date/time columns contained a few thousand observations that began before and ended after the time range we're focusing on in this report, so we've filtered them out and created a new data frame. In addition to this, our verification of ending latitudes and longitudes confirmed the existence of ten locations off the southern coast of Ghana (0° latitude, 0° longitude). Clearly, these are erroneous values that don't belong in the data and were recorded or manipulated in error. Using the same logic as when we first filtered these attributes for null values, we're going to disregard the observations that contain these 0 values during our analysis.

```
### setting rules for our variables
## validating `ride_id`
# setting the number of characters to `v_ride_id`
v_ride_id <- nchar(df_pending$ride_id)
sum(v_ride_id == 16) # the number of TRUEs should match the number of rows in the data set (5,717,504)
```

```
## [1] 5717504
```

```
## validating `rideable_type` and `member_casual`
# Creating vectors of allowed values
v_rideable <- c("electric_bike", "classic_bike", "docked_bike")

v_m_c <- c("member", "casual")

# this returns an error if any cell in the column doesn't contain an appropriate value
stopifnot(all(df_pending$rideable_type %in% v_rideable))

stopifnot(all(df_pending$member_casual %in% v_m_c))

## validating `started_at` and `ended_at`
# determining minimum and maximum values
max(df_pending$started_at)
```

```
## [1] "2023-07-31 23:59:56 UTC"
```

```
min(df_pending$started_at)
```

```
## [1] "2022-08-01 UTC"
```

```
max(df_pending$ended_at)
```

```
## [1] "2023-08-01 20:40:50 UTC"
```

```
sum(df_pending$started_at < "2022-08-01 00:00:00")
```

```
## [1] 1391
```

```
sum(df_pending$ended_at > "2023-07-31 11:59:59")
```

```
## [1] 5694
```

```
# defining the date/time range
start_date <- as.POSIXct("2022-08-01 00:00:00")
```

```r
end_date <- as.POSIXct("2023-07-31 11:59:59")


# filtering out unecessary observations
df_filtered <- subset(df_pending,
                      started_at >= start_date &
                        started_at <= end_date &
                        ended_at >= start_date &
                        ended_at <= end_date)

# making sure the filter removed the appropriate number of observations
count_pending <- sum(!is.na(df_pending$ride_id))
count_filtered <- sum(!is.na(df_filtered$ride_id))
rows_started <- sum(df_pending$started_at < "2022-08-01 00:00:00")
rows_ended <- sum(df_pending$ended_at > "2023-07-31 11:59:59")
print(if(count_pending - count_filtered ==  rows_started + rows_ended) TRUE)
```

```
## [1] TRUE
```

```r
### verifying our location data describes Chicago
# defining Chicago's location
chicago_lat_min <- 41.59
chicago_lat_max <- 42.19
chicago_lng_min <- -87.93
chicago_lng_max <- -87.53

# verifying coordinates
within_chicago <- with(df_filtered,
  start_lat >= chicago_lat_min & start_lat <= chicago_lat_max &
  start_lng >= chicago_lng_min & start_lng <= chicago_lng_max &
  end_lat >= chicago_lat_min & end_lat <= chicago_lat_max &
  end_lng >= chicago_lng_min & end_lng <= chicago_lng_max)

# filtering out potential outliers
df_chicago <- df_filtered[within_chicago, ]

# finding outliers
maximum_starting_latitude <-
  max(df_filtered$start_lat)
message <-
  paste("Maximum Starting Latitude:",
        maximum_starting_latitude)
cat(message, "\n")
```

```
## Maximum Starting Latitude: 42.07
```

```r
##
maximum_starting_longitude <-
  max(df_filtered$start_lng)
message <-
  paste("Maximum Starting Longitude:",
        maximum_starting_longitude)
cat(message, "\n")
```

```
## Maximum Starting Longitude: -87.52
```

```r
##
maximum_ending_latitude <-
  max(df_filtered$end_lat)
message <-
  paste("Maximum Ending Latitude:",
        maximum_ending_latitude)
cat(message, "\n")
```

```
## Maximum Ending Latitude: 42.18
```

```r
##
maximum_ending_longitude <-
  max(df_filtered$end_lng)
message <-
  paste("Maximum Ending Longitude:",
        maximum_ending_longitude)
cat(message, "\n")
```

```
## Maximum Ending Longitude: 0
```

```r
##
minimum_starting_latitude <-
  min(df_filtered$start_lat)
message <-
  paste("Minimum Starting Latitude:",
        minimum_starting_latitude)
cat(message, "\n")
```

```
## Minimum Starting Latitude: 41.64
```

```r
##
minimum_starting_longitude <-
  min(df_filtered$start_lng)
message <-
  paste("Minimum Starting Longitude:",
        minimum_starting_longitude)
cat(message, "\n")
```

```
## Minimum Starting Longitude: -87.92
```

```r
##
minimum_ending_latitude <-
  min(df_filtered$end_lat)
message <-
  paste("Minimum Ending Latitude:",
        minimum_ending_latitude)
cat(message, "\n")
```

```
## Minimum Ending Latitude: 0
```

```r
##
minimum_ending_longitude <-
  min(df_filtered$end_lng)
message <-
  paste("Minimum Ending Longitude:",
        minimum_ending_longitude)
cat(message, "\n")
```

```
## Minimum Ending Longitude: -88.16
```

```
# outliers
outside_chicago <- with(df_filtered,
  end_lat == 0 & end_lng <= 0)

df_outliers <- df_filtered[outside_chicago, ]
```

Citations

MPJ van der Loo and E de Jonge (2021). Data Validation Infrastructure for R. Journal of Statistical Software, 97(10) paper.