Cruise Automation Takehome EDA

Andrew Wilson

10/31/2018

Prompt

As the newest data analytics hire at an autonomous vehicle startup (Ryde Automation), you have been given an example dataset (~1000 rows, 8 columns) and are asked to suggest some business actions for Ryde based on your conclusions. We encourage you to explore the data and move in any direction that you think will benefit the company the most. If you are running short on time, please document your thought process in words and explain in detail how you would proceed if given additional time.

There is a small intentional error in the dataset to look out for! (Let us know in the report if you find it and fix it!)

This take-home exercise is meant to be open-ended. Please feel free to use whatever tools you are most comfortable with and please submit your final report in any format that best conveys your findings.

Disclaimer: This dataset has no affiliation with the data at Cruise Automation.

Table schema description:

- *user_id: identification number assigned to each user*
- *car_id: name of each car used for the ride*
- *start_time: start time of the ride in YYYY-MM-DD HH:MM:SS*
- *end_time: end time of the ride in YYYY-MM-DD HH:MM:SS*
- *num_riders: number of passengers in the ride*
- *region: the region that the car is operating in (origin)*
- *num_near_misses: the number of times that the car is close to getting into an accident during the ride*
- *price: the price that the rider paid to get on the ride*
- *rating: the rating (1 through 5) that the rider gave the ride (5 is best)*

Executive summary

Methodology

Setup, load, and clean data

Setup

The first step is to set up the environment properly:

- Remove all environment variables so that program always has the same end result given the same input data
- Set up default options for publishing from R markdown to PDF
- Load all necessary libraries, including ability to code Python in R Markdown
- Set up a custom theme for all ggplot charts

- Change some of the default settings

```
## clean workspace
rm(list=ls(all=TRUE))

# knitr setup
knitr::opts_chunk$set(
    echo = TRUE,
    message = FALSE,
    warning = FALSE,
    error = TRUE,
    comment = "#>",
    tidy.opts = list(width.cutoff = 80)
)
 # keep output with code (or not if FALSE)
knitr::opts_chunk$set(tidy.opts=list(width.cutoff=80))

# load libraries
library(scales)
library(GGally)
library(RColorBrewer)
library(readxl)
library(knitr)
library(scales)
library(reticulate)
use_python("/Users/andrewwilson/anaconda/bin/python", required=TRUE)
#library(officer)
library(lubridate)
library(tidyverse) # make sure piping function loaded last
#library(magrittr) # always have piping available


## set cruise colors
cruise_colors <- c("#fc553c", "#553d65")

## set custom theme
custom_theme <- function(base_size = 14, base_family = "Helvetica", ...){
  modifyList(theme_minimal(base_size = base_size, base_family = base_family),
             list(
               legend.background =  element_rect(color = "black")
             )
  )
}
theme_set(custom_theme())

## disable scientific notation
options(scipen=999)
```

Data import

Importing data is relatively straightforward. Simply set up the appropriate directories and read in the Cruise Takehome data. It's important to look at what fields/features are available before beginning to clean the data.

```
## set up data directories, files names, and paths
#root_dir <- "/Users/andrewwilson/Documents-backup/Projects/cruise-take-home/"
data_file_name <- "final_analytics_takehome.xlsx"
data_path <- str_c("data/", data_file_name)

## import data from Excel
df_raw <- read_excel(data_path)

# check out what data we're working with
glimpse(df_raw)
```

```
#> Observations: 1,033
#> Variables: 9
#> $ user_id         <dbl> 9, 12, 3, 10, 9, 15, 15, 19, 14, 8, 16, 20, 11...
#> $ car_id          <chr> "spiderman", "superman", "hulk", "spiderman", ...
#> $ start_time      <dttm> 2018-10-02 03:00:21, 2018-10-02 03:01:30, 201...
#> $ end_time        <dttm> 2018-10-02 03:08:19, 2018-10-02 03:09:16, 201...
#> $ num_riders      <dbl> 3, 2, 2, 3, 4, 3, 5, 1, 3, 4, 3, 2, 1, 2, 5, 5...
#> $ region          <chr> "sf", "sf", "sf", "sf", "sf", "south sf", "sf"...
#> $ num_near_misses <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
#> $ price           <dbl> 3.25, 2.95, 3.22, 2.29, 2.93, 2.48, 3.42, 5.30...
#> $ rating          <dbl> 5, 5, 5, 5, 1, 5, 5, 5, 3, 5, 3, 5, 5, 3, 5, 5...
```

Data cleaning

Cleaning data happens iteratively during the exploration process.

Data exploration in this phase exposed some interesting characteristics of the data set. Here are some general findings about the shape of the data:

- All rides are from October 2, 2018 at 3:00:21 to 23:59:21 on the same day.
- There is no pricing between $3.50 and $5.00
- There are no ride times between 30 and 60 minutes

Features can be added that are clear transformations from the existing data. I added a duration feature called `ride_mins`, and two categorical features to label the time and price as "short"/"long" and "cheap"/"expensive", respectively. Others that aren't necessary can be removed.

Anomalies and/or outliers should first be checked to make sure they aren't in fact true. If they are, there could be some interesting learnings there. If it's an obvious error, this data can be removed from the analysis. Some of the ride_mins were actually negative, so I removed this since it's physically impossible.

Factors are useful data types in R, especially when it comes to visualizing bar charts. Some of the categorical variables were converted into factors.

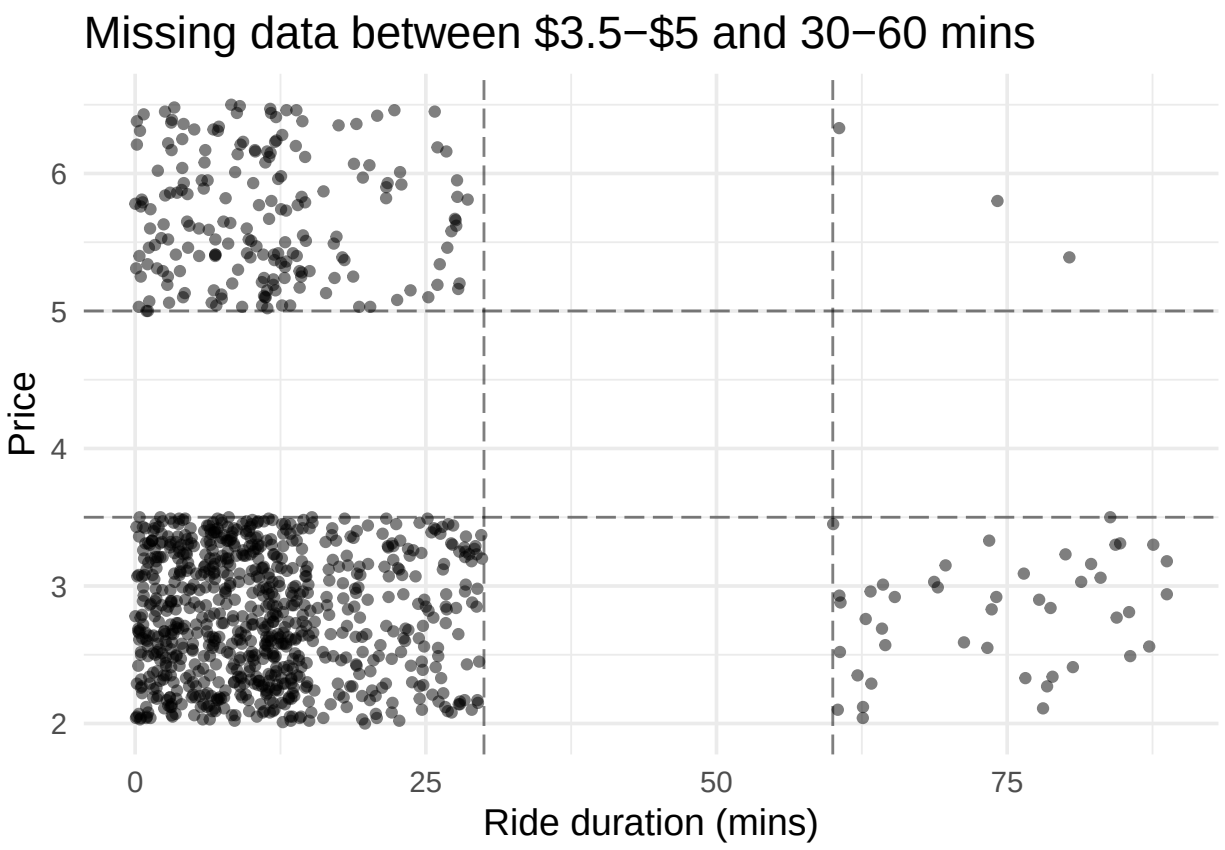
It's good to take another peek at the data set once it's been cleaned properly.

```
## initial data cleaning
df <- df_raw %>%

  # change characters to factors
  mutate_if(is.character, as.factor) %>%
  mutate(user_id = as.factor(user_id)) %>%
  mutate(num_riders = as.factor(num_riders)) %>%
  mutate(rating = as.factor(rating)) %>%
```

```r
  # calculate ride time in minutes
  mutate(ride_mins = as.numeric(end_time - start_time)/60) %>%

  # eliminate data points that have negative ride time
  filter(ride_mins>0) %>%

  # add categorical labels for long, short, cheap, and expensive rides
  mutate(time_cat = ifelse(ride_mins < 45, "short", "long")) %>%
  mutate(price_cat = ifelse(price < 4, "cheap", "expensive"))

# Check out cleansed data
glimpse(df)
```

```
#> Observations: 1,017
#> Variables: 12
#> $ user_id         <fct> 9, 12, 3, 10, 9, 15, 15, 19, 14, 8, 16, 20, 11...
#> $ car_id          <fct> spiderman, superman, hulk, spiderman, scarecro...
#> $ start_time      <dttm> 2018-10-02 03:00:21, 2018-10-02 03:01:30, 201...
#> $ end_time        <dttm> 2018-10-02 03:08:19, 2018-10-02 03:09:16, 201...
#> $ num_riders      <fct> 3, 2, 2, 3, 4, 3, 5, 1, 3, 4, 3, 2, 1, 2, 5, 5...
#> $ region          <fct> sf, sf, sf, sf, sf, south sf, sf, sf, sf, sout...
#> $ num_near_misses <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
#> $ price           <dbl> 3.25, 2.95, 3.22, 2.29, 2.93, 2.48, 3.42, 5.30...
#> $ rating          <fct> 5, 5, 5, 5, 1, 5, 5, 5, 3, 5, 3, 5, 5, 3, 5, 5...
#> $ ride_mins       <dbl> 7.9666667, 7.7666667, 7.5000000, 13.4333333, 6...
#> $ time_cat        <chr> "short", "short", "short", "short", "long", "s...
#> $ price_cat       <chr> "cheap", "cheap", "cheap", "cheap", "cheap", "...
```

```r
# visualize missing price and duration values
ggplot(df, aes(ride_mins, price)) +
  geom_point(alpha=0.5) +
  geom_vline(xintercept=c(30, 60), linetype=5, alpha=0.5) +
  geom_hline(yintercept=c(3.5, 5), linetype=5, alpha=0.5) +
  labs(x="Ride duration (mins)",
     y="Price",
     title="Missing data between $3.5-$5 and 30-60 mins")
```



Safety

A top priority for Ryde Automation is to build self driving cars that are **safe**. We'll start by examining the number of near misses (`num_near_misses`). The goal is to better understand when and how near misses occur in the data, to infer the causes, and to suggest possible actions to reduce the number of near misses in the future.

Some notable findings:

- All near misses happened on "long" rides (> 1 hour)
- All near misses happened in SF (not South SF)
- Near misses are always rated as 1 or 2 stars (creating a correlation)
- Superman has the most near misses

Tables to demonstrate results:

```
# filter data frame to get only rides with near misses
df_misses <- df %>%
  select(num_near_misses, ride_mins, region, rating, car_id) %>%
  filter(num_near_misses > 0) %>%
  arrange(desc(num_near_misses)) %>%
  mutate(ride_mins = round(ride_mins))
```

```
kable(df_misses, booktabs = TRUE,
      caption="Near misses are on long rides, in SF, and are rated at 1 or 2 stars")
```

Table 1: Near misses are on long rides, in SF, and are rated at 1 or 2 stars

num_near_misses	ride_mins	region	rating	car_id
5	84	sf	1	superman
3	63	sf	1	superman
2	69	sf	1	superman
2	65	sf	1	superman
2	83	sf	1	superman
2	78	sf	1	superman
2	73	sf	1	superman
1	81	sf	1	robin
1	79	sf	2	superman
1	65	sf	1	joker
1	79	sf	2	spiderman
1	80	sf	2	batman
1	74	sf	1	scarecrow
1	64	sf	2	spiderman
1	76	sf	2	ironman
1	61	sf	1	superman
1	61	sf	1	superman
1	78	sf	1	superman

```
# count number of misses per car
df_misses_cars <- df_misses %>% group_by(car_id) %>%
  summarize(total_near_misses = sum(num_near_misses),
            percentage_near_misses = percent(total_near_misses/sum(df_misses$num_near_misses))) %>%
  arrange(desc(total_near_misses))

kable(df_misses_cars, booktabs = TRUE,
      caption="Superman has three quarters of the near misses")
```

Table 2: Superman has three quarters of the near misses

car_id	total_near_misses	percentage_near_misses
superman	22	75.9%
spiderman	2	6.90%
batman	1	3.45%
ironman	1	3.45%
joker	1	3.45%
robin	1	3.45%
scarecrow	1	3.45%

Business actions

Ryde Automation should make an effort to better understand *why* near misses occur. Since most of the rides are in SF (not South SF), it makes sense that all near misses occur there. It also makes sense that near

misses always occur on "long" rides (>1hr), but it should be investigated given a larger data set to see if this is always the case. Not only are near misses endangering your passengers, but they are also the cause of 85% of 1 star ratings (13/15). The best way to understand why near misses occur is to look more carefully at the `Superman` car, since it is responsible for 3/4 of all near misses. Given more data about this car, we could dig in to better understand the underlying issues to prevent them from happening in other cars.

Ratings

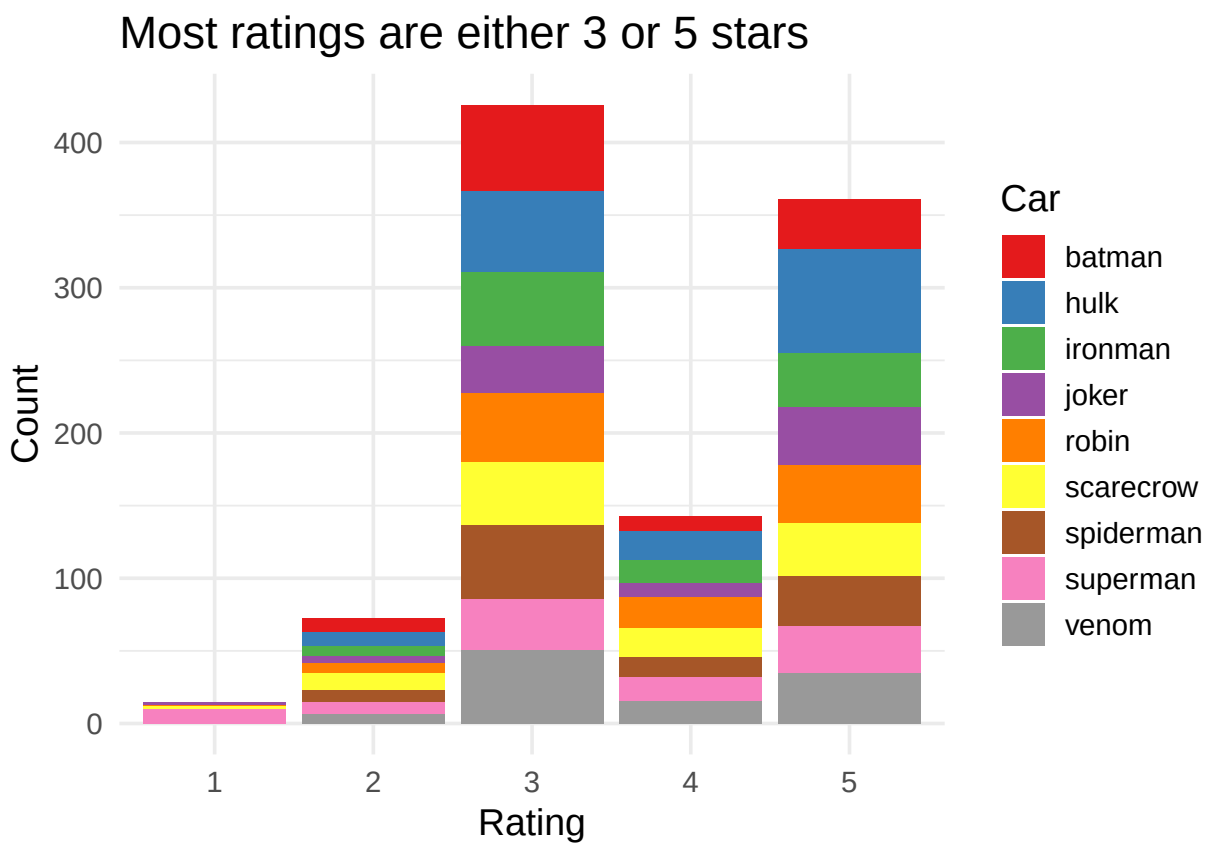
Ratings are a crucial indicator of the quality of service at Ryde Automation. They directly relate to customer retention, a critical business metric for a new ride sharing company. We'll examine this feature next, with the goal of better understanding how ratings are distributed, what causes a good/bad rating, and how to improve ratings in the future.

Some notable findings:

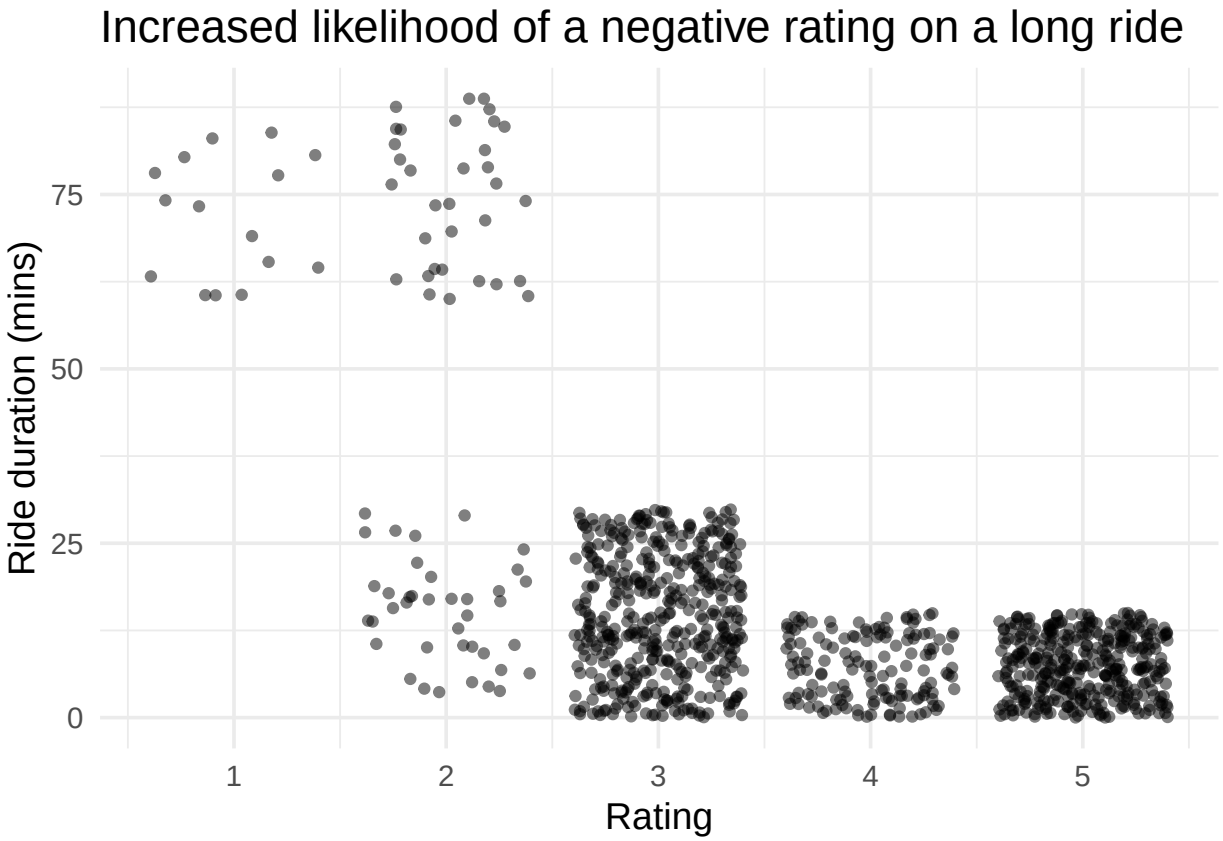
- Most ratings are either 3 or 5 stars
- Superman, as previously discussed, has most of the 1 star ratings (likely due to its numerous near misses)
- Hulk has the most 5 star ratings
- Most negaitve reviews happen on "long" rides (there is a slight negative correlation between `rating` and `ride_mins`)

Charts to demonstrate:

```
ggplot(df, aes(rating, fill = car_id)) +
  geom_bar() +
  scale_fill_brewer(palette="Set1") +
  labs(x="Rating",
       y="Count",
       fill="Car",
       title="Most ratings are either 3 or 5 stars")
```




```
ggplot(df, aes(as.integer(rating), ride_mins)) +
  geom_jitter(alpha = 0.5) +
  labs(x="Rating",
     y="Ride duration (mins)",
     title="Increased likelihood of a negative rating on a long ride")
```



Utilization

Utilization is a key metric for a ride sharing company, as it determines how much free capacity is available to fill up. This enables the company to increase revenues without increasing capital expenditures (extra profit!). In this case, I calculated utilization as the number of used seats divided by the total number of seats (5 per car) for any given time period.

Unfortunately, calculating utilization is not always as easy as it seems, and it's certainly the case with this data set. There are a few problems that we'll have to deal with in order to get to an accurate utilization number.

```
df_u <- df %>%
  select(car_id, start_time, end_time, ride_mins, num_riders) %>%
  arrange(car_id)

car_id <- df_u$car_id
start_time <- period_to_seconds(seconds(df_u$start_time))
end_time <- period_to_seconds(seconds(df_u$end_time))
ride_mins <- df_u$ride_mins
num_riders <- as.integer(df_u$num_riders)
```

```
# create an array of rides dictionaries
rides = []
for c, s, e, m, n in zip(r.car_id, r.start_time, r.end_time, r.ride_mins, r.num_riders):
  rides.append({
    "car_id": c,
```

```
    "start_time": s,
    "end_time": e,
    "ride_mins": m,
    "num_riders": n
  })
# create a function to find average of a block of rides with > 0 riders
def avg_num_riders(rides):
  weighted_riders = 0
  total_mins = 0
  for i in range(len(rides)):
    #print(rides[i]["ride_mins"], rides[i]["num_riders"])
    weighted_riders += rides[i]["ride_mins"] * rides[i]["num_riders"]
    total_mins += rides[i]["ride_mins"]
  average_riders = weighted_riders / total_mins
  #print(average_riders)
  return average_riders
```

```
# iterate over rides array to build a new array with no overlap
rides_flattened = []
ride_sequence = [rides[0]]
earliest_start_time = rides[0]["start_time"]
latest_end_time = rides[0]["end_time"]
for i in range(len(rides)-1):
  if rides[i+1]["start_time"] < latest_end_time \
  and rides[i+1]["car_id"] == rides[i]["car_id"]:
    ride_sequence.append(rides[i+1])
    latest_end_time = max(latest_end_time, rides[i+1]["end_time"])
  else:
    avg_riders_for_sequence = avg_num_riders(ride_sequence)
    rides_flattened.append({
      "car_id": rides[i]["car_id"],
      "start_time": earliest_start_time,
      "end_time": latest_end_time,
      "num_riders": avg_riders_for_sequence
    })
    if rides[i+1]["car_id"] == rides[i]["car_id"]:
      rides_flattened.append({
        "car_id": rides[i]["car_id"],
        "start_time": latest_end_time,
        "end_time": rides[i+1]["start_time"],
        "num_riders": 0.0
      })
    earliest_start_time = rides[i+1]["start_time"]
    latest_end_time = rides[i+1]["end_time"]
    ride_sequence = [rides[i+1]]

print(len(rides_flattened))
# create individual lists to transform back into an R dataframe
```

```
#> 558
```

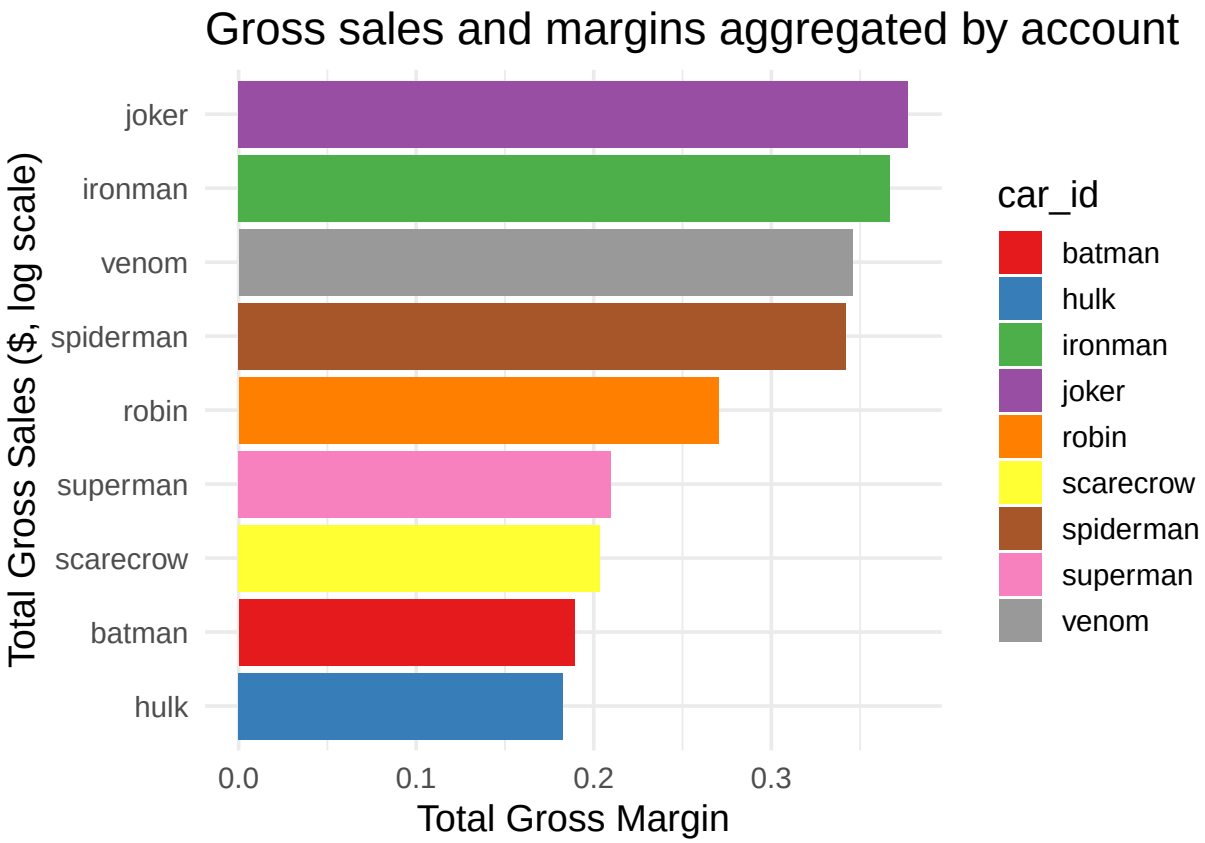
```
car_id_flattened = []
start_time_flattened = []
end_time_flattened = []
```

```
num_riders_flattened = []
for i in range(len(rides_flattened)):
  car_id_flattened.append(rides_flattened[i]["car_id"])
  start_time_flattened.append(rides_flattened[i]["start_time"])
  end_time_flattened.append(rides_flattened[i]["end_time"])
  num_riders_flattened.append(rides_flattened[i]["num_riders"])
```

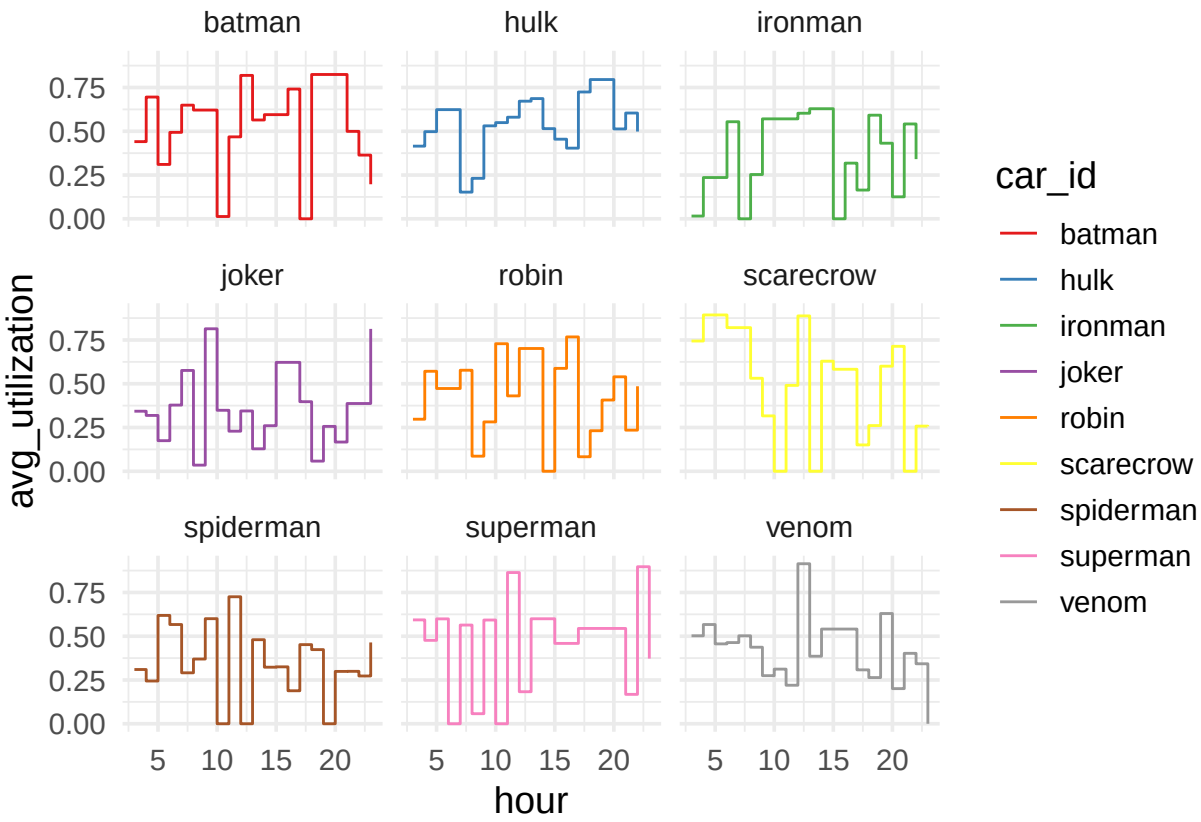
```
start_time_flattened_POSXct = as.POSIXct(py$start_time_flattened,
                                         tz="UTC",
                                         origin = "1970-01-01")
end_time_flattened_POSXct = as.POSIXct(py$end_time_flattened,
                                       tz="UTC",
                                       origin = "1970-01-01")

df_uf <- data.frame("car_id" = py$car_id_flattened,
                    "start_time" = start_time_flattened_POSXct,
                    "end_time" = end_time_flattened_POSXct,
                    "num_riders" = py$num_riders_flattened) %>%
  mutate(ride_mins = as.double(end_time - start_time)/60)

glimpse(df_uf)
```

```
#> Observations: 558
#> Variables: 5
#> $ car_id     <fct> batman, batman, batman, batman, batman, batman, bat...
#> $ start_time <dttm> 2018-10-02 03:04:27, 2018-10-02 03:16:24, 2018-10-...
#> $ end_time   <dttm> 2018-10-02 03:16:24, 2018-10-02 03:21:21, 2018-10-...
#> $ num_riders <dbl> 4.000000, 0.000000, 2.567112, 0.000000, 1.000000, 0...
#> $ ride_mins  <dbl> 11.9500000, 4.9500000, 27.1166667, 4.3333333, 9.000...
```

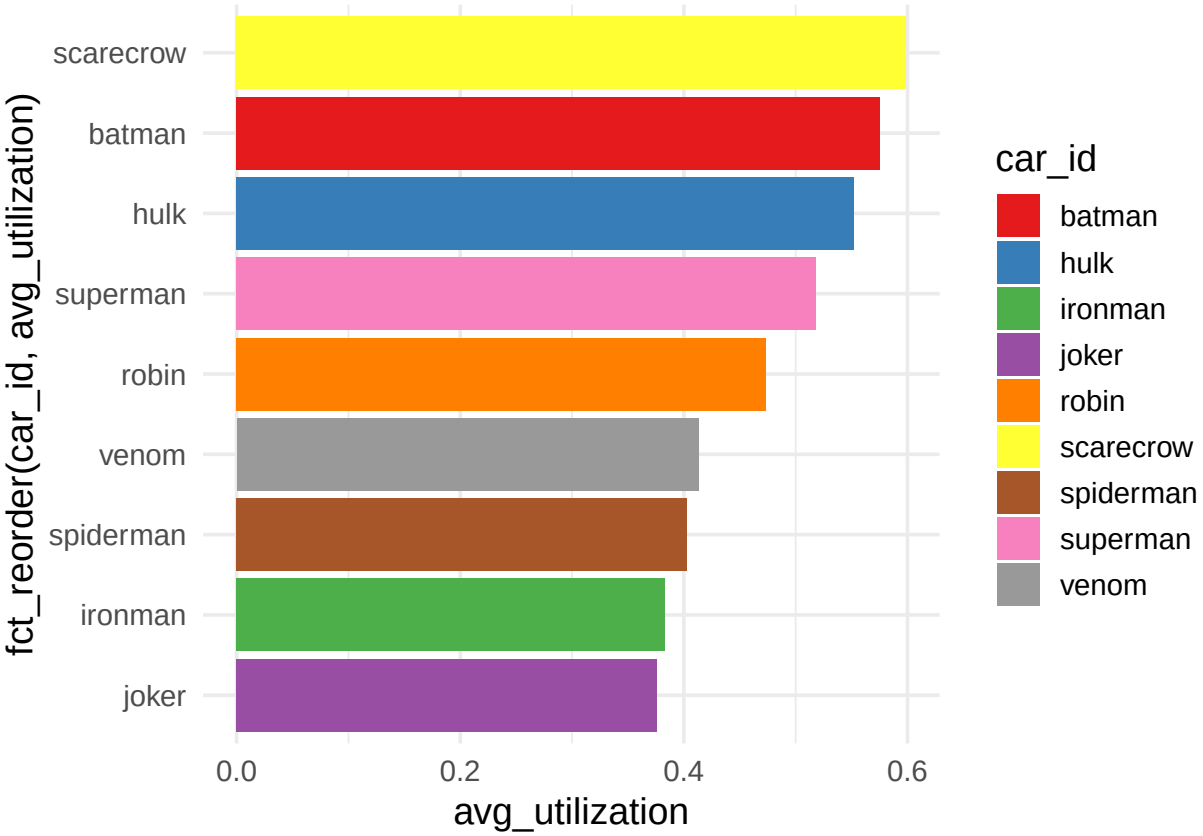
```
overall_average_utilization <- df_uf %>%
  summarize(((sum(num_riders * ride_mins)) / (sum(ride_mins)))/5) %>%
  as.double()

# average utilization per car
df_uf_car <- df_uf %>%
  group_by(car_id) %>%
  summarize(avg_num_riders = (sum(num_riders * ride_mins)) / (sum(ride_mins)),
            avg_utilization = avg_num_riders / 5,
            percent_time_between_rides = sum(ride_mins[num_riders==0])/sum(ride_mins))

ggplot(df_uf_car, aes(fct_reorder(car_id, percent_time_between_rides), percent_time_between_rides, fill=
  geom_bar(stat="identity") +
  coord_flip() +
  scale_fill_brewer(palette="Set1") +
  labs(x="Total Gross Sales ($, log scale)",
       y="Total Gross Margin",
       color="Margin (%)",
       title="Gross sales and margins aggregated by account")
```




```
# average utilization per hour per car (based on total # of seats available)
df_uf %>% mutate(hour = hour(start_time)) %>%
  group_by(car_id, hour) %>%
  summarize(avg_num_riders = (sum(num_riders * ride_mins)) / (sum(ride_mins)),
            avg_utilization = avg_num_riders / 5) %>%
  ungroup() %>%
  ggplot(aes(hour, avg_utilization, color=car_id)) +
    geom_step() +
    facet_wrap(~car_id) +
    scale_color_brewer(palette="Set1")
```




```
# average utilization per
ggplot(df_uf_car, aes(fct_reorder(car_id, avg_utilization), avg_utilization, fill=car_id)) +
  geom_bar(stat="identity") +
  coord_flip() +
  scale_fill_brewer(palette="Set1")
```



Pricing

Recommended actions and conclusion

Next steps